Table of Contents

Poster Proceedings KI2004

Designing and Simulating Individual Teleo-Reactive Agents

Krysia Broda and Christopher John Hogger

Department of Computing, Imperial College London
kb@doc.ic.ac.uk, cjh@doc.ic.ac.uk

Abstract. A method is presented for designing an individual teleo-reactive agent, based upon discounted-reward evaluation of policy-restricted subgraphs of complete situation-graphs. The main feature of the method is that it exploits explicit and definite associations of the agent's perceptions with states. The combinatorial burden that would potentially ensue from such associations can be ameliorated by suitable use of abstraction. For various *BlocksWorld* examples, the agent's predicted behaviour is compared with simulation results to provide insight into the method's power and scalability.

1 Introduction

Teleo-reactive agents [10] react to their perceptions of the world by obeying an internal program (or policy) mapping perceptions to actions. The simplest policy structure is a set of mutually-exclusive production rules of the form $perception \rightarrow action$, usually intended to control durative behaviour: given some current perception the agent performs the corresponding action until acquiring a new perception, whereupon it reacts likewise to that.

Such an agent may or may not have sufficient perceptive capability to know, at any instant, the entire state of the world. An agent of the kind described in [11] is presumed at least capable of perceiving an intended goal state whenever that state arises, and is accordingly designed with that capability in mind. Its program includes an explicit test for the goal state, whilst the nature and ordering of its rules are inferred by reductive analysis of that test. Its goal-orientedness is thus explicit in the program.

A teleo-reactive agent of the kind studied in this paper is, by contrast, presumed incapable of perceiving the entirety of any state, whether a goal state or otherwise. In particular, its program contains no rule specifically associated with a goal. The design process now relies not upon goal-reductive analysis but upon comparing the extents to which alternative programs dispose the agent towards achieving a goal – as judged, for instance, by a discounted-reward principle. A program identified on this basis is *implicitly* goal-oriented.

Our design process is based upon graphs that relate combinations of perception and objective state, and so differs from approaches that employ graphs relating perceptions alone. These typically estimate the distribution of states

associated with a current perception using some species of history, whether of past actions, past states or past perceptions. The key presumption there is that the history is *reliable* in that no changes to the state ever occur except by the known actions of the agent. By contrast, our focus is specifically upon the opposite case in which the world may undergo unpredictable exogenous changes while the agent is pursuing its goal. These changes include, for example, results of failed actions and mistaken perceptions.

We first outline our basic formulation for teleo-reactive agents using objective states and perceptions, which are pairwise combined into situations. We then introduce a method for computing a value for a given policy using the discounted reward principle and show that the results obtained correspond well with simulation results (see Case Studies 1-4). In order that our method should scale to reasonably sized-problems we use a method of *abstraction*. We make an assumption that when there is a large number of situations a good policy can be found by aggregating situations into classes and treating each such class as a single abstract (or generic) situation. The results from the example in Case Study 5 vindicate this assumption.

Our core notions are formulated using the following notation. The set of all objective states that the world may assume is denoted by $\mathcal{O}$; the set of all perceptions that the agent may have of the world is denoted by $\mathcal{P}$; the set of all actions that the agent may take is denoted by $\mathcal{A}$. For each state $o \in \mathcal{O}$, the perceptions that the agent may have of o form some subset $P(o)$ of $\mathcal{P}$, and for any perception $p \in P(o)$ the actions that the agent may take in reacting to p for some subset $A(p)$ of $\mathcal{A}$. A *policy* for the agent is any total function $f : \mathcal{P} \rightarrow \mathcal{A}$ satisfying $\forall p \in \mathcal{P},\ f(p) \in A(p)$.

For any $o \in \mathcal{O}$ and $p \in P(o)$, the pair (o, p) is called a *situation*. A *goal* is a situation that the agent is required, by the designer, to achieve. The fundamental problem in identifying a suitable policy is the incompleteness of perceptions as state descriptors. If they fully described the states then any situation (o, p) could be contracted simply to o, and the design process would need only to compare conventional state-transition graphs. However, the realistic position is that each perception contains only limited knowledge of the world. The problem is therefore how to optimize, for a goal incorporating a state, an agent that (generally) cannot recognize that state.

The agents that we consider do not necessarily have any memory capability. However, our approach easily allows for such capability, by including memory as additional perceptions. Actions that affect such perceptions can be made available to the agents. In that way suitable policies for those problems commonly addressed in the literature, such as the "tiger problem" [7] can be modelled in our framework.

Example 1: *BlocksWorld* Formulation This example, like all others in this paper, employs *BlocksWorld* to illustrate the ideas involved. It is strongly representative of state-transition systems in a wide class of domains. This world comprises a surface and some number (here 4) of identical blocks. At most one

such block may be held by the agent, whilst the other blocks are arranged as some configuration of towers standing upon the surface. A goal for this example might be the situation in which the configuration comprises a 4-tower and in which the agent is perceiving this tower. A configuration is represented by a list of integers each denoting the height (>0) of a tower. Figure 1(a) shows the possible configurations, each determining a state o. The states are named $1, \ldots, 8$ for brevity. In states 1-5 no block is being held, whilst in states 6-8 one block is being held.

(a) states

o		$P(o)$
1	[2, 2]	$\{e, i\}$
2	[1, 1, 1, 1]	$\{d, i\}$
3	[1, 1, 2]	$\{d, e, i\}$
4	[1, 3]	$\{d, f, i\}$
5	[4]	$\{g, i\}$
6	[1, 1, 1]	$\{a, h\}$
7	[1, 2]	$\{a, b, h\}$
8	[3]	$\{c, h\}$

(b) perceptions

p		$A(p)$
a	$[s1, h]$	{l, w}
b	$[s2, h]$	{l, w}
c	$[s3, h]$	{l, w}
d	$[s1, nh]$	{k, w}
e	$[s2, nh]$	{k, w}
f	$[s3, nh]$	{k, w}
g	$[s4, nh]$	{k, w}
h	$[s0, h]$	{l, w}
i	$[s0, nh]$	{w}

Fig. 1. States, perceptions and actions

A perception is represented by a list $[S, H]$, where S expresses what the agent is seeing and H expresses its holding status. The domain of S is $\{s0,\ s1,\ s2,\ s3,\ s4\}$ in which $s0$ denotes "seeing the surface" and sn $(n > 0)$ denotes "seeing an n-tower". The domain of H is $\{h,\ nh\}$ in which h denotes "holding a block" and nh denotes "not holding a block". Figure 1(b) shows the possible perceptions, named $a, \ldots, i$ for brevity. Figure 1(a) also shows alongside each state o the set $P(o)$ of possible perceptions. From this we can infer, for instance, that state 5 occurs in just two situations $(5, g)$ and $(5, i)$. Declaring perceptions and actions presumes certain physical characteristics of the agent. Here, we presume three possible actions – pick (k), place (l) and wander (w). In a k-action the agent removes the top block from a tower it is seeing, and afterwards holds that block and sees the resulting tower (or the surface, as appropriate). In an l-action the agent places a block it is holding upon whatever it sees (the surface or some tower) and afterwards is not holding a block and sees the resulting tower. A w-action merely updates the agent's perception without altering the state. Figure 1(b) shows alongside each perception p the set $A(p)$ of possible actions. The data in Fig. 1 and the assumed effects of actions jointly determine the transitions the agent could make between situations if no policy were employed to restrict its actions, i.e. if it were free to behave with maximum non-determinism. Figure 2 depicts all these transitions as an *unrestricted graph*, denoted generally by G. For brevity, each situation (o, p) is displayed there as op.

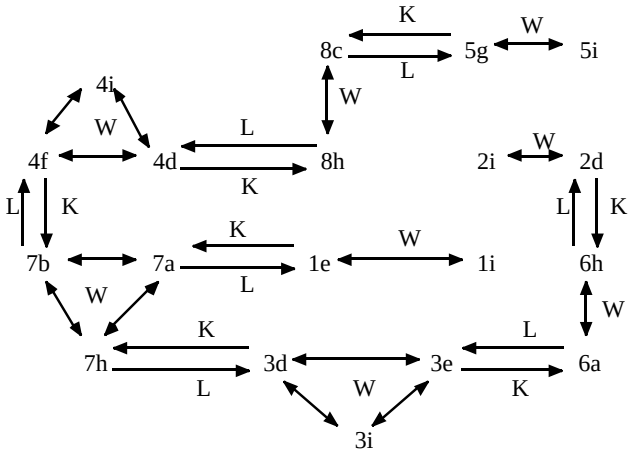


Fig. 2. Unrestricted graph G

In this example we suppress reflexive `w`-arcs, which can have no influence upon reachability. This effectively restricts the `w`-action to cause an immediate change in the agent's perception. Later, when we consider agents wandering incrementally in a positional context, we will permit a `w`-action to maintain the current perception.

A policy f renders the agent more deterministic. It prunes arcs from G to leave an *f-restricted graph* G_f such that all emergent arcs from any situation bear the same action label. Some non-determinism remains, in that a `w`-action from any situation offers (usually) arcs to several others. Assuming that the agent is to have some policy and that it can be assigned initially to any situation, our interest is in defining and identifying for it an optimal policy, given some intended goal. In the example above, the number of policies to choose from is 256, being the product of the cardinalities of all the $A(p)$ sets.

The remainder of this paper describes one particular method for predicting the merits of policies, and then tests this method against a set of case studies. The testing relies upon simulating agents in both positionless and positional modes and measuring their observed efficacies. The last of the case studies shows that abstraction at the formulation stage can enhance the method's scalability.

2 Predicting Policy Values

The value of a policy f should reflect both the agent's ability to achieve the goal and the overall benefit of it doing so, as it proceeds (implicitly) through the graph G_f. One way to predict a value for f is to use the discounted-reward principle [7]. This assigns a value to each situation on the basis of the expected rewards

gained by traversing its emergent arc(s). Various versions of this principle give differing degrees of flexibility. We employ a *Teleo-Agent Policy Evaluator* which applies this simple version: a situation S having immediate successor-set SS in G_f has a value $V(S)$ defined by $V(S) = \Sigma_{s \in SS}(p_s \times (rwd(s) + \gamma \times V(s)))$, where p_s is the probability that from S the agent proceeds next to s, $rwd(s)$ is the reward it earns by doing so and γ is a discount factor such that $0 \leq \gamma \leq 1$; in the case that $SS = \emptyset$ we have $V(S) = 0$. More general versions parameterize their rewards and discount factors by the particular actions entailed in the S-s transitions. In our version we choose two fixed numbers R and r such that $rwd(s) = R$ if s is a goal and $rwd(s) = r$ otherwise, where $R \gg r$. The difference between R and r governs the extent to which the method accords merit to the agent reaching a goal rather than not doing so, whilst γ controls the separation (but, generally, not the ranking) of policy values. The advantage of the method is that, provided $\gamma < 1$, the values of all situations are finite even when G_f contains cycles signifying non-terminating behaviour. Moreover, each value is a linear combination of other values and all may be computed efficiently by linear equation solving. On the other hand, it may not be easy to choose rewards and discount factors that relate intuitively to the physics of the agent, such as the resources expended on perceiving and acting. However, this is not necessarily a major impediment to the task of distinguishing good policies from bad ones.

The predicted value $V_{\text{pre}}(f)$ for a policy f is then the mean of all the situations' values, assuming that the situations are equally probable as initial ones for the agent to be in. The predicted optimal policy is the one having the highest predicted $V_{\text{pre}}(f)$ value. We need not always resort to such methods: for particular combinations of world, agent and goal, good policies may be ascertainable by intuition alone. Presently we will see examples where this is so, and others where it is not. Where the method is used, we (currently) evaluate all policies. This places some limitation on the method's effectiveness, as there may be many policies to examine. Various stances can be adopted on this. Firstly, if having an optimal agent is sufficiently important and if the design process is "once-and-for-all", it may be acceptable to expend heavy effort on policy evaluation. Secondly, intuitions can be applied at the start to eliminate clearly poor policies without needing to evaluate them. Thirdly, the number of policies can be reduced by the use of abstraction, that is, by the use of approximate rather than comprehensive problem formulations; this approach yields less predictive power but can still be an effective device, as we will later show by a concrete example.

If the set of situations is large, then the approach using abstractions separately partitions the set $\mathcal{P}$ of perceptions and the set $\mathcal{O}$ of objective states into classes. Each class so obtained is treated accordingly as a single abstract perception or abstract state. The partitioning is guided by intuition about the agent's lack of need to distinguish between the individual members in order to achieve the goal. Pairwise combinations of abstract states and perceptions yield abstract (or *generic*) situations and these become the nodes of the unrestricted graphs. Let (o_G, p_G) be a generic situation, then we impose the restriction, called the *principle of genericity* that some, and preferably many, of the situations belong-

ing to the cartesian product $o_G \times p_G$ are concrete situations. This means that o_G can be associated with a set of perceptions $P(o_G)$ in the same way as for objective states. Moreover, we can associate with each generic perception p_G a set of actions $A(p_G)$. The principle of genericity then also requires that each action in $A(p_G)$ should belong to each perception in p_G. These restrictions on generic situations are made so that good policies predicted using the abstract graph translate into reliable policies in practice. Case Study 5 uses abstractions and discusses this issue further.

This kind of approach can be illustrated straightaway, by using an abstraction with just two abstract states $[e4]$ and $[ne4]$, where $e4$ denotes "a 4-tower exists" and $ne4$ denotes "no 4-tower exists", and three seeing perceptors, $s0$, $s4$ and sx, the latter denoting "seeing neither the surface nor a 4-tower". This abstraction suits the goal of building a 4-tower from an arbitrary but sufficient number of blocks. Figure 3 shows the restricted graph for the policy whereby the agent, if seeing a tower of height neither 0 nor 4 (sx), can `pick` if not-holding (nh) or `place` if holding (h), but in all other cases wanders. The intended goal is the situation $([e4], [nh, s4])$. In case there are 4 blocks, the generic state $[ne4]$ consists of all states except state 5 in Fig. 1. The generic perception $[nh, sx]$ consists of perceptions d, e, f from Fig. 1. The cartesian product includes 21 pairs, of which 6 correspond to concrete situations.

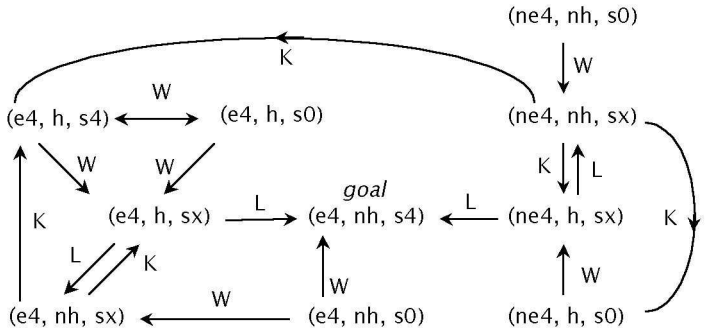


Fig. 3. Using generic situations

If an unrestricted graph is formed using generic states, as opposed to objective states, it is likely that the degree of non-determinism present will be increased. This is because there is more opportunity for variation in the destination states. For instance, in the restricted graph shown in Fig. 3, after the action `pick` from the state $([e4], [nh, sx])$, the agent may be looking either at the surface ($x = 1$), or at a 4-tower ($x = 5$) or at some other tower ($x \neq 1$ and $x \neq 5$). Initially, probabilities on multiple arcs issuing from a situation are assigned uniformly, unless other information is available. However, our simulator is able to estimate

the probabilities of traversing each arc and these empirical values can be used to give more reliable policy values.

3 Simulating Agents

We test the effectiveness of policies by using a *Teleo-Agent Simulator* to simulate agents possessing them. A single *run* assigns the agent to an initial situation S and then drives its subsequent activity in accordance with the chosen policy f. The simulated agent is made to behave deterministically, implicitly traversing some single path in G_f from S. The run terminates when the agent either reaches the goal or exceeds a prescribed bound B on the number of transitions performed. As the path is traversed, the value of $V(S)$ is computed incrementally on the same basis as used in the predictive *Policy Evaluator*. Equal numbers of runs are executed for each initial situation S. The mean observed $V(S)$ over all runs for all S then gives the observed policy value $V_{\text{obs}}(f)$.

For a set $\mathcal{F}$ of n policies we can measure the correlation between their observed and predicted values as follows. A pair $(f, g) \in \mathcal{F} \times \mathcal{F}$ (distinct f, g) for which $V_{\text{pre}}(f) \leq V_{\text{pre}}(g)$ is *concordant* if $V_{\text{obs}}(f) \leq V_{\text{obs}}(g)$, but is otherwise *discordant*. If C is the number of concordant pairs and D the number of discordant pairs, then the *Kendall rank-correlation coefficient* [8] $\tau_{\mathcal{F}}$ for $\mathcal{F}$ is $2 \times (C - D)/(n \times (n - 1))$. We can re-express this measure as a percentage $Q_{\mathcal{F}} = 50 \times (1 + \tau_{\mathcal{F}})$. In the best case $Q_{\mathcal{F}} = 100\%$, when the predicted and observed ranks of all policies agree perfectly. In the worst case $Q_{\mathcal{F}} = 0\%$, when they disagree maximally. The $Q_{\mathcal{F}}$ values cited in the case studies we report here all imply, with $> 99.8\%$ confidence, that the observed and predicted policy values are correlated.

The simulator also reports the observed success rate $SR_{\text{obs}}(f)$ for the policy, measured as the percentage of runs that reach the goal. The success rate can be predicted independently of the simulator by considering the reachability of the goal in G_f. Let $Sits$ be the set of all situations in the problem formulation (and hence in G and in all its policy-restricted subgraphs). Then the choice of f partitions $Sits$ into two disjoint subsets N_f and T_f called the *non-trough* and the *trough*, respectively. N_f contains the goal and all situations from which the goal is reachable under policy f. T_f contains all the other situations. By definition, there cannot exist any arc in G_f directed from T_f to N_f. However, there may exist one or more in the opposite direction, in which case we describe G_f as NT-bridged. The *Policy Evaluator* has the secondary function of determining N_f, T_f and the NT-bridged status for any policy f.

If G_f is not NT-bridged then the agent can reach the goal if and only if its initial situation is in N_f, so that its predicted success rate $SR_{\text{pre}}(f)$ (as a percentage) is exactly $C = 100 \times |N_f|/|Sits|$. If G_f is NT-bridged then we have only the weaker relationship $SR_{\text{pre}}(f) < C$. In either case C provides an upper bound on $SR_{\text{pre}}(f)$, which we may then compare with $SR_{\text{obs}}(f)$. Simulated runs curtailed by the bound B thereby fail to reach a goal that may have been

reachable in principle. So in general $SR_{\text{pre}}(f)$ will overestimate $SR_{\text{obs}}(f)$ by an extent that depends on B.

The simulator supports both *positionless* and *positional* simulation modes. The former associates no positional data to the agent or the towers, and so precisely mirrors the information content of the original problem formulation. This means, for instance, that when the agent picks a block from the 2-tower in the state [1, 1, 2], it is indeterminate as to which tower it will see afterwards in the new state [1, 1, 1]. By contrast, the positional mode assigns discrete grid coordinates to the agent and the towers, and exploits these to effect transitions in a manner somewhat closer to physical reality through knowing precisely where the agent is located and what it sees.

4 Case Studies in Positionless Mode

Positionless simulation and our prediction method explore an agent's behaviour in similar ways and so we expect them to give similar outcomes. It is still useful to test this by case studies. These studies also serve to introduce concepts and notations needed later in the paper when we present cases studies of agents in positional contexts.

Here we compare predicted results with those observed through positionless simulation, taking three different goals for an agent acting in a world of 4 blocks. When computing the predicted policy values we suppress any arcs emergent from the goal, in order to mirror the simulator's behaviour in terminating the agent when it reaches the goal; though instituted for the sake of rigour, this nuance has no discernible impact upon the outcomes in these particular examples. The prediction parameter values used throughout are R=100, r=-1 and γ=0.9; we determined by prior experiments that other choices would not have altered the results or conclusions reported here.

Case Study 1: [*Goal=(5,g): construct and perceive a 4-tower*] This is the case introduced in Example 1. The predicted optimal policies are

$a \rightarrow$ `w`, $b \rightarrow$ `l`, $c \rightarrow$ `l`, $d \rightarrow$ `k`, $e \rightarrow$ `k`, $f \rightarrow$ `w`, $g \rightarrow$ `k`, $h \rightarrow$ `w`, $i \rightarrow$ `w`
$a \rightarrow$ `w`, $b \rightarrow$ `l`, $c \rightarrow$ `l`, $d \rightarrow$ `k`, $e \rightarrow$ `k`, $f \rightarrow$ `w`, $g \rightarrow$ `w`, $h \rightarrow$ `w`, $i \rightarrow$ `w`

for each of which $V_{\text{pre}}(f) = 37.30$ and $SR_{\text{pre}}(f) < 73.68\%$. Their graphs are NT-bridged. Owing to the suppression of arcs from $(5, g)$ the action for perception g is actually immaterial. From 1007 simulated runs per policy (thus, 53 for each initial situation) with bound B=100, the same two policies are identically observed as optimal, having $V_{\text{obs}}(f)$=37.32 and $SR_{\text{obs}}(f) = 70.75\%$.

A broader perspective for this set $\mathcal{F}$ of 256 policies is obtained by charting their observed values against the ranks of their predicted ones. For good predictive quality the profile should decrease monotonically. The chart obtained, in Fig. 4, does so and its Kendall measure $Q_{\mathcal{F}}$ is 99.56%. The optimal policies for the given goal are intuitive, directing the agent to place a block only upon an

existing tower (so, not upon the surface) and to pick a block only from a tower of height < 3.

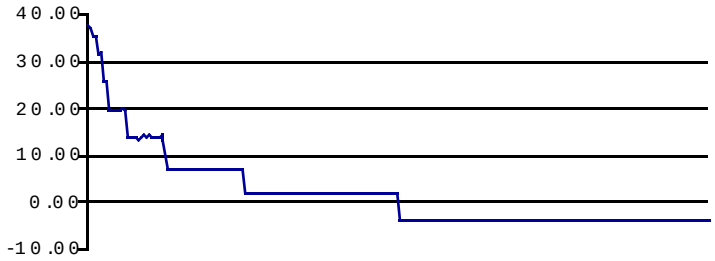


Fig. 4. Observed policy values in Case Study 1

Case Study 2: [*Goal*=$(2, i)$*: construct four 1-towers, perceive surface*] Here the predicted optimal policy is

$$a \rightarrow \texttt{w},\ b \rightarrow \texttt{w},\ c \rightarrow \texttt{w},\ d \rightarrow \texttt{w},\ e \rightarrow \texttt{k},\ f \rightarrow \texttt{k},\ g \rightarrow \texttt{k},\ h \rightarrow \texttt{l},\ i \rightarrow \texttt{w}$$

for which $V_{\text{pre}}(f)$=41.71 and $SR_{\text{pre}}(f) = 100\%$. Its graph is not NT-bridged. From 1007 simulated runs with bound B=100, the same policy is optimal, having $V_{\text{obs}}(f) = 41.76$ and $SR_{\text{obs}}(f) = 100\%$. The ranking chart, shown in Fig. 5, is again almost perfectly monotonic, and $Q_{\mathcal{F}} = 99.88\%$. Again, the optimal policy is intuitive – place a block only upon the surface and pick a block only from a tower of height > 1.

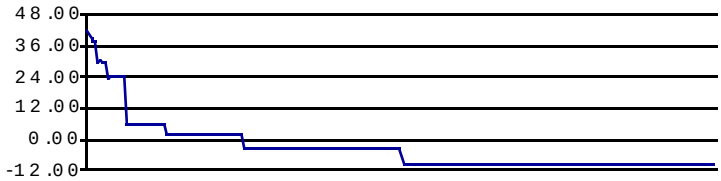


Fig. 5. Observed policy values in Case Study 2

Case Study 3: [*Goal* $= (3, i)$*: construct one 2-tower, two 1-towers, perceive surface*] Here the predicted optimal policy is

$$a \rightarrow \texttt{w},\ b \rightarrow \texttt{w},\ c \rightarrow \texttt{w},\ d \rightarrow \texttt{w},\ e \rightarrow \texttt{w},\ f \rightarrow \texttt{k},\ g \rightarrow \texttt{k},\ h \rightarrow \texttt{l},\ i \rightarrow \texttt{w}$$

for which $V_{\mathrm{pre}}(f)$=31.57 and $SR_{\mathrm{pre}}(f) = 68.42\%$. Its graph is not NT-bridged. From 1007 simulated runs with bound B=100, the same policy is optimal, having $V_{\mathrm{obs}}(f) = 31.37$ and $SR_{\mathrm{obs}}(f) = 68.42\%$. Figure 6 shows the ranking chart, which is a little less monotonic. Here $Q_{\mathcal{F}} = 91.75\%$, not quite as high as before. More than half (144) of the policies have NT-bridged graphs, whereas the former cases have just 22 and 8 respectively. The many minor anomalies in Fig. 6, whose joint effect is to reduce $Q_{\mathcal{F}}$, arise only because the constraints upon sampling and bounding cause the simulations not to cover all those ways in which the agent may variously avoid or traverse a fateful bridge from the non-trough to the trough. The optimal policy in the present case – place a block only upon the surface, and pick a block only from a tower of height > 2 – is now not so obviously optimal. One might have guessed instead at, say, the policy identified in Case Study 2, on the grounds that in the state comprising two 2-towers the agent needs to pick from one of these in order to progress; but this is mistaken, for that state does not occur sufficiently often to justify this change, and this alternative policy turns out to have less than half the value and less than half the success rate of the optimal one.

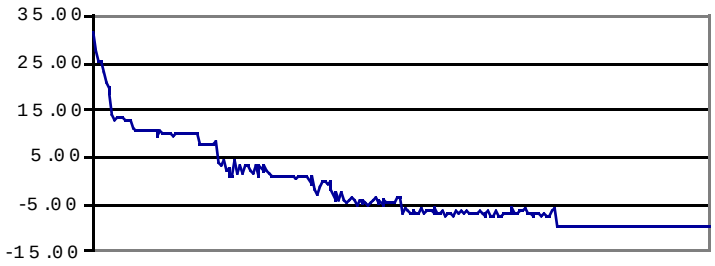


Fig. 6. Observed policy values in Case Study 3

In conclusion, the evaluator and the simulator (in positionless mode) correctly implement the underlying concepts, by mutually validating their outcomes.

5 Case Studies in Positional Mode

Our interest here is in determining how well our *Policy Evaluator* – which knows nothing of positions or of the distinct identities of identical towers – predicts the behaviour of an agent in a position-sensitive world. In its positional mode the simulator manifests an initial situation as an assignment of the towers and the agent to distinct cells in a rectangular grid having prescribable dimensions. Thereafter the simulator drives the agent's behaviour as dictated by the policy f, and updates the agent's grid position as appropriate. Moreover, towers maintain distinct identities by having distinct positions.

A key decision for this mode concerns how the agent shall perform a `w`-action and what it shall see afterwards. We decided this as follows. A `w`-action moves the agent randomly to any vacant cell in its (immediate) neighbourhood (comprising 8 cells, unless the agent is next to a grid boundary). The agent then sees the surface if its new neighbourhood is entirely vacant, but otherwise randomly sees any one tower in that new neighbourhood. This (or any alternative) decision fixes the probability distribution over the possible `w`-transitions between the prior situation and its successors. By contrast, the *Policy Evaluator* treats, by default, those transitions as equi-probable, and thus over-simplifies the agent's behaviour. So we must expect some shortfall in its predictive power when applied to a position-sensitive world, and the question is how serious (or not) that shortfall is.

The grid dimensions are another factor in the shortfall. The smaller they are, the less freedom the agent has to wander. It may become surrounded by towers, particularly near boundaries, causing a run to fail on a path that the *Policy Evaluator* regards as successful. In our studies we used a 6×6 grid, large enough to prevent such possibilities from significantly affecting the comparisons made.

It is clear that in positional mode a `w`-action may leave the agent's situation invariant. In the earlier studies this possibility was denied by our excluding reflexive `w`-arcs from G. Now, however, we add such an arc to every situation in G in order that our predictions shall take account of reflexive transitions in positional simulation. The prediction parameter values used throughout were again $R = 100$, $r = -1$ and $\gamma = 0.9$.

Case Study 4: [*Goal*=$(5, g)$*: construct and perceive a 4-tower*] This repeats Case Study 1 but in positional mode and with bound $B = 200$ (increased from the former 100 to allow for more wandering). The predicted optimal policies are the same as before, but with a slightly reduced predicted value 30.47 (reduced precisely because of the negative rewards of reflexive wanders). However, they are not quite optimal among the observed values. The observed optimal policy is a little different:

$$a \rightarrow \texttt{w},\ b \rightarrow \texttt{l},\ c \rightarrow \texttt{l},\ d \rightarrow \texttt{k},\ e \rightarrow \texttt{w},\ f \rightarrow \texttt{w},\ g \rightarrow \texttt{k},\ h \rightarrow \texttt{w},\ i \rightarrow \texttt{w}$$

On seeing a 2-tower when not holding (perception e), it marginally favours `w` over `k`. Figure 7 shows the ranking chart, for which $Q_{\mathcal{F}} = 66.91\%$. Minor sampling variations among many policies of similar value are the root cause of this reduced value. If we contract $\mathcal{F}$ to just the 20 best policies then $Q_{\mathcal{F}} = 76.32\%$, so the predictive quality is rather better in the region that matters.

The observed values in the chart are much lower than the predicted ones, since the simulated agent is wandering (in the sparse regions of the grid) much more than the prediction considers probable. However, it is the relative rather than the absolute values that are our main interest. The observed and predicted success rates remain in close agreement, since excess wandering does not alter the reachability of the goal, but merely delays its discovery.

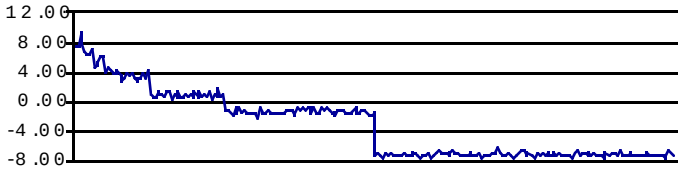


Fig. 7. Observed policy values in Case Study 4

Case Study 5: This is the study we feel is of greatest interest, since it bears directly upon the scalability of the predictive method. The goal is to construct (at least) one 2-tower and one 4-tower and to perceive the surface. If there were (say) 10 blocks, then a comprehensive formulation of this problem dealing with its 72 possible states would not be manageable. Nor might it be realistic to suppose the agent possessed 11 distinct perceptors $s0, \ldots, s10$; even if it did, the graph G would contain a huge number of situations and 2^{20} policies to consider.

Our approach to this problem was to assume that solving it did not require the planning process to distinguish between numerous slightly different situations. Instead, the formulation uses four generic states $[e2,\ e4]$, $[e2,\ ne4]$, $[ne2,\ e4]$ and $[ne2,\ ne4]$, where $e2$ denotes "a 2-tower exists" and $ne2$ denotes "no 2-tower exists" (and analogously for $e4$ and $ne4$). The seeing perceptors are $s0,\ s2,\ s4$ and sx, the latter denoting "seeing an x-tower" (x denotes "neither 2 nor 4"). This formulation gives just 24 situations and 128 policies.

As a further challenge to the predictive method the simulations were done in positional mode, using 10 blocks and making 1008 runs per policy with bound B=200. Figure 8 shows the ranking chart which, though having many fluctuations, is reasonably monotonic overall and has a surprisingly high overall quality measure $Q_{\mathcal{F}} = 86.55\%$.

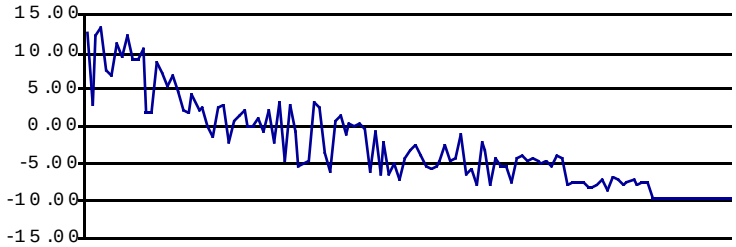


Fig. 8. Observed policy values in Case Study 5

The observed optimal policy is that which places a block only upon the surface or upon an x-tower and never picks a block. It has rank 4 among the predicted values. The observed second-best policy places a block only upon an

x-tower and picks a block only from an x-tower. This has rank 1 among the predicted values.

The fluctuations in this chart arise partly from the prediction method misjudging the probabilities on the arcs, for the reasons noted at the start of this section. In principle we can improve the method by determining more accurate probabilities (by considering how the agent actually behaves) and attaching them as data to the arcs in G_f to be used in the discounted-reward equations. For the current example it is not easy to ascertain accurate probabilities analytically, though it would no doubt be possible, by considering the density distribution of blocks, to obtain reasonable ones with sufficient effort. To show in principle that the method can be improved in this way, we made the simulator count, for each policy f, the number of times each arc was traversed, and used these totals to assign probabilities to the arcs in G_f. The *Policy Evaluator* was then re-run using these more realistic probabilities to provide updated predictions. Comparing these with the simulations' observed values yields the less volatile chart in Fig. 9.

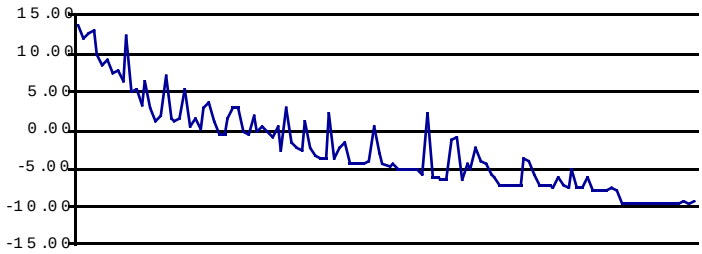


Fig. 9. Result of using better probabilities in Case Study 5

The quality $Q_{\mathcal{F}}$ over the full range has risen to 91.73%, whilst for the best 20 policies it has improved radically from 69.48% to 86.32%. In particular, the observed optimal policy is now ranked 1 among the predicted values. Some volatility remains because (i) each graph G_f offers transitions from generic situations that cannot be realized by all their concrete instances and (ii) the number of runs (42) for each initial generic situation insufficiently tests each of its many concrete instances. Cause (i) exacts an inevitable penalty for any approximate problem formulation, whilst cause (ii) merely calls for more simulation runs per policy to achieve better sampling.

A more challenging study – of building a 10-tower – used just four generic state: the state [10], the state [5, 5], states with one 5-tower and all other states. The agent could distinguish only three cases when seeing an n-tower: $n = 5$, $n < 5$ and $n > 5$. With this degree of abstraction the predictive quality $Q_{\mathcal{F}}$ over the 128 policies was 70.83%. The observed best policy, to pick only from a tower of height < 5 and place only upon a tower of height ≥ 5, was the one predicted as best. This further encouraged the use of abstraction for addressing scalability.

6 Discussion and Conclusions

Our work on teleo-reactive agents was inspired by their pioneer and originator [10, 11]. The main approach to constructing them automatically has been to employ various strategies based upon learning. An example is the work of [1] which uses inductive logic programming to learn from experience useful relations between action and effect. Related work on agent planning by learning is given in [9] and [12]. The formulation of teleo-reactive programs as restrictions of situation graphs was introduced in [2]. The discounted reward principle [7] has recently been used to plan cooperative and communicative teleo-reactive agents [3, 4]. The contributions of this paper include the following:

(i) statistical comparison of predicted discounted-reward values of teleo-reactive policies with the results of simulations;
(ii) evidence that sole reliance upon idealized situation graphs affords good predictive power for simulated agents in position-sensitive cellular worlds;
(iii) elucidation of factors involved in the differences between predictions and observations; and
(iv) evidence that the approach remains fairly robust when, for the sake of scalability, the situation graphs employ generic descriptors of states and perceptions.

Our use of graphs (here termed *op-graphs*) relating situations contrasts with the use of simpler graphs (*p-graphs*) relating perceptions alone. Such p-graphs are traditionally used to represent decision processes and to reason about them [6]. In a p-graph each arc denotes an action transition from one perception to another. Clearly an op-graph distinguishes between situations that share a common perception, whereas a p-graph does not. If in a p-graph a perception p occurs in both goal and non-goal situations, then the discounted reward method overestimates the value of p by attributing the higher reward associated with goal-reaching arcs to arcs not reaching the goal.

There is a further feature that differentiates the two kinds of formulation and is not immediately obvious. This is *accidental non-determinism*, which can occur in the case when a perception occurs in more than one objective state. Suppose action `a` takes $o1p1$ to $o2p2$ and $o3p1$ to $o4p4$ and further suppose these are the only transitions when `a` is the action in states $o1p1$ and $o3p1$. In the p-graph there will be two outgoing arcs labelled by `a` from $p1$ – one to $p2$ and one to $p4$. Thus the p-graph makes `a` into a non-deterministic action when in perception $p1$, whereas the op-graph does not. The consequence of accidental non-determinism when using a p-graph is that the overall policy value may be over or under-estimated. A node in the trough of a given restricted op-graph may, for the corresponding p-graph, lie on a path to the goal perception resulting in overestimation. Or, a short deterministic path to the goal in an op-graph might project onto a non-deterministic set of paths, including long and futile ones, in the p-graph, resulting in an under-estimation of policy value. Furthermore, in the same way that our use of op-graphs extends p-graphs, so also our abstraction of situations extends approaches, as in [5], that abstract perceptions alone.

It is therefore apparent that there are circumstances in which these factors render the p-graph approach less well suited to predicting good policies using discounted reward methods. Accidental non-determinism introduced by using a p-graph can be removed in some cases by extending perceptions (e.g. transition histories [6]) to enable detection of which objective state the agent is in for some particular perception. However, this "solution" increases the number of possible policies requiring to be considered and evaluated. Our use of op-graphs, kept to manageable proportions as necessary through the use of abstraction, also enables us to find policies that maintain an agent's pursuit of the goal even in a context of state changes induced by unpredictable exogenous factors. This is because our planning method judges how best the agent should behave whatever situation it is currently in, regardless of whether that situation results from the agent's previous action or from arbitrary exogenous disturbance.

The key aspects of teleo-reactive agents are their autonomy and their lack of dependence upon on-board computing power and sophisticated software. It is plausible that they will play a key role in some nano-technological applications, and the quest for practical ways to design them is, we believe, a worthwhile pursuit.

References

1. Benson, S. Learning Action Models for Reactive Autonomous Agents, PhD, Dept. of Computer Science, Stanford University, 1996.
2. Broda, K., Hogger, C.J. and Watson, S. Constructing Teleo-Reactive Robot Programs, *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, pp 653-657, 2000.
3. Broda, K. and Hogger, C.J. Designing and Simulating Individual Teleo-Reactive Robots, Technical Report TR 2003/8, Dept. of Computing, Imperial College London, UK, 2003.
4. Broda, K. and Hogger, C.J. Policies for Cloned Teleo-reactive Robots, To be presented to the *2nd German Conference on Multiagent System Technologies*, Erfurt, 2004.
5. Dearden, A. and Boutilier, C. Abstraction and approximate decision-theoretic planning, *Artificial Intelligence* 89: pp 219-283, 1997.
6. Dutech, A. Solving POMDPs using selected past events, *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, pp 281-286, 2000.
7. Kaelbling, L.P., Littman, M.L. and Cassandra, A.R. Planning and Acting in Partially Observable Stochastic Domains, *Artificial Intelligence* 101: pp 99-134, 1998.
8. Kendall, M.G. A New Measure of Rank Correlation, *Biometrika* 30: pp 81-93, 1938.
9. Mitchell, T. Reinforcement Learning, Machine Learning, pp 367-390, McGraw Hill 1997.
10. Nilsson, N.J. Teleo-Reactive Programs for Agent Control, *Artificial Intelligence Research* 1: pp 139-158, 1994.
11. Nilsson, N.J. Teleo-Reactive Programs and the Triple-Tower Architecture, *Electronic Transactions on Artificial Intelligence* 5: pp 99-110, 2001.
12. Ryan, M.R.K. and Pendrith, M.D. An Architecture for Modularity and Re-Use in Reinforcement Learning, *Proceedings of the 15th International Conference on Machine Learning*, Madison, Wisconsin, Morgan Kaufmann, pp 481-487, 1998.

Planning in Concurrent Multiagent Systems with the Assembly Model Checker StEAM

Tilman Mehler and Stefan Edelkamp

Fachbereich Informatik
Baroper Straße 301
Universität Dortmund
{tilman.mehler,stefan.edelkamp}@cs.uni-dortmund.de

Abstract. The exploration of a programs state space has become a popular method lately. In most cases, the intention is to check the program against formal properties - such as the absence of deadlocks. The efforts in this field gave birth to a set of powerful tools - such as the Java model checker JPF(2) and the C++ model checker StEAM. However, the potential of those tools is not limited to the verification of formal properties.
In this paper, we present a method, which uses StEAM as a planner in concurrent multiagent systems. In contrast to classical planning, we avoid the generation of an abstract model. The planner operates on the same compiled code that controls the actual system.
The approach is evaluated in a case study using a C++-implementation of a multiagent manufacturing system.

1 Introduction

Modern AI is often viewed as a research area based on the concept of *intelligent agents* [20]. The task is to describe and build agents that receive percepts from the environment. In fact, each such agent implements percepts to actions. So distributed *multiagent systems* have become an important sub-field of AI, and several classical AI topics are now broadly studied in a concurrent environment. *Planning for multiagent systems* extends classical AI Planning to domains where several agents act together. Application areas include multirobot environments, cooperating Internet agents, logistics, manufacturing etc. Approaches differ for example in their emphasis on either the *distributed plan generation* or the *distributed plan execution* process, and in the ways communication and perception is used.

The largest discrepancy between generating a plan and executing it, is that multiagent planners usually cannot directly work on existing programs that are executed. In this paper, we show how a c++ program model checker can be utilized to serve as a planning *and* execution unit to improve the performance of concurrent multiagent system implementations.

With this respect, this work provides another example of showing on the *cooperation of model checking and planning technology*. The rough view is

that finding an error in software programs is closely related to establishing goals in planning. The counterexample provided by the model checker corresponds to a sequence of actions that solves the planning problem. Concurrency is established through running threads and communication is implemented via message passing or global variables.

Directing the exploration towards the planning goal or specification error turns out to be one of the chances to tackle the *state explosion problem*, that arises due to the combinatorial growth of system states. In our model checker we work with different object-code distance metrics to measure the efforts needed to encounter the error. Thereby, we can also address the *evaluation problem* that multiagent systems or multirobot teams have, since by having access to object code we can attribute execution time to a set of source code instructions.

The goal in our research is to develop a planning system which interacts with the software units to improve the quality of the results. Since the performance of the system on test increases over time, our approach considers *incremental learning*. As a special feature, the planner should not build an abstract model of the concurrent system. Instead, planning should be feasible directly on the implementation of the software units. This introduces *planning on the sorce-code and assembly-level*.

The paper is structured as follows. We first introduce software and program model checking. Then we discuss concurrent multiagent systems and how they can be implemented in a C++ program to be checked by our model checker StEAM. Afterwards, we explain the extensions to the original model checker, so it can also be used as a planner. We present a general state exploring algorithm that interleaves simulation and planning. Next, we consider the case study on a multiagent manufacturing system, MAMP for short. For efficient exploration in these problems we subsequently study available search enhancements. Experiments show, how the interaction of StEAM as a planning system can improve the performance of the manufacturing process. Afterwards we relate to further work on planning, model checking, and concurrent multiagent systems. Finally, we draw conclusions and discuss further efforts in this field.

2 Software and Program Model Checking

Software model checking has proven a useful technique for the verification of programs written in modern programming languages - such as C/C++ and Java. Some early approaches [6,12,15] rely on the extraction of abstract models from source code, which can be verified by well-established model checkers such as SPIN [14] and SMV [18]. Although such an abstraction generally leads to a more compact state description, this method has several drawbacks. One being, that it requires a custom-build formal semantics, which translates source code to the input language of the model checker. Due to the complexity of the formal semantics of actual programming languages, the methods based on the extraction of formal models are often very

limited in the subset of programs that can be translated. Also, it is hard to prove, if the model correctly reflects all aspects of the actual program.

Newer model checkers, like the second generation of the Java Pathfinder (JPF) [21] and the C++ Model Checker StEAM [17] use a different approach. Instead of extracting a model from the source code, the investigated software is compiled to machine code. Then a virtual machine is used to check program properties directly on the compiled code. This method of *program model checking* has several advantages: First of all, the developers of the model checker do not have to re-invent the wheel by designing a translation from the program source to the input language of the model checker. For common programming languages, such as C++ and Java, plenty of infrastructure already exists, such as compilers and machine code interpreters. This infrastructure only needs to be extended with model checking abilities, such as a state space description and search methods.

In the case of the StEAM model checker, a C++ virtual machine called ICVM was extended to a model checker for concurrent C++ programs with reasonable effort. Since ICVM can compile and execute arbitrary C++ code, the full formal semantics of the programming language is covered. Furthermore, since ICVM is capable to run complex programs[1], we have strong empirical evidence, that the formal semantics of the programming language is correctly reflected in the compiled code.

Special-purpose statements are part of the control mechanism which tailors the model checking algorithm to ICVM. Essentially, these are macros, that are placed in the code of the program to be model-checked. The compiled code of such a macro is recognized by the model checker and gets parsed instead of executed. Two examples of such a statements are BEGINATOMIC and ENDATOMIC, marking a block that includes no choice point for the set of executed threads.

3 Concurrent Multiagent Systems in StEAM

We regard a *concurrent multiagent system* as a set of homo- or heterogeneous autonomous software units, operating in parallel. Some components of the system are shared among all units - such as tasks or resources. The units cooperate to reach a certain goal and the result of this cooperation depends on what decisions are made by each agent at different times.

Concurrent multiagent systems can be implemented in the program model checker StEAM in a straightforward fashion. Software units are represented as threads. Each unit is implemented by an instance of a corresponding C++-class. Each such class is derived from StEAM's thread-class *IVMThread*. After creating a class instance, a call to the *start*-method adds it to the list of concurrent processes running in the system. The shared components of the system are represented by global variables.

[1] For example it has been tested for time critical computer games like *Doom*.

Model checking in StEAM is done by performing exploration on the set of system states, which allows to store and retrieve memorized configurations in the execution of the program. In the simulation mode e.g. controlled by the user, by random choices, or by an existing trace, the model checker executes the program along one sequence of source code instructions.

Here, the model checker StEAM serves two purposes. First as a platform to *simulate* the system, second as a *planner* which interacts with the running system to get better results. We assume an *online scenario*, where planning has to be done in parallel to the execution of the software units. In particular, it is not possible to find a complete solution in advance.

We assume that the planner performs a *search on system states*, which are of the same type as that of the actual concurrent system. As a result, the planner finds a desired target state t, which maximizes the expected solution quality. State t will not necessarily fulfill the ultimate goal, because an exhaustive exploration is in general not possible due to time and space restrictions. As the next step, the planner must carry over the search results to the *environment*. However - in contrast to the plan generation - the planner does not have full control over the actual system, due to nondeterministic factors in the environment, such as the execution order of the agents. Instead, the planner must communicate with the software units to influence their behaviour, so that the actual system reaches state t or a state that has the same or better properties as t w.r.t. the solution quality.

3.1 Interleaving Planning and Simulation

We want to integrate the required planning ability into the model checker with minimal changes to the original code. First, a model checker operates on sequential process executions, rather than on parallel plans. To address this issue, the concept of *parallel steps* is added to StEAM. A parallel step means that all active processes are iterated one atomic step in random order. With parallel steps, we can more faithfully simulate the parallel execution of the software units.

Communication is realized by a concept we call *suggestion*. A suggestion is essentially a component of the system state (usually a shared variable). Each process is allowed to pass suggestions to the planner (model checker) using a special-purpose statement SUGGEST. The SUGGEST-statement takes a memory-pointer as its parameter, which must be the physical address of a 32-bit component of the system state. The model checker collects all suggestions in set SUG. When the model checker has finished his plan generation, it overwrites the values of all suggestions in the current state of the actual environment with the corresponding values in t. The software units recognize these changes and adapt their behaviour accordingly. Formally, the role of suggestions can be described as follows: Let V be the set of variables that form a state in our system. A system state is defined as a function $z : V \rightarrow \mathbb{R}$, which maps variables to their domains. Let s be the root state of a planning phase and t the designated target state. Then, the

subsequent simulation phase starts at state:

$$s' = s \setminus \left(\bigcup_{v \in Sug} \{v \mapsto s(v)\} \right) \cup \bigcup_{v \in Sug} \{v \mapsto t(v)\}.$$

A concrete example will be given in the case study in Section 4.

3.2 Combined Algorithm

Our proposed system iterates two different phases: *planning* and *simulation*. In the *planning phase* all units are frozen in their current action. Starting from a given system state, the model checker performs a breadth-first exploration of the state space until a time limit is exceeded or no more states can be expanded. For each generated state u an *evaluation function* value $h(u)$ is calculated, which measures the solution quality in u. When a time limit θ is reached, the state with the best h-value is chosen as the target state t and the *simulation phase* starts. At the beginning of the simulation phase, for each element in the set of suggestions, the model checker overwrites the value of the corresponding component in s with the value in t. Then parallel steps are executed in s until the criterion for a new planning phase is met.

Algorithm *Interleave* in Figure 1 illustrates the two phases of the system, and is based on top of a general state expanding exploration algorithm. Besides the initial state the exploration starts form, it takes the preference state evaluation function as an additional parameter. A time threshold stops planning in case the model checker does not encounter optimal depth. For the ease of presentation and the online scenario of interleaved execution and planning, we chose an endless loop as a wrapper and have not included a termination criterion in case the goal is encountered.

The function *evaluateCriterion* determines, whether a new planning phase should be initiated or not. The function is not defined here, because this criterion varies depending on the kind of system we investigate. The function *s.iterate(a)* executes one atomic step of an agent (thread, process) a in state s and returns the resulting state.

Figure 2 shows how the generated system states switch from a *tree structure* in the planning phase to a *linear sequence* in the simulation phase. We see that the simulation (path on right side of the figure) is accompaigned by intermediate planning phases (tree structure, top left and bottom right of the figure). The search tree contains the (intermediate) target state t - i.e. the generated state with the highest h-value. As we will see, state t is potentially - but not necessarily traversed in the simulation phase.

Ideally, the two phases run in parallel, that is the running multiagent system is not interrupted. Since we use the same executable in our model checker to simulate and plan, we store system states that are reached in the simulations and *suspend its execution*. The next planning phase always

Procedure *Interleave*
Input: The initial state s of the system, time limit θ, evaluation function h
1. **loop**
2. $besth \leftarrow h(s)$; $t \leftarrow s$; *Open* $\leftarrow \{s\}$
3. **while** $(time < \theta \wedge$ *Open* $\neq \emptyset)$ /* start of planning phase */
4. $\Gamma(u) \leftarrow expand(Open.getNext())$ /* expand next state in horizon list */
5. **for each** $v \in \Gamma(u)$
6. **if** $(h(v) > besth)$ $t \leftarrow v$; $besth \leftarrow h(v)$
7. **for each** $\sigma \in SUG$
8. $s.\sigma \leftarrow t.\sigma$ /* write suggestions */
9. $c \leftarrow 0$
10. **while** $(\neg c)$ /* begin of simulation phase */
11. **for each** agent a: $s \leftarrow s.iterate(a)$ /* do a parallel step */
12. $c \leftarrow evaluateCriterion()$

Fig. 1. The Implementation of the Interleaved Planning and Simulation System.

starts with the last state of the simulation, while the simulation continues with the root node of the planning process. The information that is learned by the planning algorithm to improve the execution is stored in main memory so it can be accessed by the simulation.

4 Case Study: Multiagent Manufacturing Problem

In the following we formalize a special kind of a concurrent multiagent system, which is used to evaluate our approach. A *multiagent manufacturing problem*, *MAMP* for short, is a six-tuple (A, J, R, req, cap, dur), where $A = \{a_1, \ldots, a_n\}$ denotes a set of agents, $J = \{j_1, \ldots, j_m, \diamond\}$ is a set of jobs including the empty job $\diamond$, and $R = \{r_1, \ldots, r_l\}$ is a set of resources. The mappings req, cap, and dur are defined as follows:

- $req : J \rightarrow 2^R$ defines the resource requirements of a job,
- $cap : A \rightarrow 2^R$ denotes the capabilities of an agent, and
- $dur : J \rightarrow \mathbb{N}$ is the duration of a job in terms of a discrete time measure.

A *solution* to a MAMP $m = (A, J, R, req, cap, dur)$, is a function $sol : A \rightarrow 2^{\mathbb{N} \times J}$. For $(t, \iota) \in \mathbb{N} \times J$, let $job((t, \iota)) = \iota$ and $time((t, \iota)) = t$. Furthermore, let $alljobs_{sol} : A \rightarrow 2^J$ be defined as $alljobs_{sol}(a) = \bigcup_{s \in sol(a)} job(s)$. We require sol to have the following properties.

i. For each $sol(a) = \{(t_0, \iota_0), \ldots, (t_q, \iota_q)\}$ we have that $i \neq j$ implies either $(\iota_i = \iota_j = \diamond)$ or $\iota_i \neq \iota_j$ and for each $i \geq 0$ we have $t_{i+1} \geq t_i + dur(\iota_i)$.
ii. For each $s \in sol(a)$: $req(job(s)) \subseteq cap(a)$.
iii. For all $a \neq a'$ and all $s \in sol(a), s' \in sol(a')$ with $req(job(s)) \cap req(job(s')) \neq \emptyset$ we have either $time(s) + dur(job(s)) \leq time(s')$ or $time(s) \geq time(s') + dur(job(s'))$

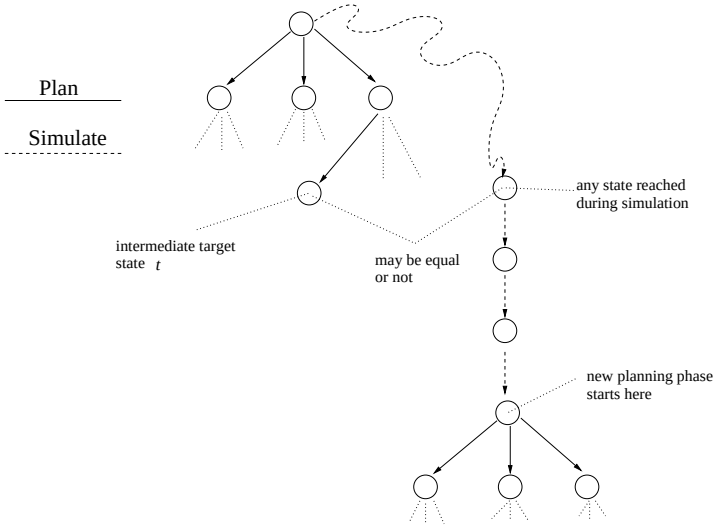


Fig. 2. State generation in the two different exploration phases.

iv. For all $a \neq a'$ we have $(alljobs_{sol}(a)\backslash\{\diamond\}) \cap alljobs_{sol}(a') = \emptyset$.
v. Last but not least, we have $\bigcup_{a \in A} alljobs_{sol}(a) = J\backslash\{\diamond\}$.

Property *i*. demands, that each agent can do at most one job at a time. Property *ii*. says, that an agent can only do those jobs, it is qualified for. Property *iii*. means, that each resource can only be used by one agent at a time and properties *iv*. and *v*. demand, that each job is done exactly once.

An *optimal solution* sol minimizes the *makespan* τ until all jobs are done:

$$\tau(sol) = \max_{a \in A, s \in sol(a)} (time(s) + dur(job(s))).$$

MAMPs are extensions to *job-shop scheduling* problems as described e.g. in [1]. As the core difference, MAMPs also have a limited set of agents with individual capabilities. While in a job-shop problem one is only concerned with the distribution of resources to jobs, MAMPs also require that for each job we have an agent available that is capable to use the required resources.

4.1 Example Instance

Apparently, the solution quality of a MAMP relates to the degree of parallelism in the manufacturing process. A good solution takes care that each agent works around the clock - if possible. Consider a small MAMP $m = \{A, J, R, req, cap, dur\}$ with $A = \{a_1, a_2\}$, $J = \{j_1, j_2, j_3\}$, $R = \{r_1, r_2, r_3, r_4, r_5\}$, $req(j_1) = \{r_1, r_2\}$, $req(j_2) = \{r_3, r_4\}$, and $req(j_3) = \{r_3, r_5\}$. Furthermore $cap(a_1) = R$, $cap(a_2) = \{r_3, r_4, r_5\}$, and $dur(j_1) = 2$, $dur(j_2) = dur(j_3) = 1$.

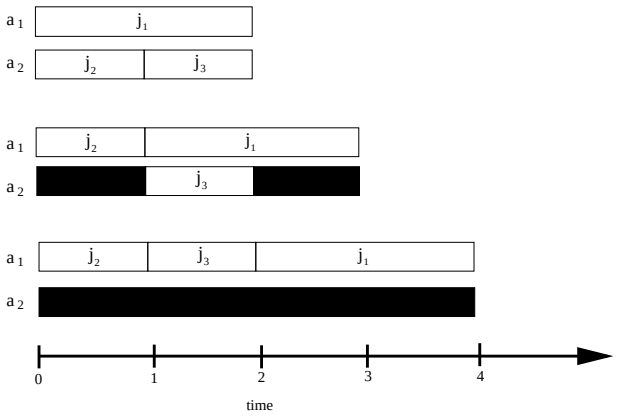


Fig. 3. Three different solutions for the same MAMP m.

Figure 3 illustrates a best, an average and a worst case solution for m. Here, we have a timeline extending from left to right. For each solution, the labeled white boxes indicate the job, which is performed by the respective agent at a given time. Black boxes indicate that the agent is idle. In the optimal solution, a_1 starts doing j_1 at $time = 0$. Parallel to that, a_2 performs j_2 and afterwards j_3. The total time required by the optimal solution is 2. Note, that the solution does not contain any black boxes. In the average solution, which is in the middle, a_1 executes j_2 at $time = 0$ and j_1 afterwards. This implies that a_2 has to be idle up to $time = 1$, because it is incapable to use r_1 needed for j_1 and r_3 is used by j_1. Then, a_2 can at least commence doing j_3 at $time = 1$ when r_3 is available again. The time needed by the average solution is 3. Finally, in the worst solution, which is the bottommost in Figure 3, a_1 first performs j_2, then j_3 and j_1, while a_2 is compelled to be idle at all time. The worst solution needs 4 time units to get all jobs done.

4.2 Implementation

For the chosen concurrent multiagent manufacturing system, our goal is to develop a planning procedure for an instance that interacts with the multiagent system to *maximize the level of parallelism* and thus to minimize the idle times of agents, which is expected to improve the quality of the resulting MAMP-solution. Additionally, as an immediate result of our proposal, we avoid the generation of an abstract model of the system. Instead, we plan directly on the actual implementation.

Information about the state of the jobs and the available resources are stored in global variables. When running, each agent a automatically searches for available jobs he is qualified for and executes them. If a finds a free job j, such that $caps(a) \subseteq req(j)$ and all resources in $req(j)$ are available, the

agent requests the resources and starts executing the job. After finishing the job - i.e. when $dur(j)$ time units have passed, a will release the resources and search for the next job. If a cannot find a job, he waits one time unit and searches again. In the case of our multiagent system, each agent passes as a suggestion the local variable *assigned*, which stores the next job to be executed by the agent. Before an agents autonomously seeks a job, it checks if the default-value of its *assigned*-variable has changed. This implies, that - as a result of the previous planning phase - the model checker has assigned a job for this agent. If this is the case, the agent skips the procedure, which looks for a feasible job and starts allocating the required resources.

5 Search Enhancements

When using breadth-first search, for n active processes (agents), the number of expanded states in the planning phase is bounded by $O(n^d)$ where d is the depth of the search tree. Without search enhancements, it is hard to reach a search depth which is deep enough to gain information for the simulation phase. As a result, the solutions obtained through the combination of planning and simulation would not be any better, than that of a purely autonomous system without planning. Several pruning and refinement techniques help our planning system to reach a larger search depth.

5.1 State Space Pruning

First of all, the model checker StEAM uses a *hash table* to store the already visited states. When a state is expanded, only those successor states are added to the *Open* list, which are not already in the hash table.

Second for a multiagent management system it does not make sense to generate those states of the search tree that correspond to an execution step of a job. Such an execution only changes local variables and thus does not influence the behaviour of the other agents. To realize this pruning, threads can announce themselves *busy* or *idle* to the model checker using special-purpose statements. In the case of the MAMP, an agent declares himself busy, when he starts working on a job and idle when the job is finished. When expanding a state, only those successors are added to the *Open* list that correspond to the execution of an idle thread.

Third, an idle agent looks for a job in one atomic step. Therefore, the execution of an agent that does not result in a job assignment yields no new information compared to the predecessor state. For the given multiagent system, this implies that we only need to store those successor states that result in the *change of a suggestion* (i.e. in a job assignment).

5.2 Evaluation Functions

As shown in Figure 2 the planner cannot always enforce the target state t to be reached, due to *clashes* in the simulation phases. A clash occurs, if

an agent a is iterated before another agent b and autonomously picks a job assigned to b. Our approach considers the value of suggestions to reflect the individual choices of each agent - in this case the job it will do next. The value of this choice is determined by the agent or the planner, before the job itself is taken. Since an agent has no information about the internal values of other agents, clashes are inevitable. This raises the question, what the planner can rely on as a result of its interaction with the system. The answer to this lies in the choice of the evaluation function that is used to determine the intermediate target state during the planning phase. In our case, we use the number of allocated jobs as the evaluation function h, where *allocated* means that an agent is working on that job. It holds, that $h(s) \leq h(s')$ for any successor state of s, since the counter of assigned jobs never decreases - even if a job is finished. In other words h describes the number of currently allocated, plus the number of finished jobs.

Theorem 1. *Let t be the intermediate target state in the search tree. Furthermore, let s_i describe the i-th state of the simulation phase and p_i denote the i-th state on the path from the initial state to t in the search tree, where $i \in \{0, \ldots, m\}$ and $p_m = t$. Then for each $0 \leq i \leq m$ we have that $h(s_i) \geq h(p_i)$.*

Proof. In fact, we have $h(s_0) = h(p_0)$ and $h(s_1) \geq h(t)$. The equality $h(s_0) = h(p_0)$ is trivial, since the two states are equal - except for the values of the suggestions. This implies that in particular, the variable which counts the number of assigned jobs has the same value in s_0 and p_0. Furthermore all suggestions in s_0 are initialized with the values from t. If we now consider the first simulation step, there are two possibilities:

In the first case, we have no clashes during the parallel step performed between s_0 and s_1. This implies that each agent picks the job assigned by the model checker (if any). So we have at least $h(s_1) = h(t)$. Additionally, agents that have no job in t may pick an unassigned job, so that $h(s_1) > h(t)$.

In the second case we assume to have clashes. For each such clash, we have one agent, that cannot pick its assigned job which decreases $h(s_1)$ by one. However, we also have another agent, that picks a job, although it has no job in t, which increases $h(s_1)$ by one. So, a clash does not influence the value of $h(s_1)$ and thus we have $h(s_1) \geq h(t)$.

Altogether, we have $h(s_0) = h(p_0)$, $h(s_1) \geq h(t)$ and since the number of taken jobs never decreases, it holds that for all $1 \leq i \leq m$, we have $h(s_i) \geq h(s_{i-1})$ and $h(p_i) \geq h(p_{i-1})$ and in particular $h(t) \geq h(p_{i-1})$ which implies $h(s_i) \geq h(p_i)$ for all $0 \leq i \leq m$. □

6 Experiments

In the experiments we used a Linux system with a 1.8 GHz CPU and 1 GB of main memory. We applied our system to randomly generated MAMPs with 5, 10, or 15 agents and 10, 20, 50, or 100 jobs. The number of resources was set to 100. Each job requires between 10 and 100 steps to be done and up to

10% of all resources. Our goal is to prove, that the combination of planning and simulation leads to better solutions than pure simulation. To do this, for each MAMP m, we first solve m ten times with pure simulation. Then we solve m ten times with the combination of planning and simulation. In both cases, we measure the number of parallel steps of the solution and the total time in seconds spent for planning. A planning phase may be at most 30 seconds long. If during planning a memory limit of 500 MB is exceeded or the open set becomes empty, the phase ends even if the time limit was not reached. A simulation phase executes at least 20 and at most 100 parallel steps. Additionally a simulation phase ends, if at least as many agents are idle, as were at the beginning.

We expect that in the average case the combination of planning and simulation gives better results (in terms of the number parallel steps), that justify the additional time spent on planning. The evaluation function that counts the number of taken jobs is expected to lead to a maximum of parallelism, since more taken jobs imply less idle threads. Table 1 shows the results. Each row represents ten runs on the same MAMP instance. The column *type* indicates the type of run, i.e. either pure simulation (*sim*) or combination of planning and simulation (*plan*). The columns *min*, *max* and μ show the minimal, maximal and average number of parallel steps needed for a solution of the respective MAMP. Finally, *pt* indicates the average time in seconds used for planning.

In almost all cases the average solution quality increases if planning is used. The only exception is the instance with 10 agents and 20 jobs, where the heuristic estimate seems to fail. Note that clashes may lead to worse solutions, since they cause an agent to waste one step realizing that the assigned job is already taken. For all other cases however, we have improvements between 6 and 59 parallel steps in the average. The total planning times range between 26 and 30 seconds, which seems odd, since each planning phase can be up to 30 seconds long. However, this can be explained by the fact, that only in the first planning phase all agents are idle. If in the subsequent planning phases only a small fraction of all agents is idle, the phases may get very short because the model checker can quickly decide, whether new jobs can be assigned or not. If for example we assume that each parallel step corresponds to one minute in reality, then the time spent for planning is neglectible compared to the time gained through the improved solution quality.

7 Related Work

Here, we will refer to other studies that either relate to the use of model checking techniques for planning, planning in multiagent systems or both.

Software model checking is a powerful method, whose capabilities are not limited to the detection of errors in a program but to *general problem solving*. For instance, [10] successfully uses JPF to solve instances of the

type	agents	jobs	min	max	μ	pt
sim	5	10	262	315	286	0
plan	5	10	241	315	255	29
sim	5	20	517	616	546	0
plan	5	20	483	527	510	27
sim	5	50	1081	1233	1149	0
plan	5	50	1084	1187	1143	29
sim	5	100	2426	2573	2478	0
plan	5	100	2310	2501	2426	26
sim	10	10	259	266	261	0
plan	10	10	136	286	246	28
sim	10	20	389	465	432	0
plan	10	20	412	467	460	30
sim	10	50	754	902	839	0
plan	10	50	771	856	814	30
sim	10	100	1732	1871	1809	0
plan	10	100	1761	1858	1783	30
sim	15	10	260	263	261	0
plan	15	10	187	236	216	30
sim	15	20	385	463	417	0
plan	15	20	382	410	387	30
sim	15	50	675	896	767	0
plan	15	50	711	747	725	30
sim	15	100	1528	1728	1607	0
plan	15	100	1497	1600	1548	30

Table 1. Experimental Results in Multiagent Manufacturing Problems.

sliding-tile puzzle. The adaption is simple: action selection is incorporated as non-deterministic choice points into the system.

Symbolic model checkers have been used for *advanced AI planning*, e.g. the Model-Based Planner (MBP) by Cimatti et al.[2] has been applied for solving *non-deterministic* and *conformant planning* problems, including *partial observable state variables* and *temporally extended goals*.

An architecture based on MBP to *interleave plan generation and plan execution* is presented in [4], where a planner generates conditional plans that branch over observations, and a controller executes actions in the plan and monitors the current state of the domain.

The power of *decision diagrams* to represent sets of planning states more efficiently has also been recognized in *probabilistic planners*, as the SPUDD system [13] and its *real-time heuristic programming* variant based on the LAO* algorithm [11] show. These planners solve *factored Markov decision*

[2] http://sra.itc.it/tools/mbp

process problems and encode probabilities and reward functions with *algebraic decision diagrams*.

TL-Plan [2] and the TAL planing system [16] apply *control pruning rules*, specified in first order temporal logic. The hand-coded rules are associated together with the planning problem description to accelerate plan finding. When expanding planning states the control rules for the successors are derived by *progression*.

Planning technology has been integrated in existing model checkers, mainly by providing the option to accelerate error detection by heuristic search [9]. These efforts are referred to as *directed model checking*. First model checking problems have been automatically converted to serve as *benchmarks* for international planning competitions[3].

The work described in [1] reduces job-shop problems to finding the shortest path in a *stopwatch automaton* and provides efficient algorithms for this task. It shows that model checking is capable to solve hard combinatorial scheduling problems.

The paper [8] describes a translation from Level 3 PDDL2.1 to *timed automata*. This makes it possible to solve planning problems with the *real-time model checker* UPPAAL. The paper also describes a case study about an implementation of the translation procedure which is applied to the PDDL description of a classical planning problem.

Some work on *multiagent systems* shares similarities with our proposal. In [7] a framework called ARPF is proposed. It allows agents to exchange resources in an environment which fully integrates *planning and cooperation*. This implies, that there must also be an *inter-agent communication*.

The work [3] proposes model checking for *multiagent systems*, with a framework that is applied to the analysis of a relevant class of multiagent protocols, namely *security protocols*. In a case study the work consider a belief-based exploration of the *Andrew authentification protocol* with the model checker nuSMV that is defined by a set of propositions and evolutions of message and freshness variables.

Multiagent planning is an AI research of growing interest. A forward search algorithm [5] based on the single-agent planner *FF* that solves multiagent problems synthesizes partially ordered temporal plans and is described in an own formal framework. It also presents a general *distributed algorithm* for solving these problems with several coordinating planners.

The paper [19] is closely related to the work at hand. It describes *ExPlanTech*, as an enhanced implementation of the *ProPlanT* multiagent system used in an actual industrial environment. *ExPlanTech* introduces the concept of *meta-agents*, which do not directly participate in the production process, but observe how agents interact and how they carry out distributed decision making. The meta-agent in *ExPlanTech* serves a similar purpose as the model checker in our system, as it is able to induce efficiency consider-

[3] http://ipc.icaps-conference.org

ations from observations of the community workflow. Still, the meta-agent uses its own abstract model of the system.

The approach we consider in this paper differs from the all the above in that it uses *program model checking* and that it is *applied to multiagent systems* in order to *interleave planning and execution* and *learn* from an existing executable of a given implementation.

8 Conclusions

In this paper we described an approach to use a software model checker for planning in concurrent multiagent systems. We did a case study on a multiagent manufacturing system that implements our formal definition. The experiments indicate that our approach is capable to improve the effectiveness of the multiagent system by maximizing the parallelism of the manufacturing process. It turns out that one of the major challenges lies in finding an appropriate evaluation function for the explored states. A suitable definition depends on the type and implementation of the actual concurrent system.

As the main contribution of the paper we see the proposal of a planning methodology, which does not require the construction of an abstract model of the environment. Instead, planning is performed directly on the compiled code, which is the same as used by the simulation. In the future, we would like to try the approach on actual multiagent systems - be it pure software environments or physical systems like a group of robots. Certainly, many multiagent systems are more complex than the MAMP architecture presented in this paper as they e.g. also involve inter-agent communication. We kept our framework simple for deriving a multiagent prototype to test our planning approach. We are determined to make the approach applicable to real multiagent environments in the same way as we - and others - want make assembly model checking applicable to real-world software products. In the long term, both areas may highly benefit from the assembly model checking approach, because it eliminates the task of building an abstract model of the actual system or program. Not only does this save a considerable amount of resources, usually spent on constructing the models, but performing model checking and planning on the actual implementation also avoids possible inconsistencies between the model and the real system.

Acknowledgements The first author is supported by DFG in the project ED 74/2: *Directed Model Checking*, and the second author is supported by DFG in the project ED 74/3: *Heuristic Search*.

References

1. Y. Abdeddaim and O. Maler. Preemptive job-shop scheduling using stopwatchautomata. In J.-P. Katoen and P. Stevens, editors, *AIPS Workshop on Planning via Model Checking*, pages 7–13, 2002.
2. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116:123–191, 2000.

3. M. Benerecetti and A. Cimatti. Symbolic model checking for multi-agent systems. In *LP-Workshop on Computational Logic in Multi-Agent Systems (CLIMA)*, pages 312–323, 2001.
4. P. Bertoli, A. Cimatti, and P. Traverso. Interleaving execution and planning via symbolic model checking. In *ICAPS Workshop on Planning under Uncertainty and Incomplete Information*, 2003.
5. M. Brenner. Multiagent planning with partially ordered temporal plans. In *Internatinal Joint Conference on Artificial Intelligence (IJCAI)*, pages 1513–1514, 2003.
6. J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubachand, and H. Zheng. Extracting finite-state models from Java source code. In *International Conference on Software Engineering (ICSE)*, pages 439–448, 2000.
7. M. M. de Weerdt, J. Tonino, and C. Witteveen. Cooperative heuristic multi-agent planning. In *Belgium-Netherlands Artificial Intelligence Conference (BNAIC)*, pages 275–282, 2001.
8. H. Dierks, G. Behrmann, and K. G. Lahrsen. Solving planning problems using real-time model checking. In *AIPS Workshop on Planning via Model Checking*, pages 30–39, 2002.
9. S. Edelkamp, S. Leue, and A. Lluch-Lafuente. Directed explicit-state model checking in the validation of communication protocols. *International Journal on Software Tools for Technology (STTT)*, 2004. To appear.
10. S. Edelkamp and T. Mehler. Byte code distance heuristics and trail direction for model checking Java programs. In *IJCAI-Workshop on Model Checking and Artificial Intelligence (MoChArt)*, pages 69–76, 2003.
11. E. A. Hansen and S. Zilberstein. LAO * : A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.
12. K. Havelund. Java PathFinder user guide. Technical report, NASA Ames Research Center, 1999.
13. J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Conference on Uncertainty in Articial Intelligence (UAI)*, pages 279–288, 1999.
14. G. J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.
15. G. J. Holzmann. Logic verification of ANSI-C code with SPIN. In *Model Checking Software (SPIN)*, pages 131–147, 2000.
16. J. Kvarnström, P. Doherty, and P. Haslum. Extending TALplanner with concurrency and ressources. In *European Conference on Artificial Intelligence (ECAI)*, pages 501–505, 2000.
17. P. Leven, T. Mehler, and S. Edelkamp. Directed error detection in c++ with the assembly-level model checker StEAM. In *Model Checking of Software (SPIN)*, 2004.
18. K. L. McMillan. The SMV system. Technical report, Carnegie-Mellon University, 1992.
19. M. Pěchouček, A. Říha, J. Vokřínek, V. Mařík, and V. Pražma. Explantech: applying multi-agent systems in production planning. *International Journal of Production Research*, 40(15):3681–3692, 2002.
20. S. Russel and P. Norvig. *Artificial Intelligence, a Modern Approach*. Prentice Hall, 1995.
21. W. Visser, K. Havelund, G. Brat, and S. Park. Java PathFinder - second generation of a Java model checker. In *Post-CAV Workshop on Advances in Verification, (WAVe)*, 2000.

A Fuzzy Data Envelopment Analysis Model Based on Dual Program

Hsuan-Shih Lee

Department of Shipping and Transportation Management
National Taiwan Ocean University
Keelung 202, Taiwan
Republic of China
hslee@axp1.stm.ntou.edu.tw

Abstract. DEA (data envelopment analysis) is a non-parametric technique for measuring and evaluating the relative efficiencies of a set of decision-making units (DMUs) in terms of a set of common inputs and outputs. Traditionally, the data of inputs and outputs are assumed to be measured with precision, i.e., the coefficients of DEA models are crisp value. However, this may not be always true. There are many circumstances where precise inputs and outputs can not be obtained. Under such situations, data of inputs and outputs can be represented by fuzzy numbers. Based on the dual program of DEA models, we propose fuzzy DEA models for CCR and BCC models. Our fuzzy DEA models provide crisp efficiency with fuzzy input and output data.
Keywords: Data Envelopment Analysis (DEA); Fuzzy DEA; Fuzzy Numbers; Dual Program;

1 Introduction

Since the advent of the work of Charnes et at. [4], data envelopment analysis (DEA) is a methodology that has been widely used to measure relative efficiencies within a group of decision making units (DMUs) that utilize several inputs to produce a set of outputs. It has been applied to evaluate schools, hospitals and various organizations with multiple inputs and outputs. Unlike regression analysis, DEA uses only a single set of observations per DMU. Moreover, DEA is sensitive to outliers. Therefore, it is very difficult to evaluate the efficiency of DMU with inputs uncertain by conventional DEA models.

Most of the previous studies that deal with inaccurate and imprecise inputs outputs in DEA models have simply used stochastic models [1, 7, 13] or simulation methods [2]. Shortcomings of these methods are noted in [5]. Since the advent of fuzzy set theory [16], it has been proven to be useful in modeling uncertainty or imprecise value. The DEA models with fuzzy data can more realistically represent real-world problems where data may be missing or uncertain than traditional DEA models. Sengupta [14] proposed a fuzzy DEA model and solved it by Zimmermann's method [17]. Triantis et al. [15] solved their fuzzy model with the parametric approach proposed by Carlsson et al. [3]. Kao and

Liu [10, 11] transformed the fuzzy DEA model to a family of the conventional crisp DEA models by applying the α-cut approach. Hougaard [9] extended scores of technical efficiency to fuzzy intervals to allow decision maker to use scores in combination with other sources of available performance information. Entani et al. [6] developed a DEA model with an interval efficiency. Guo et al. [8] proposed a fuzzy DEA model to deal with efficiency evaluation problem with given fuzzy input and output data, and extended the efficiency to be a fuzzy number.

The fuzzy DEA model in [8] is based on the output/input ratio. Therefore, the efficiency obtained is a fuzzy number if the data of input and output are fuzzy. However, if the relative efficiency is measured in terms of the distance from the frontier in stead of the output/input ratio, the efficiency is still a crisp number even if the data are fuzzy. A crisp efficiency will be more helpful for decision makers because it provides clearer information. In this paper, we are going to derive fuzzy DEA models from this perspective.

This paper is organized as follows: Section 2 reviews traditional DEA models and their duals. In section 3, fuzzy DEA models are derived from the duals and their linear programming equivalents are presented. Numerical examples are demonstrated in section 4. Finally, concluding remarks are made in section 5.

2 The data envelopment analysis model

DEA is a mathematical model that measures the relative efficiency of DMUs with multiple inputs and outputs with no obvious production function to aggregate the data in its entirety. Relative efficiency is defined as the ratio of total weighted output over weighted input. By comparing n units with s outputs denoted by y_{rk}, $r = 1, \ldots, s$ and m inputs denoted by x_{ik}, $i = 1, \ldots, m$, the efficiency measure for DMU k is

$$\begin{array}{ll} h_k = Max \sum_{r=1}^{s} u_r y_{rk} & \\ s.t. \quad \sum_{i=1}^{m} v_i x_{ik} = 1, & \\ \sum_{i=1}^{m} v_j x_{ij} - \sum_{r=1}^{s} u_r y_{rj} \geq 0 \;\; for\ j = 1, \ldots, n, & (1) \\ u_r \geq 0 \;\; for\ r = 1, \ldots, s, & \\ v_i \geq 0 \;\; for\ i = 1, \ldots, m. & \end{array}$$

Mode (1), often referred to as the input-oriented CCR (Charnes Cooper Rhodes) model [4], assumes that the production function exhibits constant returns-to-scale. The following is referred to as the output-oriented CCR model:

$$\begin{array}{ll} \frac{1}{g_k} = Min \sum_{i=1}^{m} v_i x_{ik} & \\ s.t. \quad \sum_{r=1}^{s} u_r y_{rk} = 1, & \\ \sum_{i=1}^{m} v_j x_{ij} - \sum_{r=1}^{s} u_r y_{rj} \geq 0 \;\; for\ j = 1, \ldots, n, & (2) \\ u_r \geq 0 \;\; for\ r = 1, \ldots, s, & \\ v_i \geq 0 \;\; for\ i = 1, \ldots, m. & \end{array}$$

The BCC (Banker Chang Cooper) model (2) adds an additional constant variable, c_k, in order to permit variable returns-to-scale. The following is input-

oriented BCC model:

$$\begin{array}{l} h_k = Max \sum_{r=1}^{s} u_r y_{rk} + c_k \\ s.t. \qquad\qquad \sum_{i=1}^{m} v_i x_{ik} = 1, \\ \sum_{i=1}^{m} v_j x_{ij} - \sum_{r=1}^{s} u_r y_{rj} - c_k \geq 0 \;\; for\; j = 1, \ldots, n, \\ u_r \geq 0 \;\; for\; r = 1, \ldots, s, \\ v_i \geq 0 \;\; for\; i = 1, \ldots, m. \end{array} \tag{3}$$

Model (4) is referred to as the output-oriented BCC model:

$$\begin{array}{l} \frac{1}{g_k} = Min \sum_{i=1}^{m} v_i x_{ik} + c_k \\ s.t. \qquad\qquad \sum_{r=1}^{s} u_r y_{rk} = 1, \\ \sum_{i=1}^{m} v_j x_{ij} - \sum_{r=1}^{s} u_r y_{rj} + c_k \geq 0 \;\; for\; j = 1, \ldots, n, \\ u_r \geq 0 \;\; for\; r = 1, \ldots, s, \\ v_i \geq 0 \;\; for\; i = 1, \ldots, m. \end{array} \tag{4}$$

If a DMU proves to be inefficient, a combination of other efficient units can produce either greater output for the same composite of inputs, use fewer inputs to produce the same composite of outputs or some combination of the two. A hypothetical decision making unit, k', can be composed as an aggregate of the efficient units referred to as the efficient reference set for inefficient unit k. The solution to the dual problem of above models directly computes the multipliers required to compile aggregated efficient unit k'. Therefore, we will focus on the dual models. The dual of model (1) is:

$$\begin{array}{l} h_k = Min\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj} x_{ij} - \theta_k x_{ik} \leq 0 \quad for\; i = 1, \ldots, m, \\ \sum_{j=1}^{n} \lambda_{kj} y_{rj} \geq y_{rk} \; for\; r = 1, \ldots, s, \\ \lambda_{kj} \geq 0 \quad for\; j = 1, \ldots, n. \end{array} \tag{5}$$

The dual of model (2) is:

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj} x_{ij} - x_{ik} \leq 0 \quad for\; i = 1, \ldots, m, \\ \sum_{j=1}^{n} \lambda_{kj} y_{rj} \geq \theta_k y_{rk} \; for\; r = 1, \ldots, s, \\ \lambda_{kj} \geq 0 \quad for\; j = 1, \ldots, n. \end{array} \tag{6}$$

The dual of model (3) is:

$$\begin{array}{l} h_k = Min\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj} x_{ij} - \theta_k x_{ik} \leq 0 \quad for\; i = 1, \ldots, m, \\ \sum_{j=1}^{n} \lambda_{kj} y_{rj} \geq y_{rk} \; for\; r = 1, \ldots, s, \\ \sum_{j=1}^{n} \lambda_{kj} = 1. \end{array} \tag{7}$$

The dual of model (4) is:

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj} x_{ij} - x_{ik} \leq 0 \quad for\; i = 1, \ldots, m, \\ \sum_{j=1}^{n} \lambda_{kj} y_{rj} \geq \theta_k y_{rk} \; for\; r = 1, \ldots, s, \\ \sum_{j=1}^{n} \lambda_{kj} = 1. \end{array} \tag{8}$$

3 Fuzzy DEA models

In this section, we are going to extend models (5)-(8) to fuzzy environment.

Definition 1. *The α-cut of fuzzy set $\tilde{A}$, $\tilde{A}^\alpha$, is the crisp set $\tilde{A}^\alpha = \{x|\mu_{\tilde{A}}(x) \geq \alpha\}$. The support of $\tilde{A}$ is the crisp set $Supp(\tilde{A}) = \{x|\mu_{\tilde{A}}(x) > 0\}$. $\tilde{A}$ is normal iff $sup_{x\in U}\mu_{\tilde{A}}(x) = 1$, where U is the universal set.*

Definition 2. *A fuzzy subset $\tilde{A}$ of real number R is convex iff*

$$\mu_{\tilde{A}}(\lambda x + (1-\lambda)y) \geq (\mu_{\tilde{A}}(x) \wedge \mu_{\tilde{A}}(y)), \forall x, y \in R, \forall \lambda \in [0,1],$$

where $\wedge$ denotes the minimum operator.

Definition 3. *$\tilde{A}$ is a fuzzy number iff $\tilde{A}$ is a normal and convex fuzzy subset of R.*

Definition 4. *A triangular fuzzy number $\tilde{A}$ is a fuzzy number with piecewise linear membership function $\mu_{\tilde{A}}$ defined by*

$$\mu_{\tilde{A}}(x) = \begin{cases} \frac{x-a_L}{a_M-a_L}, & a_L \leq x \leq a_M, \\ \frac{a_R-x}{a_R-a_M}, & a_M \leq x \leq a_R, \\ 0, & otherwise, \end{cases}$$

which can be denoted as a triplet (a_L, a_M, a_R).

Definition 5. *Let $\tilde{A}$ and $\tilde{B}$ be two fuzzy numbers. Let $\circ$ be a operation on real numbers, such as +, -, *, $\wedge$, $\vee$, etc. By extension principle, the extended operation $\circ$ on fuzzy numbers can be defined by*

$$\mu_{\tilde{A}\circ\tilde{B}}(z) = \sup_{x,y:z=x\circ y} \{\mu_{\tilde{A}}(x) \wedge \mu_{\tilde{B}}(y)\}. \tag{9}$$

Definition 6. *Let $\tilde{A}$ and $\tilde{B}$ be two fuzzy numbers. $\tilde{A} \leq \tilde{B}$ if only if $\tilde{A} \vee \tilde{B} = \tilde{B}$.*

Definition 7. *Let $\tilde{A}$ be a fuzzy number. Then $\tilde{A}_\alpha^L$ and $\tilde{A}_\alpha^U$ are defined as*

$$\tilde{A}_\alpha^L = \inf_{\mu_{\tilde{A}}(z)\geq\alpha}(z) \tag{10}$$

and

$$\tilde{A}_\alpha^U = \sup_{\mu_{\tilde{A}}(z)\geq\alpha}(z) \tag{11}$$

respectively.

Assume inputs and outputs of DMUs are represented by triangular fuzzy numbers. Fuzzy DEA models correspond to (5)-(8) are shown as follows. Fuzzy model of (5) is

$$\begin{aligned} & h_k = Min\theta_k \\ & s.t. \sum_{j=1}^n \lambda_{kj}\tilde{X}_{ij} - \theta_k\tilde{X}_{ik} \leq 0 \quad for\ i = 1,\ldots,m, \\ & \qquad \sum_{j=1}^n \lambda_{kj}\tilde{Y}_{rj} \geq \tilde{Y}_{rk} \ for\ r = 1,\ldots,s, \\ & \qquad \lambda_{kj} \geq 0 \quad for\ j = 1,\ldots,n. \end{aligned} \tag{12}$$

Fuzzy model (6) is

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj}\tilde{X}_{ij} - \tilde{X}_{ik} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \qquad\qquad \sum_{j=1}^{n} \lambda_{kj}\tilde{Y}_{rj} \geq \theta_k \tilde{Y}_{rk}\ for\ r = 1, \ldots, s, \\ \qquad\qquad\qquad \lambda_{kj} \geq 0 \qquad for\ j = 1, \ldots, n. \end{array} \tag{13}$$

Fuzzy model (7) is

$$\begin{array}{l} h_k = Min\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj}\tilde{X}_{ij} - \theta_k \tilde{X}_{ik} \leq 0 \quad for\ i = 1, \ldots, m, \\ \qquad\qquad \sum_{j=1}^{n} \lambda_{kj}\tilde{Y}_{rj} \geq \tilde{Y}_{rk}\ for\ r = 1, \ldots, s, \\ \qquad\qquad\qquad \sum_{j=1}^{n} \lambda_{kj} = 1. \end{array} \tag{14}$$

Fuzzy model (8) is

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t. \sum_{j=1}^{n} \lambda_{kj}\tilde{X}_{ij} - \tilde{X}_{ik} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \qquad\qquad \sum_{j=1}^{n} \lambda_{kj}\tilde{Y}_{rj} \geq \theta_k \tilde{Y}_{rk}\ for\ r = 1, \ldots, s, \\ \qquad\qquad\qquad \sum_{j=1}^{n} \lambda_{kj} = 1. \end{array} \tag{15}$$

Because of the convexity of fuzzy numbers, we can have the following definition equivalent to definition 6.

Definition 8. *Let $\tilde{A}$ and $\tilde{B}$ be two fuzzy numbers. $\tilde{A} \leq \tilde{B}$ if only if $\tilde{A}_\alpha^L \leq \tilde{B}_\alpha^L$ and $\tilde{A}_\alpha^R \leq \tilde{B}_\alpha^R$ for $\forall \alpha \in [0,1]$.*

Lemma 1. *Let $\tilde{A}$ and $\tilde{B}$ be two triangular fuzzy numbers denoted by (a_L, a_M, a_R) and (b_L, b_M, b_R) respectively. Then $\tilde{A} \leq \tilde{B}$ if only if $a_L \leq b_L$, $a_M \leq b_M$ and $a_R \leq b_R$.*

Proof: By definition 4 and 7, we have

$$\begin{array}{c} \tilde{A}_\alpha^L = \alpha(a_M - a_L) + a_L, \\ \tilde{A}_\alpha^R = a_R - \alpha(a_R - a_M), \\ \tilde{B}_\alpha^L = \alpha(b_M - b_L) + b_L, \\ \tilde{B}_\alpha^R = b_R - \alpha(b_R - b_M), \end{array}$$

If for all $\alpha \in [0,1]$, $\tilde{A}_\alpha^L \leq \tilde{B}_\alpha^L$ and $\tilde{A}_\alpha^R \leq \tilde{B}_\alpha^R$ then

$$\alpha(a_M - a_L) + a_L \leq \alpha(b_M - b_L) + b_L$$

and

$$a_R - \alpha(a_R - a_M) \leq b_R - \alpha(b_R - b_M).$$

Let $\alpha = 1$. We have $a_M \leq b_M$. Let $\alpha = 0$. We have $a_L \leq b_L$ and $a_R \leq b_R$.

If $a_L \leq b_L$, $a_M \leq b_M$ and $a_R \leq b_R$ then

$$\alpha(a_M - a_L) + a_L \leq \alpha(b_M - b_L) + b_L$$

and

$$a_R - \alpha(a_R - a_M) \leq b_R - \alpha(b_R - b_M)$$

for $\alpha \in [0,1]$. That is, $\tilde{A}_\alpha^L \leq \tilde{B}_\alpha^L$ and $\tilde{A}_\alpha^R \leq \tilde{B}_\alpha^R$ for all $\alpha \in [0,1]$. □

Let $\tilde{X}_{ij} = (x_{ij_L}, x_{ij_M}, x_{ij_R})$ and $\tilde{Y}_{rj} = (y_{rj_L}, y_{rj_M}, y_{rj_R})$. Following lemma 1, we can rewrite fuzzy model (12) as follows:

$$\begin{array}{l} h_k = Min\theta_k \\ s.t. \ \sum_{j=1}^n \lambda_{kj} x_{ij_L} - \theta_k x_{ik_L} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_M} - \theta_k x_{ik_M} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_R} - \theta_k x_{ik_R} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_L} \geq y_{rk_L} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_M} \geq y_{rk_M} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_R} \geq y_{rk_R} \quad for\ r = 1, \ldots, s, \\ \quad \lambda_{kj} \geq 0 \qquad for\ j = 1, \ldots, n. \end{array} \tag{16}$$

Fuzzy model (13) can be equivalently rewritten as model (17):

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t. \ \sum_{j=1}^n \lambda_{kj} x_{ij_L} - x_{ik_L} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_M} - x_{ik_M} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_R} - x_{ik_R} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_L} \geq \theta_k y_{rk_L} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_M} \geq \theta_k y_{rk_M} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_R} \geq \theta_k y_{rk_R} \quad for\ r = 1, \ldots, s, \\ \quad \lambda_{kj} \geq 0 \qquad for\ j = 1, \ldots, n. \end{array} \tag{17}$$

Fuzzy model (14) can be equivalently rewritten as model (18):

$$\begin{array}{l} h_k = Min\theta_k \\ s.t. \ \sum_{j=1}^n \lambda_{kj} x_{ij_L} - \theta_k x_{ik_L} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_M} - \theta_k x_{ik_M} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} x_{ij_R} - \theta_k x_{ik_R} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_L} \geq y_{rk_L} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_M} \geq y_{rk_M} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} y_{rj_R} \geq y_{rk_R} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^n \lambda_{kj} = 1. \end{array} \tag{18}$$

Fuzzy model (15) can be equivalently rewritten as model (19):

$$\begin{array}{l} \frac{1}{g_k} = Max\theta_k \\ s.t.\ \sum_{j=1}^{n} \lambda_{kj} x_{ij_L} - x_{ik_L} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^{n} \lambda_{kj} x_{ij_M} - x_{ik_M} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^{n} \lambda_{kj} x_{ij_R} - x_{ik_R} \leq 0 \qquad for\ i = 1, \ldots, m, \\ \quad \sum_{j=1}^{n} \lambda_{kj} y_{rj_L} \geq \theta_k y_{rk_L} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^{n} \lambda_{kj} y_{rj_M} \geq \theta_k y_{rk_M} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^{n} \lambda_{kj} y_{rj_R} \geq \theta_k y_{rk_R} \quad for\ r = 1, \ldots, s, \\ \quad \sum_{j=1}^{n} \lambda_{kj} = 1. \end{array} \tag{19}$$

To solve fuzzy models (12)-(15), we solve their equivalent linear programming problems (16)-(19).

Theorem 1. *Efficiencies given by fuzzy models (12)-(15) are all no greater than 1. That is,* $h_k \leq 1$ *and* $g_k \leq 1$.

Proof: For models (12) and (14), let

$$\lambda_{kj} = \begin{cases} 1\ if\ j = k \\ 0\ if\ j \neq k. \end{cases}$$

Then $\theta_k = 1$. Let θ_k^* be the optimal objective value of models (12) and (13). We have

$$\theta_k^* \leq \theta_k = 1.$$

That is, $h_k \leq 1$.

Similarly, for models (13) and (15), let

$$\lambda_{kj} = \begin{cases} 1\ if\ j = k \\ 0\ if\ j \neq k. \end{cases}$$

Then $\theta_k = 1$. Let θ_k^* be the optimal objective value of models (13) and (15). We have

$$\theta_k^* \geq \theta_k = 1.$$

That is, $g_k \leq 1$. □

4 Numerical examples

First, we consider DMUs with one fuzzy input and one fuzzy output as shown in table 1. The efficiencies of DMUs in table 1 measured by fuzzy DEA model (12) is given in table 2.

Next we demonstrate our fuzzy DEA with a more complicated example shown in table 3 where DMUs are assumed to have two inputs and two outputs. The efficiencies of DMUs in table 3 measured by fuzzy DEA model (12) are shown in table 4.

Table 1. DMUs with one fuzzy input and one fuzzy output

DMUs	A	B	C	D	E
inputs	(1.5,2.0,2.5)	(2.5,3.0, 3.5)	(2.4,3.0,3.6)	(4.0,5.0,6.0)	(4.5,5.0,5.5)
outputs	(0.7,1.0,1.3)	(2.3,3.0,3.7)	(1.6,2.0,2.4)	(3.9,4.0,4.1)	(1.8,2.0,2.2)

Table 2. Efficiencies of DMUs in table 1 measured by fuzzy DEA model (12)

	A	B	C	D	E
h_k	0.5855856	1.0	0.7166667	1.0	0.4980237

Table 3. DMUs with two fuzzy inputs and two fuzzy outputs

DMUs	A	B	C	D	E
input 1	(3.5,4.0,4.5)	(2.9,2.9,2.9)	(4.1,4.9,5.4)	(3.4,4.1,4.8)	(5.9,6.5,7.1)
input 2	(1.9,2.1,2.3)	(1.4,1.5,1.6)	(2.2,2.6,3.0)	(2.2,2.3,2.4)	(3.6,4.1,4.6)
output 1	(2.4,2.6,2.8)	(2.2,2.2,2.2)	(2.7,3.2,3.7)	(2.5,2.9.3.4)	(4.4,5.1,5.8)
output 2	(3.8,4.1,4.4)	(3.2,3.5,3.7)	(4.3,5.1,5.9)	(5.5,5.7,5.9)	(6.5,7.4,8.3)

Table 4. Efficiencies of DMUs in table 3 measured by fuzzy DEA model (12)

	A	B	C	D	E
h_k	0.9279023	1.0	1.0	1.0	1.0

5 Conclusions

Traditional DEA deals with crisp data. When data are difficult to measure, fuzzy numbers may be introduced to represent the vagueness or uncertainty of the data. In this paper, we extended the duals of DEA models to fuzzy environment. In our extension, the efficiency measured with fuzzy data is a crisp value which enables decision makers to decide whether a DMU is relative efficient or not very clear. It is also obvious that our fuzzy models encompass conventional DEA models.

Acknowledge

This research work was partially supported by the National Science Council of the Republic of China under grant No. NSC92-2416-H-019-002-.

References

1. P.W. Bauer, Recent developments in the econometric estimation of frontiers, J. Econometrics 46 (1990) 39-56.
2. R.D. Banker, H. Chang, W.W. Cooper, Simulation studies of efficiency, returns to scale and misspecification with nonlinear functions in DEA, Ann. Oper. Res. 66 (1996) 233-253.
3. C. Carlsson, P. Korhonen, A paramet4ric approach to fuzzy linear programming, Fuzzy Sets and Systems 20 (1986) 17-30.
4. A. Charnes, W.W. Cooper, E. Rhodes, Measuring the efficiency of decision-making units, European J. Oper. Res. 2 (1978) 429-444.
5. W.W. Cooper, K. Tone, Measures of inefficiency in data envelopment analysis and stochastic frontier estimation, European J. Oper. Res. 99 (1997) 72-88.
6. T. Entani, Y. Maeda, H. Tananka, Dual models of interval DEA and its extension to interval data, European J. Oper. Res. 136 (2002) 32-45.
7. W.H. Greene, A Gamma-distributed stochastic frontier model, J. Econometrics 46 (1990) 141-163.
8. P. Guo, H. Tanaka, Fuzzy DEA: a perceptual evaluation method, Fuzzy Sets and Systems 119 (2001) 149-160.
9. J.L. Hougaad, Fuzzy scores of technical efficiency, European J. Oper. Res. 115 (1999) 529-541.
10. C. Kao, S.T. Liu, Data envelopment analysis with missing data: an application to University libraries in Taiwan, Journal of the Operational Research Society 51 (2000) 897-905.
11. C. Kao, S.T. Liu, Fuzzy efficiency measures in data envelopment analysis, Fuzzy Sets and Systems 113 (2000) 429-437.
12. S. Lerworasirikul, S.-C. Fang, J.A. Jonies, H. L.W. Nuttle, Fuzzy data envelopment analysis (DEA): a possibility approach, Fuzzy Sets and Systems 139 (2003) 379-394.
13. P. Schmidt, T.F. Lin, Simple tests of alternative specifications in stochastic frontier models, J. Econometrics 24 (1984) 349-31.
14. J.K. Sengupta, A fuzzy systems approach in data envelopment analysis, Comput. Math. Appl. 24 (1992) 259-266.
15. K. Triantis, O. Girod, A mathematical programming approach for measuring techincal efficiency in a fuzzy environment, J. Prod. Anal. 10 (1998) 85-102.

16. L.A. Zadeh, Fuzzy sets, Information and Control 8 (1965) 338-353.
17. H.J. Zimmermann, Description and optimization of fuzzy system, Interat. J. General System 2 (1976) 209-216.

Optimization of Complex Systems by Processes of Self-modeling

Christina Stoica[1], Jürgen Klüver[2] & Jörn Schmidt[3]

[1] University of Duisburg-Essen, Information Technologies and Educational Processes, 45117 Essen, Germany
christina.stoica@uni-essen.de
http://www.cobasc.de
[2] University of Duisburg-Essen, Information Technologies and Educational Processes, 45117 Essen, Germany
juergen.kluever@uni-essen.de
http://www.cobasc.de
[3] University of Duisburg-Essen, Information Technologies and Educational Processes, 45117 Essen, Germany
joernschmidt@uni-essen.de
http://www.cobasc.de

Abstract. The hybrid system SOCAIN is described as a formal model of the self-modeling of complex systems. The system consists of a stochastic cellular automaton (CA) coupled with a genetic algorithm (GA) and an interactive net (IN), also coupled with a GA. The IN is understood as the self-model of the CA. It can be shown that the adaptive success of the whole system SOCAIN is much greater than the successes of both subsystems alone. Reasons for this are given and the logic of the system is discussed both logically and sociologically.

1. Introduction

Frequently the problem of the modeling of complex systems is best solved by using so-called Soft-Computing techniques [16],[20]. The term "Soft-computing" describes different formal techniques like neural networks (NN), evolutionary algorithms (EA), fuzzy set theory and fuzzy logic, cellular automata (CA) and Boolean networks (BN). The common characteristic of these on first sight very different models is their orientation to real processes, i.e. biological, social or cognitive ones. In other words, Soft-Computing models use the logic of complex real systems in order to capture the basic features of real systems as adequately as possible.

Soft-Computing models are very powerful tools because they are potentially equivalent to Universal Turing machines [17]. The potential logical universality of cellular automata for example was proven for the special case of the Game of Life [1]. That means, according to the physical Church-Turing-hypothesis, that each real system can be modeled as completely as the research question demands it by a suited Soft-Computing system like e.g., cellular automata [23]. The power of such systems

can even be increased if several of them are combined in form of so called hybrid systems.

In order to avoid misunderstandings one should remember that not all Soft Computing Systems are Universal Turing machines but only those with suited rules of interactions. Most cellular automata are much too simple to generate the complex behavior of Universal Turing machines. That is why we speak of potential Universal Turing machines, meaning that it is always possible to construct a special soft computing model logically equivalent to Universal Turing machines.

Hybrid systems are simulation programs, which consist of two - or even more - different subprograms; these are coupled together to fulfill tasks, which neither of them could do alone or only worse than the whole hybrid system. Goonatilake and Khebbal ([6], p.7) proposed a general classification scheme for hybrid systems: They differentiate between (a) "function replacing hybrids" and (b) "intercommunicating hybrids" (we skip their third category, because that is no hybrid at all). For reasons we cannot discuss here we prefer the terms "vertically coupled systems" and "horizontally coupled systems" [cf. 14]. Vertically coupled systems are combinations of programs where one program is operating upon the rules and/or parameters of the second. In such cases, according to the concepts of logical semantics [21], we speak of the first system as the meta-system and of the second as the base system. An example for a vertically coupled system, which is described below is the hybrid system SOZION, a combination of a cellular automaton (CA) as the base system and a genetic algorithm (GA) as the meta-system. The GA modifies the rules, especially certain parameters according to certain evaluation criteria. Neural nets that are combined with GAs in order to optimize the network's topology are also vertically coupled systems.

Horizontally coupled systems operate in a sort of division of labor insofar as one system does one part of a job, transferring its results to the second system, and the second system performs another part of the task. The two systems are operating on the same logical level, in contrast to the case of vertical coupling. An example of a horizontally coupled system is THEOPRO [13], which consists of a knowledge-based system and a neural net. The knowledge based system constructs a special net, consisting of concepts of a theory of society, and the according weight matrix; the neural net performs simulations of social processes according to the theory and gives its results back to the knowledge based system. Of course, with horizontally coupled systems the problem arises whether the results of one system are compatible with the results of the second, as the two systems normally operate with very different rule sets. We shall return to this problem in section 4.

We have by now created higher order hybrids by combining hybrids of these two types: The program SOCAIN – Society model by Cellular Automata and Interactive Net - contains a CA, horizontally coupled with an interactive neural net (IN), and GAs operating on both base systems, i.e. the IN and the CA. In this case we have a horizontally-vertically-coupled system.

Hybrid systems offer important possibilities for the modeling and simulating of complex systems. These shall be demonstrated with the problem of the self-modeling of complex systems.

2. The Problem

Since Hofstadter [8] it has become common to speak of complex systems as "self referential systems". It is not possible here to define this and related concepts precisely (see e.g. [18]); apparently, "self-reference" contains several different aspects among which the concept of "self-modeling" is a salient feature. This means that complex systems like language, mind, or society are able to construct models of themselves as parts of their different states. One can speak about language by using language and construct models of it, the mind can model itself by thinking about thinking, and social systems build models of themselves by mapping their structures onto parts of them. Two famous examples shall illustrate this characteristic for the case of social systems:

(a) As part of his ethnological studies, Geertz described the institution of the Balinese cockfight as a simulation of the "social matrix" of the Balinese society in the sense that the important structural aspects of the society are mapped - mirrored - onto the institution of the cockfight. Among these are class distinctions, the inequality of wealth and the hierarchy between the sexes. The cockfight is an important part of the Balinese culture and as a part it constitutes a model of the whole society - "it is a story they tell themselves about themselves" ([5], p.26).

(b) In their studies about the French higher education system, and especially the French universities, Bourdieu and Passeron [2] analyze the education system as a "reproduction" of the society in general. With "reproduction" they imply two aspects: on the one hand, the structures of society, especially the structures of inequality, are mirrored in the subsystem of education; on the other hand, the educational process, determined by the structures of the educational system, reproduces the structures of inequality as a function for the whole society. Of course, the educational system can only do this because its structures are reflections of societal structures in general and are originated this way. Thus, the educational system as a part of society is also a model of society. The importance of social reproduction via modeling the society as a whole by a part of society has always been stressed by theorists who are influenced by Marx (e.g. [7]), though not in these terms.

Self-reference can easily lead into logical paradoxes, as it is well known since the logical and set theoretical antinomies. That is why self-reference in formal models is inadmissible in logical semantics [21]. But, as we have seen, because complex systems like societies, minds or language possess self-reference and especially self-modeling as two of their most fundamental aspects. We can assume that the main difference between the systems of the natural sciences on the one hand and systems like society, mind, or language - the topics of the humanities - on the other hand lies exactly in these features. Of course, physical or biological systems are very often complex too. Social and cognitive systems just have an additional dimension of complexity. If formal models cannot capture these aspects then the construction of formal models is indeed, as many theoretical sociologists believe, not suited for the analysis of social systems. Therefore, the most important task is to show that it is possible despite the cautions of the logical semanticists to construct consistent models and corresponding computer programs with the ability of self-modeling.

Probably because of the problem of the logical paradoxes with respect to self-referentiality computer scientists have started to analyze self-referential systems only

since the last decade. In 1981, for example, Hofstadter and Dennett observed that to model such phenomena with computer programs seems like " a kind of magic trick which we feel is very close to the core of consciousness. It will perhaps prove one day to be a key element ... in the approach toward artificial intelligence." ([9], p. 380) To be sure, hybrid systems, consisting of different Soft-Computing models, are no "magic trick" but they seem to be a promising possibility to model self-referential processes without risking the danger of logical paradoxes.

In order to understand why hybrid systems can avoid to get caught in the traps of the well-known logical paradoxes one has to remember that such systems are no static structure but dynamical systems like the brain. The brain, as we know, can also avoid logical paradoxes by introducing the famous difference between object and meta-level – if it is needed. Consider the antinomy of the Barber as Russell has described the paradox of "the set of all sets that do not contain themselves as elements". The barber is the man who shaves all and only those men of the village who do not shave themselves. Who shaves the barber? By assuming that the barber does not shave himself the conclusion is that he shaves himself and vice versa.

The solution of this paradox is that the concept of "barber" in this definition belongs to another category. In other words, the "barber" is not on the same logical level of "men who do not shave themselves". Therefore, the "barber" is not an element of the set of men that constitute the potential paradoxically category, but he is something like a "meta-person". That is the well-known solution of the antinomy of the "set of all sets that do not contain themselves as an element".

The brain, of course, is able to introduce this distinction between the barber and the men of the village in order to avoid confusion by logical paradoxes. A logically hybrid system can basically do the same: if the meta-level of the system perceives that there are logical paradoxes on the base level, the whole system does not break down but analyses its base level in order to remove the contradicting elements from the base level – perhaps by transforming them to one meta level, generating new categories that contain the problematic elements and so on. *This* dealing with logical paradoxes certainly is valid only for such particular cases and is no guarantee that other logical paradoxes will not occur some time. But that is just the way the human mind deals with cases like that. It solves the problems at hand and deals with other problems only when they arise. With a famous remark by John Holland, the inventor of the genetic algorithm, one can say that hybrid systems of this sort are "muddling through" like the human mind does.

It is possible to demonstrate the same way that such hybrid systems are also principally able to overcome the restrictions of Gödel's incompleteness theorem in a very similar way [14]. They introduce new elements – new axioms in the case of axiomatic systems – to the base system if the incompleteness *with respect to certain problems* is perceived. Then the base system is able to solve the respective problems. To be sure, the whole base system will remain incomplete, for Gödel's theorem is still valid *in principle*. But a hybrid system is able to "outgödel" [3] an incomplete system by completing it as much as the problem demands. It is just another example of "muddling through".

3. The Program SOCAIN

As mentioned above, SOCAIN is a horizontally-vertically-coupled system, which consists of two hybrid systems. The base systems are a CA and an IN, the meta-system in both systems is a GA.

The CA is rather simple because it belongs to a formal analogue of Wolfram class II [22],[23]; that means that it will reach an attractor state after a limited number of time steps and will remain there (a point attractor or an attractor with only short periods). Its rules are "totalistic": The transition of a cell is determined by computing the mean value of the cells of the neighborhood (a Moore neighborhood). In contrast to most CAs, the rules are not deterministic but stochastic; changing of a state value only occurs with a certain probability. The state of the cells can take one of nine values - which one depends on the neighborhood and on the probability values which define the relations between the different states. The different values of the cell states symbolize different social classes, i.e., each cell represents a member of a society who belongs to a certain social class. There are nine different classes, which is optional. Because the optimization process of the whole system was of particular interest to us the number of nine bears no particular significance.

The theoretical purpose of the model was to test a particular theory of social evolution, i.e. the theory of social differentiation. The assumption of this theory (cf. e.g. [4]) is that social evolution occurred in terms of the emerging of different social roles that are aggregated in according social classes. The causes of the emerging of social classes are mostly seen in two external factors: overpopulation on the one hand and difficult environmental conditions on the other hand like for example changing climate. In other words, the respective societies had to solve environmental problems by generating new forms of social order that were more effective than the old ones of hunter-gatherer societies. The structural solutions literally *all* societies found when confronted with such problems was the generation of social classes, containing specialized social roles. The advantages of this solution are that the occupants of these roles because of their specialization are more effective than people who are not specialized. In addition, the vertical ordering of the classes into "higher" and "lower" ones obtains a decision structure of the whole society that also is more effective – although not as democratic – than the egalitarian structure of the hunter-gatherer societies.

The CA was constructed to simulate this social differentiation, namely the differentiation of a homogenous society into a class society (see [14]). In the original version the CA was coupled with a GA, which modified the probability values and decided on switching any of the rules on or off. We chose a stochastic CA because we believe that complex systems tend to avoid too drastical changes. Before they abandon whole social structures they try to optimize by just varying certain probability values of their rules.

The IN is a recurrent network. Each unit is connected with all others, but the weight values of the network are not changed during the runs. This makes INs suitable for simulations of special social and cognitive processes [19], although INs are not able to learn by themselves. Our IN consists of nine units and, accordingly, a weight matrix of 9 * 9 = 81 weight components. A run of the IN begins with an

"external activation" of one ore more units; the run ends when the activation values of the units stabilize. The IN uses the standard linear activation rule

$$A_i = \Sigma_j A_j * w_{ij} \quad (1)$$

where A_i is the activation value of the receiving unit i, A_j is the activation value of a sending unit j and w_{ij} is the weight value of the relation between the units j and i. Other activation rules are possible, especially nonlinear ones, but we wanted to keep the whole system as simple as possible.

The GA is also standardized. It contains the usual genetic operators of crossover and mutation and is non-elitistic. That means that the best results from a GA-run are not preserved, though elitistic GAs often do better than non elitistic ones. Its evaluation function, i.e. the algorithm for choosing the "best" solution, is described below. The GA operates on vectors in which those rules of the base systems are represented that are to be changed; the vectors use simple numerical codes, i.e. integers for the CA-rules and real numbers for the IN (these are the values of the weight matrix).

To be more exact, the GA of the CA operates the following way: the genetic operators – mutation and crossover – operate on a "rule-changing vector". The components of this vector are either 1, 0 or –1. Each component represents either the antecedens or the action part of one of the transition rules; in addition, other components represent the elements of a probability matrix A (see below). 1 means that an antecedens part is valid and the same holds for the action part. Different combinations of antecedens and action part values determine the transitions (see below). In all other cases the rule is "locked", i.e., nothing happens. In the case of the elements of the probability matrix, 1 means the increase of the respective probability value, 0 means that the value remains constant and –1 means the decrease. In the case of the hybrid IN the GA operates on a vector where the components are nothing else than the elements of the weight matrix, i.e., the weight values are changed via the genetic operators.

The two systems are coupled in the following way: The transformation of cell states within the CA is determined by the probability matrix", i.e. the 9 * 9 matrix containing the transition probabilities from each of the nine possible cell states to any other. The GA of this first hybrid system operates on the transition *rules*, allowing or forbidding certain transitions, as well as on the transition *probabilities*, decreasing or increasing those values. The CA always runs three steps; then the "success" of the development of the CA is evaluated. Afterwards the GA starts its changing operations.

Cell transformation in the CA proceeds according to the following rules:

1. A cell with index value i **increases** its index value (ascendence),
   i. if the average of cell indices in its Moore neighborhood is greater or equal i (u≥i) **and**
   ii. if the corresponding component in the antecedens part of the rule vector has the value 1 or 3 (neutral element) **and**
   iii. if the corresponding component of the action part of the rule vector is 1 **and**
   iv. if the probability of ascendence according to the procedure defined below is given.

2. Correspondingly a cell **decreases** its index value (descendence),
   i. if the average of cell indices in its Moore neighborhood is less than i (u<i) **and**
   ii. if the corresponding component in the antecedence part of the rule vector has the value -1 or 0 (neutral element) **and**
   iii. if the corresponding component of the action part of the rule vector is 1 **and**
   iv. if the probability of descendence according to the procedure defined below is given.
3. Otherwise the cell does not transform.

To calculate the transformation probability the elements of the rows i of the upper and lower triangle of the probability matrix **A** (representing the probabilities for ascendence and descendence) separately are consecutively added according to

$$c_{ij} = \Sigma a_{ik} \qquad (2)$$

where k runs from i+1 to j (upper triangle) or from j to i-1 (lower triangle), forming the cumulated probability matrix **C** (c_{ij}=0).

A random number ($0.0 \leq r < 1.0$) is generated and the cell index is transformed to the first index j - beginning from the main diagonal of the cumulated probability matrix - with c_{ij} greater than the random number; if no element c_{ij} fulfills the condition, the cell is not transformed.

After 10 runs (CA+GA) the optimized best probability matrix is taken over by the second hybrid system as the initial weight matrix of the IN. In addition, the relative numbers of cells in each cell state of the actual CA serve as external activations of the IN. The GA of the second system optimizes the weight matrix with reference to its distance to the "target vector", which is the very same as that of the first system (see below). The timing of this system is: the IN runs several time steps (usually 50-80) until it reaches an attractor state. Then the GA starts its operations. The best optimized weight matrix - and only this matrix - is then fed back to the first system, and so on.

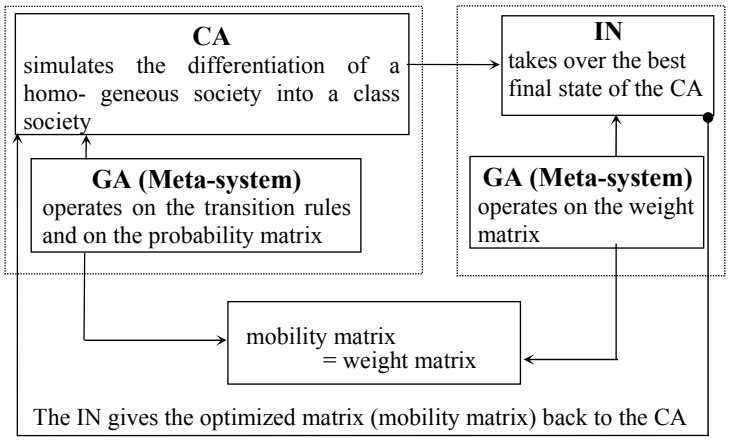


Fig. 1. The structure of the hybrid system SOCAIN

In more detail, SOCAIN - SOcial Cellular Automaton with an INteractive Network - always starts with the CA that symbolizes the "real" and whole society. It has one initial state, namely a homogenous state with only the lowest cell state representing rural people. The GA uses 20 "rule vectors" that correspond to 20 rule sets for the CA, or, briefly expressed, to 20 different CA. The selection of the best of the different CAs is based on the evaluation vector

$$V = (\Sigma_i A_i * B_{i1} * F_1 , ..., \Sigma_i A_i * B_{i6} * F_6 \quad (3)$$

where V is the "value" of the social system, A_i is the number of cells in the state i and therefore the number of members of the class i, B_{ij} is the contribution of the class i to the system function j, and F_j is a weight factor representing the value that the function j has to the whole system (this is explained in more detail in [14]). The selection criterion for the GA is the smallest possible distance d of the resulting vector V to a "target vector" or "environment vector" E:

$$d = | E - V | \quad (4)$$

After the GA has operated a certain number of times on the CA, the CA maps itself onto an IN in the way described above. The initial state of the IN, that is the initial activation values of the units, is given by the best final state of the CA before the mapping CA $\rightarrow$ IN. Then the IN starts its runs, gets optimized by the corresponding GA, and gives a modified matrix back to the CA which starts again as before. This procedure is repeated until a sufficiently low value of d is achieved for the whole system. The IN is a model of the CA insofar as the mobility matrix of the CA and a global measure of its last state are directly transformed into the IN. Thus the IN reproduces the structure of the CA. But it is a reductive model because on the one hand, the important geometry of the CA has no counterpart in the IN - it operates only on a macro level from the view of the CA. The CA operates on a micro level and gets its macro level in form of the IN as a kind of aggregation effect. The GA of the second system, on the other hand, can only change the weight values of the IN-matrix and does not affect the transition rules as in the case of the CA. Moreover, it has to be pointed out that the basic algorithms of the two base systems CA and IN are totally different.

4. Results

SOCAIN was compared to the original – SOZION = CA + GA, where there is no coupling to the IN - with reference to its effectiveness to minimize d. The results are quite remarkable:

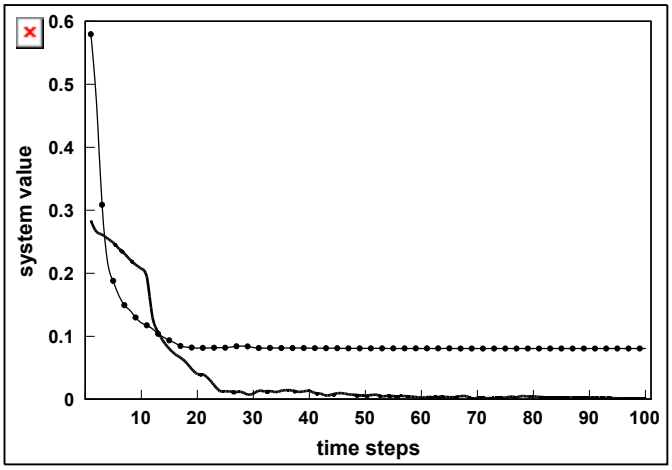


Fig. 2. Optimization with IN and without IN (dotted) in 2 typical runs of SOCAIN.

SOZION typically reaches optimal distances of about 0.08 after 50 to 100 time steps, a result, which is not improved by further optimization. The reason is that the set of rule changing vectors of the GA tends to become nearly homogenous, so that only higher mutation rates can lead to variation. But "pure" mutations, as is generally known, in most cases generate worse results.

Combining SOZION with the second system - SOCAIN - leads to much lower distances - typically lower than 0.01 - within fewer time steps. Surprisingly, the optimization process within the second system, i.e. the GA operating on the IN, is in itself much less effective; it rarely achieves distances below 0.2. In particular, often the hybrid IN even decreases its results: after having reached values of approx. 0.2 the results start to get worse till d = 0.4 Nevertheless, these sub-optimal weight matrices are capable of considerably improving the optimization process within the first system when taken over by this. Apparently, the IN/GA system is a very suitable model for the CA/GA system, improving its operation by "pushing" the values of the mobility matrix.

These results are rather astonishing at first sight since CA and IN operate with very different kinds of rules. This has to be explained in connection with particular ordering parameters, which we call "meta-parameters":
Logically speaking, adaptive systems do not have one set of rules on one logical level but at least two different sets of rules: The one set of rules constitutes the *base system,* i.e. the whole of the rule governed interactions and actors of the "real" system. The

second set contains the *meta-rules* of the system, i.e. the rules by which the base rules are changed. That is why we call adaptive systems *hybrid systems* in a logical sense:

It is well known that the rule sets of base systems can be classified by so-called ordering parameters [11],[12]. Ordering parameters are numerical values of the *rule set* of a particular system. Consider for example the rules of a binary Boolean net like the logical disjunction. This rule determines the transition frequency of the unit states. The logical disjunction generates in three cases a 1 as result and in one case a 0. In this case the value of the so-called ordering P-parameter is 3/4 or 0.75. The Boolean function of logical equivalence on the other hand generates in two cases a 1 and in two cases a 0 as results. In this case P = 0.5. It is possible to prognosticate the *principally* possible kind of dynamics such a system can generate: if the P-values are low, i.e., near 0.5, the system can generate rather complex dynamics with attractors of long periods; if the values of P are higher than 0.62, the system can generate only simple trajectories with point attractors or attractors with small periods. In this sense the different ordering parameters classify the according systems: they determine which kind of Wolfram class the system belongs to. We could demonstrate that in addition to the ordering parameters discussed by Kauffman the topology of complex systems, i.e. the rules that determine "who interacts with whom" contains at least another ordering parameter that we named the v-parameter - for the German word "Verknüpfung" = connection - [12].

If one tries to classify adaptive, that is, hybrid systems in the same way by introducing according ordering parameters then one has to look for *meta parameters,* i.e. parameters on the meta level of the whole rule system (meaning base rules and meta rules together). These meta-parameters determine the adaptive success of an adaptive system: in other words, specific values of meta-parameters decide whether an adaptive system reaches its targets fast or not, with a sufficient accuracy or not, and in a sufficient time or not. Meta-parameters describe the manner of changing the base rules in a quantitative way and are thus a feature of the meta-rules (for a systematic analysis of meta-parameters see [14]). They are in this sense the parallel of the mentioned ordering parameters of the rules sets of the base systems.

When distinguishing between base rules and meta-rules, an obvious criterion for the variability of an adaptive system is the ruthlessness (ρ) by which the meta-rules operate on the base rules. The ρ-parameter measures the quantity of changing a specific rule. In the case of SOCAIN the GA operates on the CA in the way that the values of the probability matrix are changed only rather softly, i.e. by raising or lowering them by a constant factor of approximately 0.05. Therefore, a probability parameter is changed as a consequence of a GA-operation for example from 0.3 to 0.35. The corresponding weight matrix of the IN is changed much more radically: A weight value in the matrix of the IN can be changed by one GA-operation from 0.3 to 0.9 or even from 0 to 0.9. In the case of the CA we obtain $\rho = 0.05$, in the case of the IN $\rho = 0.9$. When the CA gets its mobility matrix from the IN with the more radical changing, the CA gets much better results in optimizing itself to a specific environment than with its "own" matrix, i.e. the matrix which resulted from the more prudent changes.

Therefore, it seems that high ruthlessness gets better results, but things are not that simple: The hybrid IN with high ruthlessness is often not as good as the hybrid CA with low ρ, as was mentioned above. The best results are obtained when the hybrid

CA operates with low ρ, i.e. in a soft way, with the results it gets from the operations of the IN with high ρ. Therefore the best way seems to be to operate with *changing* ρ. The high ruthlessness of IN/GA is the way for CA/GA to get a fresh start when its own way of changing is too subtle. The difference of ρ in the two particular hybrid systems CA/GA and IN/GA explains the whole behavior of SOCAIN as well as the not very successful adaptive behavior of the IN.

In principle, these results are already known from evolutionary processes like, e.g., biological evolution. High and drastic mutation rates on the one hand mostly generate bad results because they frequently lead away from local optima the system has already obtained. This explains the behavior of the hybrid IN, because the high ρ-values are mathematically equivalent to high mutation rates. On the other hand, such high mutation rates just *because they force the system away form local suboptima* enable the system to try for new and better optima. This is what the hybrid CA does after having obtained new probability values from the IN. In this sense, the behavior of SOCAIN can be seen as a generalization of processes that are well known on more concrete physical levels.

5. Discussion

1. The GA of the CA mainly operates on the probability matrix and not, for example, on the frequency values of the transition rules according to the idea that a complex system can change the probability values of its rules more easily than the rules in general. If we take a look at political decisions, we can see that politicians first try to change the "probability" structure of the system, for example the possibility to choose a profession or to get a better education. Changing social rules more radically is a sensitive point, and it is the last possibility if the varying of the probability structure does not have the desired effect.
2. In a formal sense the system IN/GA can be interpreted as the result of the self-mapping of the system CA/GA. So the operations of IN/GA are to be understood as virtual experiments with the own future of the whole system - a sort of mathematical *gedankenexperiment*. That is why the ruthlessness of the operations of IN/GA can be much greater than in the case of CA/GA. Virtual experiments can be done with no consideration for the costs. If they are successful, the system can take over the results without having to pay for failures.
3. Sociologically speaking, social systems can achieve the ability of modeling themselves in rather different ways, as was demonstrated with the examples in section 2. Another possibility is a political one: The CA is a "bottom up" system, i.e. it can be understood as the model of a society where the social processes are local interactions between different social roles. The IN, on the other hand, is a "top down" system insofar as only macro aspects of the society - social classes - are taken into consideration. Thus a social system can model its bottom up processes by a top down model and improve its results; metaphorically speaking, the system is able to make politics "from above" through the process of self-modeling. This is the old dream of social planners and dictatorial utopians, which

often yielded disastrous results. The Killing Fields of Cambodia or the totalitarian states of the Soviet Union and National Socialistic Germany will not be the last examples. All these cases have got one thing in common: wishful thinking, based on some ideology, had designed a model of a "real" society and tried to carry it through by force. Neither means nor ends were clearly analyzed.

Our results hint at the possibility that democratic societies can use their own capability of self modeling in a "soft" way, namely by modeling a hypothetical future and discussing the advantages and disadvantages of the results. These modeling processes could be done in the way and by the use of computer systems like SOCAIN. Presently, however, this is rather science fiction than science.

4. SOCAIN was constructed for theoretical reasons, i.e. the modeling of certain social processes containing self-referentiality. But the applicability of SOCAIN is not restricted to theoretical problems like this. Consider for example a firm that is differentiated in several departments, containing a lot of employees. If this firm has to improve its economic success by changing its organization then a model of the SOCIAN-type can be immediately applied on the optimization problems of this firm. The same holds, e.g. for a military unit, differentiated into different subunits. According applications can be done with each system – political, organizational or even cognitive – where the elements of the base system can be aggregated in the way the elements of the CA of SOCIAN are aggregated t into social classes.

   Hofstadter and Dennett, as was mentioned above, describe the process of the modeling of self-referentiality as "a kind of miracle" that is probably one of the most promising ways to AI. Apparently it is indeed possible to model such complex processes without running into the traps of logical paradoxes. That is particularly important not only for social systems but also for the mind and brain as cognitive systems without doubt are able to perform processes of self-referentiality in most complex ways. Self-modeling is one aspect of self-referentiality; other types of cognitive self-referential processes like the conceptual formation of analogies and the mutual interdependent interplay of cognitive and social dynamics can be modeled in similar way (Klüver et al. 2004)

   We must leave it to other experts whether or not "true" AI is possible. But certainly such hybrid systems as SOCAIN help us to gain insight into complex processes that could be described without these modeling tools only in verbal and imprecise manner. Without doubt hybrid systems can be regarded as an important contribution to the science of formal model building.

References:

1. Berlekamp, E., Conway, J. H. and Guy, R.: Winning Ways for Your Mathematical Plays. New York: Academic Press (1982)
2. Bourdieu, P., Passeron, J.C.: Les Héritiers. Les Etudiants et la Culture. Paris: Editions de la Minuit (1964)
3. Chaitin, G.: the Unknowable. Singapore: Springer (1999)
4. Eder, K.: Die Entstehung staatlich organisierter Gesellschaften. Frankfurt: Suhrkamp (1976)
5. Geertz, C.: Deep Play: Notes on the Balinese Cockfight. In: Daedalus (1972) 101, 1–37.
6. Goonatilake, S., Khebbal, S. (eds.): Intelligent Hybrid Systems. Chichester-New York: John Wiley (1995)
7. Habermas, J.: Theorie des kommunikativen Handelns. Frankfurt: Suhrkamp (1981)
8. Hofstadter, D.R.: Gödel, Escher, Bach. An Eternal Golden Braid. New York: Basic Books (1985)
9. Hofstadter, D.R., Dennett, D.C.: The Mind. New York: Basic Books (1981)
10. Kauffman, S.: The Origins of Order. Oxford: Oxford University Press (1993)
11. Kauffman, S.: At Home in the Universe. Oxford: Oxford University Press (1995)
12. Klüver, J., Schmidt, J.: Control Parameters in Boolean Networks and Cellular Automata Revisited: From a Logical and a Sociological Point of View. In: Complexity (1999) 5/1, 45–52
13. Klüver, J.: Sociological Discourses in Virtual Reality. In: Social Science Computer Review (1996) 14,3, 280 –292.
14. Klüver, J.: The Dynamics and Evolution of Social Systems. New Foundations of a Mathematical Sociology. Dordrecht: Kluwer Academics Publishers (2000)
15. Klüver, J., Malecki, R., Stoica, C., Schmidt, J.: Sociocultural Evolution and Cognitive Ontogenesis: A Sociocultural-Cognitive Algorithm. In: Klüver, J. (ed.): On Sociocultural Evolution. Special Issue of CMOT – Computational and Mathematical Organization Theory (2004)
16. Niskanen, V.A.: Soft Computing Methods in Human Sciences. Heidelberg: Springer (2004)
17. Rasmussen, S., Knudsen, C., Feldberg, R.: Dynamics of Programmable Matter. In: Langton, C.G., Taylor, C., Farmer, J.D., Rasmussen S., (eds.): Artificial Life II. Reading (MA): Addison Wesley (1992)
18. Roth, G.: Das Gehirn und seine Wirklichkeit. Kognitive Neurobiologie und ihre philosophischen Konsequenzen. Frankfurt (M): Suhrkamp (1996)
19. Stoica, C.: Die Vernetzung sozialer Einheiten. Hybride Interaktive Neuronale Netzwerke in den Kommunikations- und Sozialwissenschaften. Wiesbaden: DUV (2000)
20. Stoica, C., Klüver, J.: Soft Computing. E-Course in the virtual curriculum Computer Science and Economy. Universities of Duisburg-Essen and Bamberg (2002)
21. Tarski, A.: Logics, Semantics, Metamathematics. Oxford: Oxford University Press (1956)
22. Wolfram, S.: Universality and Complexity in Cellular Automata. In: Wolfram, S.: Theory and Applications of Cellular Automata. Singapore: World Scientific (1986)
23. Wolfram, S.: A New Kind of Science. Champagne (Ill): Wolfram Media (2002)

Towards building Agent-Based Emergency Medical Services Systems

Basmah El Haddad

Computer Science Dept. Artificial intelligence, Humboldt Universität zu Berlin, Germany
Humboldt-University of Berlin, Department of Computer Science
Unter den Linden 6, 10099 Berlin, Germany[1]
elhaddad@informatik.hu-berlin.de

Abstract. The papers main aim is to proof that agents integrated in emergency medical services systems (EMSS) promise efficiency and effectiveness. It assures the advantage of using agent technology in EMSS during its different phases to support and assist personnel and decision-makers with flexible information flows and solutions of complicated distribution problems. First it introduces generally the EMSS, definition, nature, characteristics and phases. Then it provides a brief overview of agents and multi-agent systems showing their beneficial use to solve conflicts suggest solutions and provide information that improves the human rescue process. The paper includes a case study, describing the emergency medical control unit in Egypt, its structure, function and role. It presents a scenario in which agents will be integrated and introduces the agent-based EMSS architecture. A pre-hospital EMSS model will be explained and analyzed in details with its different contained agent types and tasks.

1 Introduction

The basic definition of agents, their properties and the nature of the emergency medical services system processes raises the importance of using agent-based systems in this domain. Agent's technology is a promising approach that supports the distributed work groups in the EMSS domain with the information that facilitates their communication and coordination needs. It plays an important role in assisting and supporting their work. The autonomous property of agents allows them to act in behalf of the user to lighten up his workload while he is practicing his activities and tasks. Through their social property they are able to communicate, cooperate and negotiate with human actors as well as other agents in their society. In the proposed agent-based EMSS model, the agents can proactively take the initiatives to maintain certain tasks. They introduce reliable, required information and data to the right actors at the right time. They also submit the decision-makers with proposed solutions and suggestions in a timely fashion way. Every minute people are born and others die. Life and death are the

[1] From October, Institute of National Planning , Salah Salem Nasr City, Cairo, Egypt

most natural phenomena in our lives. The problem arises when the society or the responsible organizations feel they could have done better or they didn't do their best to save a persons life. A lot of studies and reports announce that *traum*a is the leading cause of morbidity and mortality all over the world in patients less than 54 years. Trauma is defined as *an emotional shock causing a lasting harmful effect, as a result of unpleasant experience from wound or injury*. One of the most common causes of trauma is road traffic accident (RTA) [1]. Apart from the cause of trauma, injured patients require timely diagnosis and treatment by a multidisciplinary team of health care professionals, supported by the appropriate resources, to diminish or eliminate the risk of death or permanent disabilities. Death can be avoided by good planning and providing immediate medical attention. Actually most death that are prevented in the trauma cases are prevented in the first hour of the accident, which is also known as the *golden hour* [1, 2]. According to the journal of trauma; the probability of survival depends upon:

- The severity and type of injury
- Appropriate care at the scene
- The care shortly after admission *golden hour*
- Incidence of complications and quality of care at the intensive care unit (ICU)
- Pre-existing serious disease and the age factor (as an indirect reflection of cardiac reserve) [3]

From this point of view the main goals of the proposed agent-based EMSS model is to make use of the agent's properties to fulfill the following:

1. Gather and handle medical and organizational information and data fast, easy, secure and ensure information sharing and integration from various data sources.
2. Improve information flow between the involved parties by introducing reliable and actual information resulting in an enhancement of medical treatment
3. Easy up the communication and coordination between the different parties during the process
4. Allow negotiations of the responsible directions targeting an optimal solution (hospital, control unit, mobile units, police dep., fire dep. etc.)
5. Assure a detailed and reliable creation of various protocols; identification protocol at the emergency site, an emergency protocol during patient transportation for physicians in hospitals
6. Assist decision makers in making their different decisions introducing the needed information, data, protocols and suggested solutions

2 Emergency Medical Services System

From a medical point of view one can define an emergency as "*any circumstance that calls for an immediate action and in which the element of time in transporting the sick, wounded or injured for medical treatment is essential to the health or life of any person. Such circumstances include, but are not limited to, general accidents, traffic accidents and acts of violence resulting in personal injury, and sudden illness"* [3]. EMSS should have the ability to identify and modify illness and injury risks, provide acute illness and injury care and follow-up, and contribute to treatment of chronic

conditions and community health monitoring. EMSS differ from country to country according to the different policies, strategies, plans and various organizations responsibilities, coordination and management. It also relates to differences in geography, population distribution, medical resources, practice, history and local expectations. Although there are differences but all the systems agree in their basic roles, main tasks and the fundamental guidelines [1]. Experience, skills and medical knowledge showed that the provision of EMS in the field, prior to arrival at a hospital, could save lives. Medical care should be provided in a system in which all components providing services should be functioning in a well coordinated manner insuring continuity between phases in the course of the care of the emergency patient as well as the continuity and harmony during each phase. The EMSS is *"a coordinated arrangement of resources (including personnel, equipment, and facilities) which are organized to respond to medical emergencies, regardless of cause"* [4].

2.1 Nature and Characteristics of Emergency Medical Services System

EMSS processes have almost similar nature. They share characteristics that enable the integration of agents in their structure. Following are some of their defining properties

- *Distributed*: required data and information are spread among different requesters and units, distributed across various kinds of media (fax, data sheets, data bases, etc.) and introduced in different ontologies. During the patient transportation process, data and information will be accessed from; accident scene, control unit, mobile unit, and various hospitals. Emergency case record, patient history and patient electronic records should be exchanged and transferred if available.
- *Parallel*: many of the tasks contained at the EMSS processes run parallel. The pre-hospital phase includes parallel tasks; command a mobile unit to go to the patient, making communication and negotiation with appropriate hospitals, at the same time requesting new data and information about the emergency case from the caller etc.
- *Decentralized control*: the emergency control unit, paramedics at the mobile unit, clinical stuff, physicians at the EU, patients themselves and others affect the workflow of processes and information during the different phases of treatment resulting in a decentralized control, which requires good coordination of activities.
- *Communicative*: all the data, information, schedules, request etc. have to be communicated and exchanged easily during the treatment and processes [5]

2.2 Emergency Medical Services System Phases

EMSS provides delivering medical services at the different phases of the patient's care it implies that prevention of the death is possible at the accident spot by providing first aid treatment, transporting the patient to the nearest appropriate hospital and providing proper medical treatment. The injured or traumatically patient goes through different phases, in which he needs the most efficient medical care, skilled physicians and surgeons etc. According to the various definitions and phases of Emergency Medical Care Systems, it includes four fundamental phases, which are the heart of the system:

- Pre-hospital Care Phase
- Hospital & EU Phase
- Acute Care & Critical Phase
- Post-Hospital & Rehabilitation Phase

Eight major components operate together to support the activities and responsibilities of the EMSS during the different phases; personnel and training, communications, transportation, assessment of hospitals, emergency units and critical care centers, system organization and management, data collection, system evaluation and information management, education and support, disaster medical response [6]. The proposed agent-based EMSS model will improve medical care, health organization management and communicating of important data through each phase of the service.

3 Agents

Many researchers have been trying to find a general definition of the term "Agent", but all these definitions just light up some aspects of agents. One can tell that a software agent is a software entity that applies artificial intelligence techniques to choose the best set of actions to perform in order to reach a goal specified by the user. It should react in a flexible, proactive, dynamic, autonomous and intelligent way to the changes produced in its environment [7]. Another definition is IBM's definition "*Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires*". While the Wooldridge and Jennings definition for an agent "*… a hardware or (more usually) software-based computer system that enjoys some main properties of autonomy, social ability, reactivity and pro-activeness*" [8].

3.1 Multi-agent systems

A multi-agent system may be defined as "*a collection of autonomous agents that communicate between themselves to coordinate their activities in order to be able to solve collectively a problem that could not be tackled by any agent individually*" [7].
These group of agents can cooperate together to exchange information or knowledge in different and diverse social definitions and aspects of cooperation, coordination and competence. They also have the ability to negotiate on some common issue to fulfill their different tasks, functions and decision-making. Humans can easily be integrated in the multi-agent systems having their own role besides the software agents [9].

3.2 Agent-Based Emergency Medical Services System's Characteristics

Agent-based systems could fulfill the EMSS demands and requirements. They solve some of the problems facing the different actors during the various phases of their work in the EMS domain. The following Agent-based system characteristics illustrate the benefits of using agents and multi-agents in the EMSS domain:

- *Distributed problem solving mechanism*: the multi-agent system components can be running on different machines in different places, while each agent keeps the knowledge and information to solve his part of the problem. They offer a flexible way of solving distributed problems. In the pre-hospital phase there will be different agents distributed through various places. The *localcontrolunit-agent* and the *emergencycase-agent* at the local control unit, the *mobileunit-agent* and the *paramedic-agent* at the mobile unit and the physician-*agent* at the EU and others at the hospital. Each one has a certain role and some tasks to be done.
- *Sociability:* agents interact with each other (and possibly humans) via some kind of agent-communication language to obtain their goals. They can establish various messaging types and complex dialogues, in which they exchange information and data, negotiate and collaborate to coordinate actions, activities and solve problems. During the pre-hospital phase the *localcontrolunit-agent* communicates and exchanges data with the human operator as well as the *emergencycase-agent*. They sometimes negotiate with the hospital and EU agents.
- *Responsiveness and Reactivity*: agents perceive their environment, respond and act in a timely fashion to changes that occur at the perceived environment. They also take account to the changing environment and the changing user needs.
- *Pro-activeness:* agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative and make suggestions. They are able to perform tasks that maybe important for the user without to wait for his command. During the pre-hospital phase the *emergency case-agent* will record the emergency case and keep it actualized whenever there is any change. It communicates with different hospitals and negotiates for a certain place once it senses the new case.
- *Adaptivity:* it tailors the interactions reflecting the user needs and changes its behavior according to the previous experience.
- *Autonomy:* agents operate without the direct intervention of humans or others and have some control over their actions; they can make their own decisions based on their internal state and the environmental information they receive. During the pre-hospital phase the final control will be with human actors because of some aspects such as legacy problems, security factor and trust issues [9, 10].

4 Structure of the Emergency Medical Services System in Egypt

The EMSS in Egypt aims to provide an emergency medical top quality treatment and care for the Egyptian as well as the tourists. It performs a lot of services covering the whole country and is distributed along 27 governorates. The system is structured to contain a main central control unit in Cairo, which is connected to all the local control units at the different places, which are also connected with the EM centers, stations, tents at the cities or along the high ways. It's also connected to the hospitals and the ambulance mobile units through wireless networks. The main emergency medical control unit can communicate with decision-makers and authorities to manage in case of emergencies or disasters. The structure of the EMSS is shown in Fig. 1.

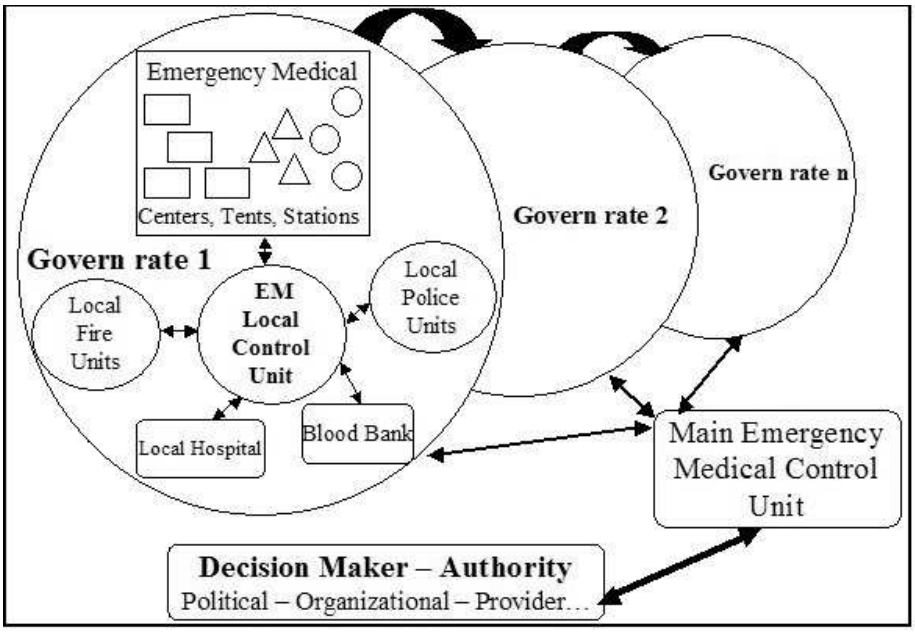


Fig. 1. Emergency Medical Services System Structure, Egypt

Case study: Cairo. The EM local control unit in Cairo is divided in two divisions; the first division is equipped with four phones connected to 30 lines to receive the emergency calls. These calls will be forward to the second division, which contains four other persons connected with the emergency mobile units, stations and centers through a wireless network. Each one is responsible for one of the four geographically quarters of the city (east, west, south and north). Their main task is to send the right emergency mobile units to the injured persons at the scene of the RTA or emergency as fast as possible trying to reduce the definitive treatment time by offering rapid medical care to the highest priority medical institute, which deals with that type of emergency. The EM local control unit structure in Cairo is shown in Fig. 2.

4.1 Scenario "What happens while Receiving an Emergency Call"

Upon receiving an emergency call at the emergency local control unit a certain scenario begins to take place. The emergency local control unit operator receives the emergency call and decides if it's a disaster or a normal emergency case. In case of a disaster he alarms the main control unit to begin the disaster management plans. In case of a normal emergency, the operator asks about the kind of emergency and the emergency place. He starts to record all the vital data and forward the case to the appropriate person at the second division. The operator keeps at the phone trying to take

information and data as much as possible. The responsible person at the second division decides the mobile unit type to be sent and orders one to go to the emergency place to transfer the injured patient to a certain hospital. He keeps contact to the mobile unit until it gets to the patient and then until it receives the hospital. He records all the important times beginning from the time of receiving the call, till ordering the mobile unit, till reaching the patient, till transferring the patient to the hospital. During this phase the local control unit operator keeps aware of all the new upcoming information or data concerning this case.

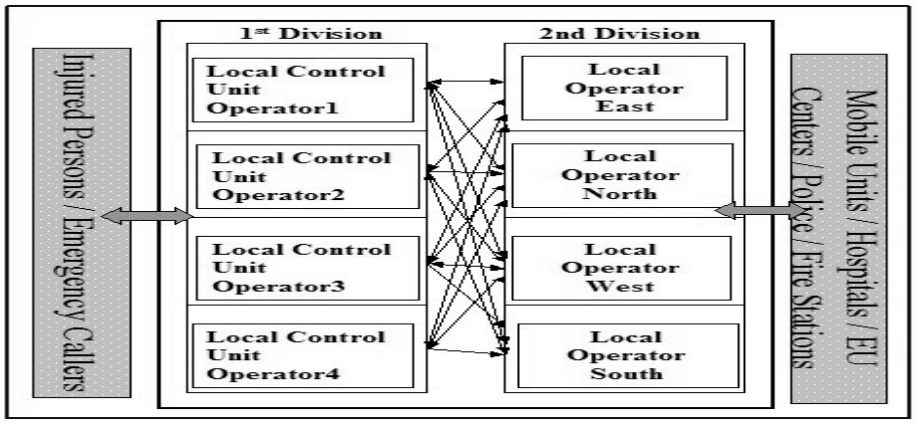


Fig. 2. Structure of the Emergency Medical Local Control Unit of the EMSS, Cairo

4.2 Emergency Medical Services System Local Control Unit Tasks and Activities

The main role of the emergency local control unit at the EMSS includes different tasks, activities, decisions and commands, which should be ordered and which are considered as their main responsibility. In the proposed model, agents will support these activities:

- Receiving the emergency call and checking its correctness.
- Recording and forwarding the data and required medical services needed to the appropriate coordinator according to it's geographically responsibility.
- Registering the case with all its available information and time of call.
- Deciding the number of mobile units which has to be sent in this case.
- Deciding the appropriate mobile unit type which is needed in this case.
- Deciding and contacting the nearest appropriate mobile unit to the scene place.
- Command the emergency mobile unit to the emergency place.
- Decide the hospital to which the case will be sent.
- Forward the initial information and data to the mobile unit and the hospital.
- Capture and keep aware of all new incoming information and make appropriate and continuous updating to the emergency record with recent information received.
- Following the emergency mobile unit path recording the arrival and departure time at the emergency place and at the hospital.

– Informing and cooperating with the authorities to manage a crisis or disaster.

4.3 Emergency Medical Services System Problems and Limitations

At this Scenario and others there are a lot of problems threatening the well-being of the patients, diminishing and weaken the functionality of the EMSS at its different phases. These happen almost because of the overload and stress of the human actors during this process. From this critical situation and point of view, one supposes that an agent-based system could be of great help to overcome the following problems and limitations:

– Failure of correct emergency case identification and detection.
– Wrong emergency case unification.
– Uncertain, unreliable and un-actual information and data.
– Misdiagnosis at the mobile unit and at the operation room.
– Absence of clear and plain roles and responsibility assignments.
– Wrong decisions and treatment due to the distributed nature of the problem.
– Failure of leading communication and negotiation processes between the involved parties (local control unit, mobile unit and hospital).
– Miscoordination, mismanagement of the emergency case at the local control unit.
– Lack of information integrity.

5 Agent-Based Emergency Medical Services System Process Model

The agent-based EMSS model is a model that presents human workflows during the EMSS phases, which are performed with the aid of software agents that support these workflows. It contains main processes that function in a synchronized way to supply the patient needs from the first moment he calls for help till arriving at the EU of a hospital and resuming by providing the specialists at the hospital with their different needs. Each process includes different sub processes that fulfil certain required tasks and activities. Within each process and sub process there are many roles, which could be human roles or agent roles. The agent-based model functions through the cooperation between human roles and software agents. They communicate, negotiate and exchange data and information with different types of software agents. The more coordination and organization between the different involved parties of human actors and software agents, the more efficiency and reliability will be gained in the real world system. The interaction, communication and coordination of software agents within the processes and sub processes will be realized through the messaging protocols. They communicate through exchanging different types of messages; -propose- accept-reject-retract-disagree… or counter propose - a course of action. The agents are not just communicating and exchanging some data and information but they can through these messaging techniques negotiate and handle about certain subjects and aspects. Fig.3 shows the main processes of the EMSS model.

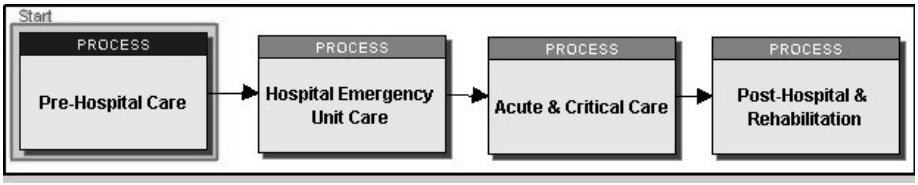


Fig. 3. Emergency Medical Services Main Processes

each of these processes contains their sub processes as follows:

1. Pre-hospital care process
   - Notification of an emergency call
     - Decide the mobile unit type
     - Search for the nearest mobile unit to the patient
     - Search for the most suitable hospital
   - Patient transportation to the hospital
2. Emergency unit care process at the hospital
   - First aid emergency medical treatment at the emergency unit
   - Scoring system
3. Acute care management at the hospital
   - Operation / Care management plan
4. Post-hospital & rehabilitation process

5.1 Agent-Based Emergency Medical Services System Architecture

The model follows the multi-layered Plaides collaborative agent-based architecture. It contains three layers of abstraction, for the task, interface and information specific collaborative agents. The interface agents are the interface between the users and the system, which enables the user to make their requests and analyse the results. The task agents will be used to get their instruction from the control or medical personnel to fulfil the user tasks by contacting and communicating other agents while the information agents in the information layer will collaborate with one another to provide the information to the requesting agents. They get their information from many different sources and databases. They support the information sharing between the control and medical team and can be used automatically for collecting various distributed medical or administrative data and information [11]. Fig. 4 shows the proposed agent-based EMSS architecture with its different contained agents.

5.2 Agent-Based Emergency Medical Services System Pre-hospital Process Model

The Model has been built using the AGIL (Agent- based Information Logistic) shell, which is a process-modelling tool that helps to model different processes and design intelligent agents within a good and easy user interface. During the communications between software agents and human actors or software agents within its society, they

make use of different sort of medium. A medium can be either analogue (e.g., a telephone call, a fax message, or a patient record) or an agent message, which can be defined as an ontology, which specifies the information content. All the roles, activities and different type of mediums are presented at the AGIL shell through specific process graph symbols.

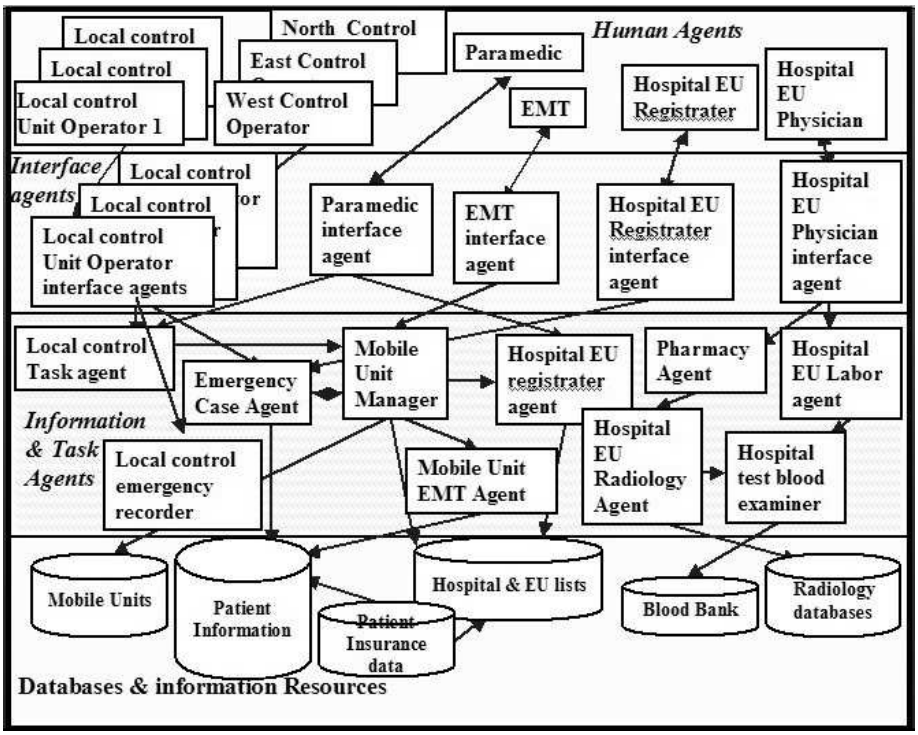


Fig. 4. Agent-Based Emergency Medical Services System Architecture (adapted from Pleiades system o Carnegie Mellon University (CUM) collaborative multi-agent system architecture) [11]

Notification of an Emergency Call Process Model At this process, see Fig. 5, one follows the scenario "What happens while Receiving an Emergency Call" mentioned before. The process has been analyzed to detect application scenarios of agents and to be optimized by introducing agents within it. The emergency call will be received at the local control unit as mentioned before. In case of a disaster the human local control unit operator sends an alarm message through its *local control interface agent*, which forwards this alarm through an acoustic medium to the main control unit manager. The manager receives this alarm and begins to invoke the disaster management plans and contacts other required parties according to the situation. In case of a normal emergency, the human local control unit operator through the user interaction invokes its *local control operator agent* to identify the caller and create the caller-id. The operator continues getting the vital important information from the caller and forwards it to its agent. The *local control operator agent* records the vital data and creates an

E_Case_Record. It also initiates an *emergency case agent*, which will follow this case till the end of its management. The main task of the *emergency case agent* is to actualize and update the E_Case_Record during the whole process and commands the other subprocesses. It also checks for duplications and solve conflicts. There are different conflicts for example like; receiving more than one call

- for the same case with different description and information
- for the same case with time lag
- for an escalated case invoking other cases
- for two different cases at the same place assuming there is just one case etc.

There will be also a *mobile unit advisor agent*, which has the task to decide which mobile unit has to be sent to the patient according to the specific cases. There are different mobile unit types; ordinary mobile unit, mobile unit with incubators for safe transportation of premature, mobile unit containing oxygen cylinder, mobile intensive care unit equipped with all means of advanced cardiac life support and advanced trauma life support, surgical mobile unit for surgical operation, air mobile units to reach touristic and rural places. Each case has its special requirements that have to be fulfilled. The *emergency management agent* commands to search for the nearest appropriate mobile unit to the patient and the most suitable hospital during different sub processes and according to specific constraints and actualized E_Case_Record.

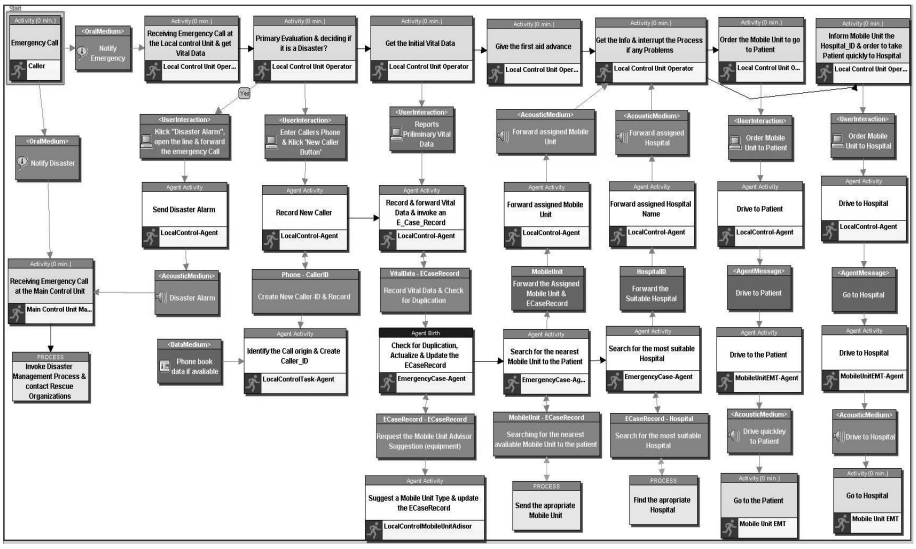


Fig. 5. Notification of an Emergency Call Process (this Fig is just to give a general overview)

Search for the nearest Mobile Unit to the Patient In this process the main task is to find the nearest mobile unit to the accident scene that can convey the patient. The *emergencycase-agent* invokes this search and forwards the assignment to the *mobileunitmanager-agent*, which broadcasts a request to all the mobile units of the free available mobile units list. It becomes *requests* or *refuses* from the different *mobileunitEMT-agents*. According to these answers it takes a decision, which unit will

be forwarded to the patient. This solution has to be suggested first to the human local control unit operator to give its final decision. The model will contain an option so that the *mobileunitmanager-agent* can make the final decision and order the mobile unit immediately to go to the patient, but for authentication reasons and legacy problems the final decision will be held by the human agent. The model suggests solutions and makes communications, to easy up the workload and provides explanations and results. In case that the *mobileunitmanager-agent* doesn't get an acceptance from the mobile units at all it begins to contact other *mobileunitmanager-agents* of other geographical sectors.

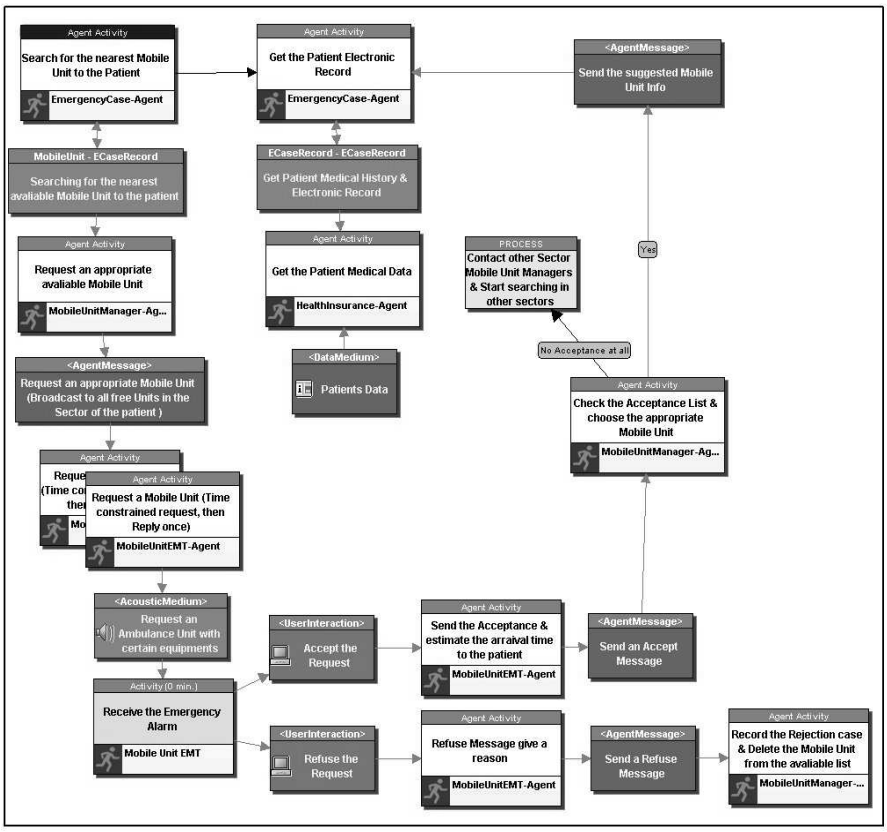


Fig. 6. Search for the nearest Mobile Unit to the Patient

Search for the most suitable Hospital At this process the most suitable hospital for this case will be searched. A broadcast will be send to the hospitalregistrator-agenst, which will search the hospital availability to receive this case. It should get the E-Case-Record to make the right decision according to the hospital facilities to deal with such cases or number of beds available. The hospitalmanager-agent will decide the most suitable and appropriate hospital from various aspects of availability or location or

specialization. Negotiations will be evolved at these processes as a lot of factors influence this process decisions. A geographical positioning system could be integrated in the model if it's available.

5.3 Design of the Emergency Medical Services System Pre-hospital Agents

The model will contain the following different agent types with their activities:

- *Interface Agents:* are personal assistance collaborating and communicating with the user to supply him with requests, information and commands. They represent each of the involved persons in the process, which means an interface agent for each of the different specialists during various phases of the system. Interface agents for control operators, mobile unit EMTs, mobile unit paramedics, hospital registrators, physicians, surgeons, lab doc. etc. All these agents supply their human actors with a proactive user interface to obtain their different needs and get all the data and information in a best way among it. This interface could be a normal computer monitor but it also can be a kind of hand held device that informs its owner with the latest and recent changes and variations (PDA) [12].

Table 1. Human Activities versus Interface Agents Activities

Human Agents		Interface Agents	
Local control Unit Operator	Receiving Emergency Call at the Local control Unit & get Vital Data, Primary Evaluation & deciding if it is a Disaster, Get the Initial Vital Data , Give the first aid advance, Get the Info & interrupt the Process if any Problems, Order the Mobile Unit to go to Patient, Inform Mobile Unit the Hospital_ID & order to take Patient quickly to Hospital	**Local Control Agent**	Record New Caller, Send Disaster Alarm, Record & Forward Vital Data & Invoke an E_Case_Record, Forward assigned Hospital, Forward assigned Mobile Unit, Command to drive to Patient, Command to drive to Hospital
Mobile Unit EMT	Receive the Emergency Alarm, Go to the Patient, Put the Patient in the Ambulance Unit, Connect the patient to the Monitor, Go to Hospital ⟷	**MobileUnit EMT-Agent**	Request a Mobile Unit (Time constrained request & Reply once), Send the Acceptance & estimate the arrival time to the patient, Refuse Message give a reason, Record the arrival time, drive to hospital
Mobile Unit Paramedic	Examine the patient, Initial Assessment of Patient status & Decide decease pattern & urgent needs, Stabilization, Begin with the needed 1st Aid services, Triage Process	**Mobile Paramedic-Agent**	Search for the appropriate hospital, Record the non stopping data permanently
Caller	Emergency Call	**Hospital A.**	Reply the Availability Percent,...
Main Control Unit Man.	Receiving Emergency Call at the Main Control Unit	**FireDep-A.**	
		PoliceDep.A	

– *Task Agents:* are controlled by the interface agent and processes tasks in behalf of their users. They fulfill information processing activities, solve problems and terminate when finished. The task agents perform different tasks; like initiating an E_Case_Record, deciding the appropriate mobile unit type to be sent to the patients,

solving nested organizational conflicts between the control unit, hospitals and the mobile units. At a later phase they can function as an operation scheduler, trauma-team consulter, blood examiner etc. Task agents have to know the model of the task domain as well as how to perform the task and gather the needed information. They collaborate together within the task layer to solve conflicts and get their request from the information specific agents [5, 12].

– *Information Agents:* collect and provide various data and information from their distributed data sources easily and fast. They respond to information requests and have the ability to cooperate with other agents when needed to provide the user with his needs [5]. The information-specific agents have to know how to access the databases and solve conflicts. Information strategies and protocols for coordination with relevant software agents are of great importance see Fig. 5.

In this model each agent has been designed to fulfill his activities and tasks. Table.1 shows the human activities versus the interface agent activities contained in the pre-hospital phase. Table..2 give an overview of task and information agent activities.

Table 2. Task and Information Agents Activities

EmergencyCase-Agent	Task Agent	Check for Duplication, Actualize & Update the ECaseRecord, Search for the nearest Mobile Unit to the Patient, Search for the most suitable Hospital, Get the Patient Electronic Record
MobileUnit Manager-Agent	Task Agent	Request an appropriate available Mobile Unit, Record the Rejection case & Delete the Mobile Unit from the available list, Check the Acceptance List & choose the appropriate Mobile Unit
LocalControl Task-Agent	Task Agent	Identify the Call origin & Create CallerID
LocalControl MobileUnit-Advisor	Task Agent	Suggest a Mobile Unit Type & update the EcaseRecord
HospitalManager-Agent	Task Agent	Search for the appropriate hospital
Health-Insurance Agent	Information Agent	Get the Patient Medical Data
HospitalInfo-Agent	Information Agent	Get the Hospital Information

Contribution and Further Work

Through this paper one can recognize the benefits and importance of using agent-based EMSS. Agents will be of great help and support for the involved persons to eliminate a lot of information problems. Beside the discussed benefits of building and implementing this model, it will face some problems; legacy, authority, privacy, social trust, special trust etc. This should not be an obstacle or hinder that prevents us to profit from using the agents technology. The main goal of going forward in this domain is our aim for successful patient outcomes. The proposed agent-based model will provide all

the requirements and demands of the EMSS and will overcome most of its limitations and problems. It insures the following:

- Care of the patient at each stage will be coordinated with services to be delivered at other stages in the response.
- These various services may be provided by different organizations. While it is important that each organization maintain operational autonomy to efficiently provide services, this autonomy must be balanced by the equally important need for multi-organizational cooperation in order to complete the "chain of survival" through all phases of the EMSS.
- Coordination of all aspects of the EMSS; system participants, facilitating the interdependent relationships, which are necessary for coordinated care services
- It assures a high quality of coordination, communication and negotiation.

The real system implementation with the various agents has just begun and is in its first phase. Object-Oriented Programming concepts will be used with Java language and Jade (Java Agent DEvelopment Framework) , which simplifies the implementation of multi-agent systems through a middle-ware that complies with FIPA specifications.

References

1. David, B., et al.: Trauma System Agenda For the Future: American Trauma Society Supported by the U.S. Department of Transportation, National Highway Traffic Safety Administration. (2002) .Website: www.nhtsa.dot.gov/people/injury/ems/TRAUMA_SYSTEM/index.htm
2. Peden, M., McGee, M., Shama, G.: The Injury Chart Book: A graphical overview of the global burden of injuries. World Health Organization, Geneva, (2000)
3. Fransis, L.: Benefits of Trauma Scoring Systems in the Emergency Unit at El Kasr El Ainy. Cairo University, (2000) in Arabic
4. Walker, J., et al.: Ambulance Services, city of Temple, (2003) chapter 5
5. Knublauch, H., et al., Towards a Multi-Agent System for Pro-active Information Management in Anesthesia: Fourth International Conference on Autonomous Agents (Agents 2000), Workshop on Autonomous Agents in Health Care, Barcelona, Spain (2000)
6. Nabors, M., Harris, M., Pletz B.: The Roles and Responsibilities of Local Emergency Medical Services Agencies within the California Emergency Medical Services System: A Position Paper by the Emergency Medical Services, Administrators Association of California, (1996)
7. Moreno A.: Medical Applications of Multi-Agent Systems. Rovira University, Spain, (2002)
8. Franklin, S., Graesser, A.: Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer- Verlag, (1996)
9. El Hadddad, B.: Towards building Multi-Agent Systems as Supporters in the Health Care Domain. CSP Workshop , Poland (2003)
10. Bradshow, J.: Software Agents, American Association for Artificial Intelligence, (1997)
11. Sycara, K., et al.: Distributed Intelligent Agents. Carnegie Mellon University, Pittsburgh, USA, (1996)
12. Knublauch, H., Rose, T.: Tool-Supported Process Analysis and Design for the Development of Multi-Agent systems. Bologna, Italy (2002) book chapter in: F. Giunchiglia, J. Odell, G. Weiss (Eds.), Agent-oriented software engineering III, Lecture Notes in Computer Science, Vol. 2585, Springer-Verlag (2003)
13. Jennings, N.R., Wooldridge, M.J.: Agent Technology Foundation, Application, and Markets. Berlin, Heidelberg, Springer-Verlag, (1988)

Object-oriented Model-based Extensions of Robot Control Languages

Armin Müller, Alexandra Kirsch, Michael Beetz

Informatik IX, Technische Universität München

Abstract. More than a decade after mobile robots arrived in many research labs it is still difficult to find plan-based autonomous robot controllers that perform, beyond doubt, better than they possibly could without applying AI methods. One of the main reason for this situation is abstraction. AI based control techniques typically abstract away from the mechanisms that generate the physical behavior and refuse the use of control structures that have proven to be necessary for producing flexible and reliable robot behavior. The consequence is: AI-based control mechanisms can neither explain and diagnose how a certain behavior resulted from a given plan nor can they revise the plans to improve its physical performance.
In our view, a substantial improvement on this situation is not possible without having a new generation of robot control languages. These languages must, on the one hand, be expressive enough for specifying and producing high performance robot behavior and, on the other hand, be transparent and explicit enough to enable execution time inference mechanisms to reason about, and manipulate these control programs. This paper reports on aspects of the design of RPL-II, which we propose as such a next generation control language. We describe the nuts and bolts of extending our existing language RPL to support explicit models of physical systems, and object-oriented modeling of control tasks and programs. We show the application of these concepts in the context of autonomous robot soccer.

1 Introduction

Robot control languages have an enormous impact on the performance of AI-based robot controllers. The languages allow for explicit and transparent representation of behavior specifications for reasoning and execution time program manipulation, and they provide the control structures for making the robot behavior flexible, reliable, and responsive. Despite their importance research on the design of robot control languages that enable intelligent robot control is largely neglected in AI — primarily for historical reasons.

As the predominant software architecture for autonomous robot control most researchers have used layered architectures, most notably the 3T architectures [15]. Characteristic for these layered software architectures is the use of multiple control languages: a programming language for the low-level reactive control and a very simple high-level language for strategic planning. This way the planner can still nurture the illusion of plans being sequences of plan steps and many existing planning techniques

carry over to robot control more or less the way they are. To bridge the gap between the partially ordered sets of actions (goal steps) and the low-level feedback control routines most software architecture use an intermediate control layer. In this intermediate layer an interpreter for a reactive plan language, such as RAP [7] or PRS [9], takes the high-level plan steps, selects methods for carrying them out based on sensor data, and executes them in a robust manner.

Unfortunately, this layered abstraction of robot control comes at high cost. The planning mechanisms cannot diagnose the behavior produced by a given plan because the behavior producing mechanism is much more sophisticated than assumed by the planning mechanisms. In addition, the planning mechanisms cannot exploit the variety of control structures offered by reactive plan languages to produce better behavior.

Let us illustrate this point using the following example taken from the autonomous robot soccer domain. A robot is to score a goal. An AI planner would typically produce a simple two step plan: (1) get the ball; (2) dribble it into the goal, because ball possession is a precondition for scoring. The navigation and the dribbling actions are considered as atomic black boxes. Unfortunately, these mechanisms do not allow for the generation of high performance plans with high scoring probability such as the one depicted in Figure 1(right). To compute such a high performance plan planning mechanisms have to tailor the parameterizations of the individual actions using accurate causal and physical models of the control routines they use.

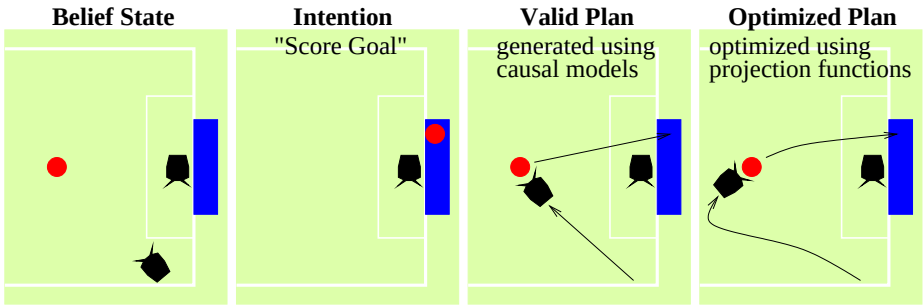


Fig. 1. Chaining of actions.

As plan-based robot control systems come of age and are applied to real world tasks a new generation of software architectures arises that are capable of dealing with these problems. These second generation software architectures share a number of important principles. These shared principles include (1) lightweight reasoning is embedded into the robot control languages; (2) the architectures invite programmers to specify models of the robot and its environment explicitly within the program code; (3) have much richer languages for the specification of goals in terms of constraints on the values of state variables.

In our research group we are currently working on the next generation of the robot control/plan language RPL [12] called RPL-II. RPL-II allows for the explicit specification and representation of robot learning problems [5], for the specification of explicit

robot and environment models, is object-oriented and supports the specification of specialization hierarchies for control tasks and routines and reasoning about them. RPL-II is an industrial strength robot control/plan language. It is implemented unlike its predecessor on a public domain CommonLisp using state of the art software tools such as Corba, UFFI, etc. RPL-II is applied to a variety of control tasks with different characteristics including mid-size robot soccer, a simulated household robot, and a robot assistant in an intelligent camera-equipped office environment.

This paper focuses on a particular aspect of the design of RPL-II, namely the representation and specification of the system that the controller controls and the control tasks that it performs. In a companion paper [5] we have described the new language features that support experience-based learning. The main contributions of the paper are the means for specifying state variables and their management, control tasks and control routines, and object oriented programming.

We demonstrate the application of these mechanisms to the implementation of our next generation controller for autonomous robot soccer. The implementation results in control programs that make extensive and explicit use of learning mechanisms and routines that can be much better reasoned about and transformed. This is primarily achieved through the explicit representation of physical entities and control tasks and the reasoning about them.

The remainder of the paper is organized as follows. In section 2 we describe the problems we encountered with our former control program for the AGILO soccer robots and sketch the ways we want to solve them. Section 3 introduces the basic concepts for model-based reasoning about physical control tasks. The use of these concepts is then demonstrated in the sections 4 and 5. We conclude with our next intended extensions of RPL-II, a discussion of related work, and our conclusions.

2 Languages at the Reactive Layer

The reasons why we want to use a model-based, object-oriented approach in structuring our control programs, are our experiences with former controllers of the AGILO soccer robots. [6] Let us therefore sketch how the controllers worked, which problems occurred and the conclusions we draw from them.

The control program ran in a low-level control loop that (1) updates the world model (section 2.1) and (2) chooses a command (section 2.2) in every time step (every 0.01 sec).

2.1 State representation

We need a set of variables containing information about the most likely current state of the world. We call this information “belief state”. The belief state is represented in a class “world model”, that provides variables and functions for any value the robot might want to know. It is not possible to differentiate between

- constant values (that would actually deserve the name “world model”),
- values taken from the percept or belief state vector, and

– values calculated from the belief state and world model.

So there is a representation of the state of the world, but with several drawbacks. The variables representing the state are not related to any physical values. In different parts of the program variables with different names, but the same physical meaning can occur. The orientation of the robot might in one place be called phi, in another place phi-deg. On the other hand does the same variable name not necessarily denote the same physical value. The variable pos-x can at one time express the robot's position, at another time the position of the ball.

Besides, the robot and the environment are not represented explicitly. When starting the robot in a certain environment the right configuration file has to be loaded and constant values are set. So we have just variables filled with values that have apparently nothing to do with the outside world.

Another problem is the heterogeneity of measuring units. This is especially hard when it comes to angles. Sometimes the value is given in degrees, sometimes in radian measure. In the action selector it is quite save to assume degrees, but there is no standard if degrees range from 0° to 360° or from -180° to $+180^\circ$.

2.2 Action Selection

Figure 2 shows a simplified extract of our old action selection routine. As is easily visible, there is only procedural knowledge in the controller. The information exchange between calling and called procedures is done by passing a variable param. The value(s) of this parameter sometimes denote the goal, sometimes a parameterization of the called function.

Furthermore the structure of the code seems very arbitrary. The purpose of the first three if-conditions is to trap failures. Then a more interesting part follows which decides what to do whenever the robot has the ball. Then again we have two failure conditions testing if the robot is in one of the penalty areas. Here we can see another problem. The reaction of being in the own penalty area or in that of the opponent team is almost the same and could be done by the same or a related function.

So in the end only three of eight cases build up the real controller code. The rest is only there for trapping failures. Even worse, the interesting parts are spread over the code and intercepted by failure testing. It is hopeless to reason about the best action, when there is no difference between failure trapping and real action selection.

Also the granularity of decisions seems ill-founded in our old controller code. When the robot has the ball, all we want to do is getting the ball somehow into the goal. Here the controller already decides how to do this (by dribbling the ball into the goal, kicking it or passing it to another player).

Another nuisance of our former controller is that it works in single time steps only and the decisions are purely reactive.

3 Key Concepts of RPL-II

After we have given a summary of the necessity of building a model-based, object-oriented system in section 2, we now have a closer look at the concepts we want to employ. These concepts are

```
function run-soccer-agent(worldmodel)
  var param := null
  var command
  if (not worldmodel.on-field) then command := NO-OP
  elseif (worldmodel.stuck-time > MIN-STUCK-TIME) then command := STUCK
  elseif (not worldmodel.localized) then command := RELOCALIZE
  elseif worldmodel.ball-is-in-guiderail then
    // do a lot of calculations and decide whether to call
    // PASS2POS, SHOOT2GOAL or DRIBBLE (with the goal as destination)
    param := ⟨x_dest,y_dest⟩
    command := DRIBBLE // for example
  elseif (worldmodel.time-in-opp-penalty-area > MAX-PA-TIME) then
    param := voronoi-pathplanning
    command := LEAVE-OPP-PENALTY-AREA
  elseif (worldmodel.time-in-own-penalty-area > MAX-PA-TIME) then
    param := voronoi-pathplanning
    command := LEAVE-OWN-PENALTY-AREA
  elseif (worldmodel.nearest-to-ball = my-robot-no) then
    param := voronoi-pathplanning
    command := GO2BALL
  else command := FACE-BALL
  execute(command, worldmodel, param)
```

Fig. 2. Code extract of our old AGILO controller.

1. state representation with globally accessible state variables
2. goal representation as constraints over state variables
3. control tasks and control routines arranged in an object hierarchy

We use the robot control language RPL (section 3.1), that provides constructs for monitoring failures while performing an action and parallel execution of processes. So now we think more in terms of actions than in terms of low-level commands and control loops.

For the representation of the robot's belief state we use globally known state variables that are described in more detail in section 3.2. Every variable corresponds to a physical value. We have not yet approached the representation of measuring units, although it should not be difficult within our framework.

The goal is now specified explicitly, tightly coupled to the state variables (section 3.3). We regard a goal as an intention how the world should be changed.

Finally, we require means of how to reach a given goal from a certain belief state. Our control procedures are structured along two lines, an object-oriented inheritance hierarchy and a calling hierarchy involving two classes of control procedures (section 3.4). We represent procedures as first-class objects, which allows for the specification of relevant calling parameters. Thus we can maintain a uniform calling mechanisms for all procedures. Inheritance is also an important factor when it comes to representing similarities between procedures. This makes the implementation very structured and concise.

To structure the calling hierarchy we introduce two classes of procedures, *control tasks* and *control routines*. A skill like "get-ball-into-goal" is implemented as a control

task. The different possibilities to fulfill the job like “dribble-ball-into-goal” or “kick-ball-into-goal” are represented as control routines. Control tasks and routines are called alternatively. The success and failure testing is completely done in the control task, as well as the choice of the appropriate routine in the current situation.

3.1 The Reactive Plan Language RPL

The robot’s plans are implemented in RPL (Reactive Plan Language) [12], which has been successfully employed in different projects [4, 3, 2]. RPL provides conditionals, loops, program variables, processes, and subroutines as well as high-level constructs (interrupts, monitors) for synchronizing parallel actions. To make plans reactive and robust, it incorporates sensing and monitoring actions, and reactions triggered by observed events.

Connecting Control Routines to “Sensors” Successful interaction with the environment requires robots to respond to events and asynchronously process sensor data and feedback arriving from the control processes. RPL provides *fluents*, registers or program variables that signal changes of their values. Fluents are used to store events, sensor reports and feedback generated by low-level control modules. Moreover, since fluents can be set by sensing processes, physical control routines or by assignment statements, they are also used to trigger and guard the execution of high-level control routines.

Fluents can also be combined into digital circuits that compute derived events or states such as the robot’s current distance to the ball. That fluent would be updated every time the position of the robot or the ball changes, since it is calculated out of the respective fluents.

Fluents are best understood in conjunction with the RPL statements that respond to changes of fluent values. The RPL statement **whenever** F B is an endless loop that executes B whenever the fluent F gets the value “true.” Besides **whenever**, **wait for**(F) is another control abstraction that makes use of fluents. It blocks a thread of control until F becomes true.

Behavior Composition sources use control structures for reacting to asynchronous events, coordinating concurrent control processes, and using feedback from control processes to make the behavior robust and efficient. RPL provides several control structures to specify the interactions between concurrent control processes (figure 3). The control structures differ in how they synchronize processes and how they deal with failures.

The **in parallel do**-construct runs a set of processes in parallel and fails if any of the processes fails. The second construct, **try in parallel**, can be used to run alternative methods in parallel. The compound statement succeeds if one of the processes succeeds. Upon success, the remaining processes are terminated. Similarly **try in order** executes the alternatives in the given order. It succeeds when one process terminates successfully, it fails when all alternatives fail. **with policy** P B means “execute the primary activity B such that the execution satisfies the policy P.” Policies are concurrent processes that run while the primary activity is active and interrupt the primary if necessary. Additional concepts for the synchronization of concurrent processes include semaphores and priorities.

in parallel do $p_1 ... p_n$	**in parallel do** navigate(⟨1.3, 2.0⟩) face-ball()
try in parallel $p_1 ... p_n$	**try in parallel** calculate-position-with-odometry() calculate-position-with-camera()
try in order $p_1 ... p_n$	**try in order** score-goal() distract-opponent()
with policy p b	**with policy** check-holding-ball() dribble(⟨4.2, 1.9⟩)

Fig. 3. Some RPL control structures and their usage.

3.2 State Representation

We represent the state of the world by globally declared variables. Figure 4 shows the class hierarchy of these variables.

Every value that is globally known throughout the system is called a *global value*. Every variable has a name and a (current) value. The world consists of values changing over time and values that remain constant. *Constants* are initialized when the system is started and represent the world model (section 4).

More interesting are *state variables*. Their value is represented as an RPL fluent, because it changes over time. Apart from being informed when a state variable has changed, it is often necessary to have access to former values. A *recording state variable* keeps a history of its past values. The history values can be accessed by the same function get-value that is used to obtain the current value of a state variable by specifying an additional parameter giving the number of steps we want to look back in time.

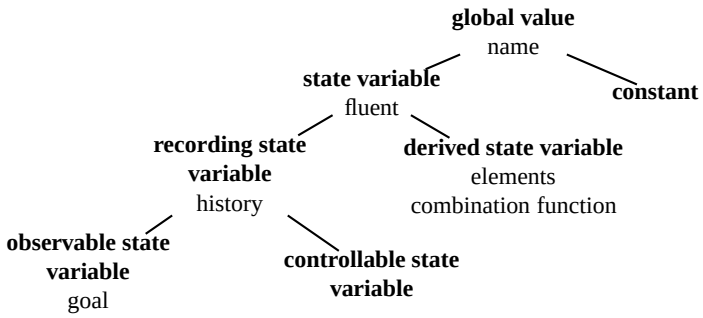


Fig. 4. Class hierarchy of globally known variables.

With these specifications we can now define *observable state variables* that represent our percept or belief state and *controllable state variables* representing the command. An observable state variable has an additional parameter for setting a goal value. This is explained in more detail in section 3.3.

The representation of the plain percept as state variables is usually not sufficient to make adequate decisions. For example we might want to react if a player is leaving the boundary of the soccer field. Or maybe we want to know if the robot has been stuck over a longer period of time. Therefore we introduced the concept of *derived state*

variables. From the outside, derived state variables are accessed just like observable state variables. But instead of keeping a history we remember the components and the function that produces the value of the derived state variable taking the component state variables as input. When a past value is accessed it is calculated from the history elements of the components.

As an example we have a look at the state variables in our soccer robots. The observable state variables include pos-x, pos-y and phi-deg which represent the x- and y-coordinates of the robot and its orientation as well as ball-x and ball-y denoting the position of the ball. The controllable state variables are c-rotation, c-translation, which set the robot's rotational and translational velocities, and kick, a boolean variable indicating whether to use the kicking device. Now we can define a derived state variable denoting the robot's distance to the ball:

make-instance derived-state-var
  name: distance-to-ball
  elementary fluents: *pos-x, pos-y, ball-x, ball-y*
  combination function: $\sqrt{(\textit{pos-x} - \textit{ball-x})^2 + (\textit{pos-y} - \textit{ball-y})^2}$

The fluent of the variable *distance-to-ball* depends on the state variables given in elementary fluents. The combination function calculates the current distance of the robot and the ball.

In order to test whether the robot is approaching the ball, we only need to check whether the distance to the ball has decreased:

fluent approaching-ball
  (get-value(distance-to-ball, t) $<$ get-value(distance-to-ball, t-1))

Since *distance-to-ball* is not a *recording state variable*, it does not have a history of its own. As the components and the function for obtaining the value are known and the components have a history, older values of *distance-to-ball* can be calculated.

3.3 Goals

Up to now our controller follows the very simple policy of our former control program described in section 2. In the future we would like to use a belief-desire-intention structure for the representation of top level goals or intentions (section 6). On the lower level we have to address the issue of how to tell a routine what to do.

There are two points of view for representing goals. First, we could order a routine to do something for us like "go to position $\langle$1.0, -1.5$\rangle$". So we have to pass a data structure that the routine must know how to interpret. The drawback of this idea is that there is no explicit relationship to the state variables *pos-x* and *pos-y*. The routine just knows that when these two state variables have the value of the goal specification the work is done.

Now the situation can also be seen as follows. Our control program wants to alter the world in a certain way, it might for example want to be in a state where the robot is at position $\langle$1.0, -1.5$\rangle$. It can now tell the corresponding state variables that it would like to have them changed to a different value. Then the controller calls a routine that is

best fit to produce the desired state from the current situation. The called routine looks up the goal values of the state variables and tries to reach them.

3.4 Procedures

To support an explicit representation of the agent program, we describe procedures as first class objects, so that we can reason about aspects such as performance measures or we can find out if a procedure has yet to be learned.

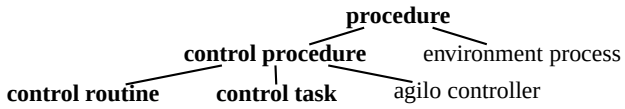


Fig. 5. Class hierarchy of procedures.

Figure 5 shows the basic class hierarchy of procedures. A *procedure* is any kind of function. At the moment, the for us most interesting subclass of *procedure* is a *control procedure*. A *control procedure* maps recent percepts to a command. Other kinds of procedures like environment processes, that map a command to a world state, might play a larger role in the future when we will model environment and perception processes.

For a good robot behavior we found it necessary to introduce two concepts of control procedures, *control tasks* and *control routines*. A robot should have certain skills, in the robot soccer domain we need skills such as dribbling or scoring a goal. These skills are called *control tasks*. Usually there are different ways to perform a skill. For example, in order to score a goal the robot might use its kicking device or dribble the ball into the goal. These implementations are called *control routines*. The job of the control task is to decide which control routine should be called in the current situation. This decision is based on models of the control routines that predict the time needed to fulfill a task or the probability of being successful.

Figure 6 shows a typical pattern of how control tasks and routines call each other. The task of scoring a goal can be achieved by two routines. One of them kicks the ball into the goal, the other one dribbles the ball to a point inside the goal. This second routine can be implemented like this:

```
control routine dribble-ball-into-goal
  p := find-goal-point()
  adjust-goal(pos-x ← p.x, pos-y ← p.y)
  execute(dribble)
```

We see that we need the task of dribbling in order to fulfill our goal. Therefore the control task *dribble* is executed, which can again be implemented by different control routines.

The “procedures” we are talking about are actually objects, the function that is really running is the generic function execute. With an object oriented approach we use inheritance mechanisms to get compact and concise implementations of the execute methods. This object oriented approach is especially useful in the domain of robot soccer where

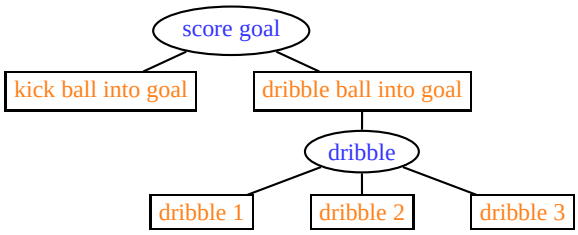


Fig. 6. Typical calling pattern of control tasks and control routines

almost every action comes down to navigation. So by using an object hierarchy we save a lot of work and redundancy.

Control Tasks Every skill is modeled as a *control task*. It should know when it has succeeded and when a failure has occurred and either react to it or abort execution. A special case of a failure is the exceeding of time resources. Since a control task's job is to choose the best control routine, it must know which control routines are available. All this information is given in the class definition:

```
class control task
  success
  failure
  time-out
  available routines
```

In most cases the control task will choose a control routine and check for failures or success during the execution. Such a method is shown in figure 7. What remains to do is the specification of the method choose-control-routine, which has to be implemented for each control task using models provided by the control routines.

```
method execute (ct of class control-task)
  r := choose-control-routine (ct)
  with-policy
    in parallel do
      whenever ct.failure fail("general failure")
      seq
        wait time ct.time-out
        fail("time-out")
    try in parallel
      execute(r)
      wait for ct.success
```

Fig. 7. Method execute for class *control task*

Control Routines are implementations of a *control tasks*. Since a control routine is always called by a control task, we don't have to worry about success or failure conditions. In both cases, the routine is interrupted by the control task. If there are errors the control task cannot detect, we can check them in the execute function of the control routine and return a fail command.

A control routine should not only reach a given goal state, it should also be able to predict how long it will take to reach this goal, what the accuracy of the solution will be or the probability of being successful at all. Thus, a control routine requires not only an execute method, but also methods providing information about its behavior.

4 Description of the Agent and the Environment

An intelligent robotic agent is more than just a program. It is a complex system that interacts with an environment through percepts and commands. We use this model to describe the agent, the environment and their interactions.

4.1 Declaration of the System Components

Fundamentally our system consists of two parts: an agent and an environment as described in [14]. These two components are absolutely independent, an agent can run in different environments and an environment can be the home of different agents. Therefore our first step is to state which agent should run in which environment. These declarations are principally used to declare state variables and constants (see section 3.2).

Figure 8 shows the parts we have to specify and how this information is used in global variables. Our *agent* consists of a *body*, an *architecture* and a *program*. The program is a control procedure that is called when the agent is given the command to run. The body describes physical properties of the agent like its dimensions. The architecture provides the connection to the environment, it describes which features the agent can receive as a percept and what kind of command can be given. The *environment* the agent acts in has certain properties that remain unchanged over time.

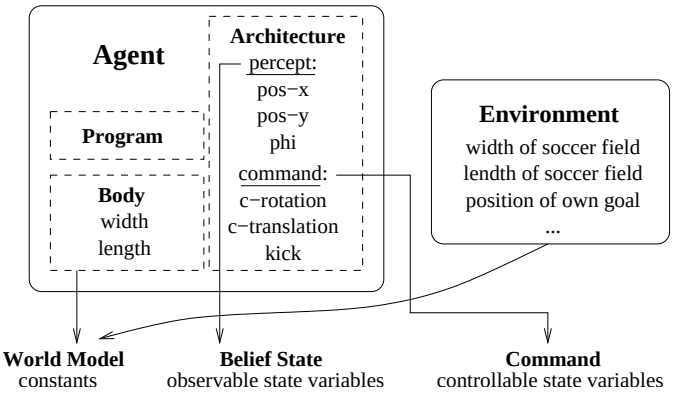


Fig. 8. Initialization of constants and state variables.

To summarize the information given by the agent and the environment we have three kinds of information, which is then provided by the global variables described in section 3.2: (1) a constant world model, (2) a belief state changing over time, and (3) a command.

4.2 Running the System

Now that we have declared an agent and an environment, we can run the agent. To do this, we call the function run-agent, which calls a method boot and starts a process update, both depending on the *agent-architecture* and the *environment*, and starts an RPL process that runs the *agent-program*. After initialization by the boot method agent and environment don't interact directly. The communication is done by the process update that gives the command in the controllable state variables to the environment and receives the percept, which it writes to the observable state variables. The state variables are set and read by a different process called RPL process which is running the agent program (figure 9).

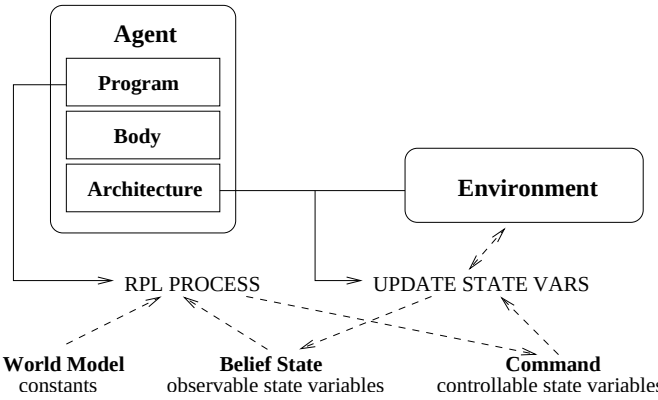


Fig. 9. The system at work.

When the agent has finished its job or when we want to stop the agent from the outside, the function kill-agent is called to stop the RPL and update processes and to call a method shutdown that specializes over the *architecture* and the *environment*.

5 The Agilo Controller

Using the concepts described in the previous sections we have implemented a very simple controller for our soccer robots. The controller consists of two processes: a monitoring and a controlling process. For this purpose the RPL construct with-policy can be used very effectively as shown in figure 10. The action selection is now concentrated in one loop, whereas the failure testing is done outside.

Of course, this is a very simple controller that has to be enhanced. We are planning to use a belief-desire-intention architecture to make sophisticated decisions (see also section 6).

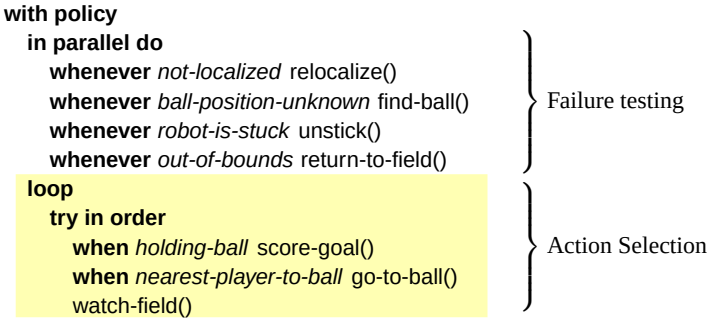


Fig. 10. Controller of our soccer robots

6 Research Agenda for RPL-II

We are developing RPL-II, the next generation of the robot control and plan language RPL. RPL-II supports the specification of high performance robot control programs by combining the expressiveness of RPL with respect to behavior specifications with advanced concepts that enable AI based control mechanisms to better reason about and manipulate control programs during their execution.

The extensions we have realized so far include the support of specifying explicit models of the physical systems to be controlled and object-oriented modeling of control tasks and routines. In two companion papers we have described extensions for the explicit representation of learning tasks in experience-based learning [5] and the tighter integration of programming and learning [10].

Still, these research results present only initial steps of the development of RPL-II, as a second generation AI-based robot control language. So let us briefly sketch the next steps on our research agenda: (1) comprehensive mechanisms for goal management, (2) improved physical system modeling, and (3) bootstrapping learning mechanisms for complex application tasks.

Comprehensive Goal Management. At the moment we only have the notion of low-level goals that are essentially constraints on state variables. So far RPL-II does not support goal selection that is consistent with the robot beliefs and other intentions of the robot. To support these goal management mechanisms we will add “desires” and “intentions” in addition to the current concept “goals” as first class objects in RPL-II and provide the respective reasoning mechanisms. This will give us the possibility of a much better action selection than the rule-based policy we are using now.

Deep models of state variables. While our current extensions make state variables explicit in the program code they still do not specify their physical meaning. We plan to provide such mechanisms by requiring programmers to specify the physical meaning in an explicit domain model formalized in a description logic. We believe that a concept taxonomy for a wide range can be provided and only small modifications for the individual application is needed. The slightly increased modeling effort will pay off immensely because using the domain model automated reasoning processes will be capable of solving much harder reasoning problems. For example, that all the conditions of a behavior trigger are observable or that two control routines will not interfere because the state variables they change are independent of each other.

Bootstrapping Learning Mechanisms. Finally, in RPL-II it will be possible to run partially specified control programs. The interpreter will then detect control tasks that the robot has no control routines for and acquire them by solving the associated learning tasks. This way a control program can complete itself or adapt itself to new environments and tasks by means of bootstrap learning.

7 Related Work

Model-based programming has been a major issue in several space exploration projects like Remote Agent [13], Livingstone [18], the Mission Data System Project [16], Reactive Model-based Programming Language [17] and others [1, 8, 11]. All of these projects represent the physical behavior of very complex systems in an explicit manner. This gives them the power to use lightweight reasoning techniques. The systems in these projects have to work very reliably. Vital parts of the physical systems are redundant, so that in the case of failure the system can be reconfigured. For this purpose the properties of the physical system parts have to be known by the controller. In our case reliability is not such an important issue. However, the environment our soccer robots have to deal with is much more dynamical. So we are interested in a robot that can adapt its control program to changing situations in its environment.

8 Conclusions

In this paper we introduce model-based concepts for the programming of robot controllers. The representation of state knowledge is done by state variables that are known throughout the system. Tightly coupled to the state variables is the representation of low-level goals. Those goals are achieved by control procedures arranged in two hierarchies: an object hierarchy that exploits inheritance mechanisms and a calling hierarchy including control tasks and control routines.

These concepts enable us to describe the robot and its environment declaratively. Using the robot control language RPL we can build a highly structured control program, where failure handling and action selection are separated.

On this basis we plan to include learning mechanisms as well as lightweight reasoning techniques. We still need to implement higher-level concepts like a belief-desire-intention architecture or logic representations to facilitate reasoning in the controller.

References

1. A. Barrett. Domain compilation for embedded real-time planning. In *Proceedings of the ICAPS'03 Workshop on Plan Execution*, 2003.
2. M. Beetz. Structured Reactive Controllers. *Journal of Autonomous Agents and Multi-Agent Systems. Special Issue: Best Papers of the International Conference on Autonomous Agents '99*, 4:25–55, March/June 2001.
3. M. Beetz. *Plan-based Control of Robotic Agents*, volume LNAI 2554 of *Lecture Notes in Artificial Intelligence*. Springer Publishers, 2002.

4. M. Beetz, T. Arbuckle, M. Bennewitz, W. Burgard, A. Cremers, D. Fox, H. Grosskreutz, D. Hähnel, and D. Schulz. Integrated plan-based control of autonomous service robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65, 2001.
5. M. Beetz, A. Kirsch, and A. Müller. Rpl-learn: Extending an autonomous robot control language to perform experience-based learning. In *3rd International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS)*, 2004.
6. M. Beetz, T. Schmitt, R. Hanek, S. Buck, F. Stulp, D. Schröter, and B. Radig. The AGILO robot soccer team - experience-based learning and probabilistic reasoning in autonomous robot control. *Autonomous Robots*, 2004. accepted for publication.
7. J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. Technical report 672, Yale University, Department of Computer Science, January 1989.
8. M. Ingham, R. Ragno, and B. C. Williams. A reactive model-based programming language for robotic space explorers. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, Montreal, Canada, 2001.
9. F. Ingrand, M. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, 7(6), 1992.
10. A. Kirsch, A. Müller, and M. Beetz. Programming robot controllers that learn. submitted to International Conference on Intelligent Robots and Systems (IROS), 2004.
11. R. Knight, S. Chien, and G. Rabideau. Extending the representational power of model-based systems using generalized timelines. In *The 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, Montreal, Canada, 2001.
12. D. McDermott. A Reactive Plan Language. Research Report YALEU/DCS/RR-864, Yale University, 1991.
13. N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence*, 103(1-2):5–48, August 1998.
14. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.
15. S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.
16. R. Volpe and S. Peters. Rover technology development and infusion for the 2009 mars science laboratory mission. In *Proceedings of 7th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS)*, 2003.
17. B. C. Williams, M. Ingham, S. H. Chung, and P. H. Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software*, 9(1):212–237, January 2003.
18. B. C. Williams and P. P. Nayak. Livingstone: Onboard model-based configuration and health management. In *Proceedings of AAAI-96*, 1996.

Meta-Reasoning in Multiple-Strategy Proof Planning

Andreas Meier Erica Melis

German Research Institute for Artificial Intelligence (DFKI)
Saarbrücken, Germany
ameier,melis@dfki.de

Abstract. Monitoring a solution process and applying the right action at the right moment are at the heart of intelligent problem solving by humans. This includes the analysis of failure events and the development of "recommendations" to overcome typical failures.
We present how meta-reasoning on failures can be used in multiple-strategy proof planning. This reasoning can exploit failures to guide subsequent proof plan manipulations and refinements. In the automated proof construction with the proof planner MULTI such failure reasoning cannot only ease the derivation of a solution proof plan but is required for some problems to find a solution at all.

1 Introduction

In a problem solving process, a step may not result in the expected progress or may not be applicable as expected. Hence, it is part of intelligent problem solving to analyze a failure event and to develop "recommendations" to handle typical failures, i.e., to guide the subsequent solution process. This also holds for mathematical theorem proving for which "*monitoring the state of a solution as it evolves and taking appropriate action in the light of new information*" is a key skill as Schoenfeld points out in his book on mathematical problem solving [13].

Monitoring the solution process and using "recommendations" requires a flexible control approach and reasoning about the problem solving situation. Intelligent humans do not rely upon pre-determined control to guide their problem solving. Instead, they draw upon a repertoire of heuristics for dynamic solution construction. As opposed to human problem solving, search-based theorem proving systems often employ restricted control components. Typically, only the local selection of the next step of the solution construction is subject of control reasoning. Other decisions are hard-coded into the system and monitoring and overseeing the entire problem-solving process is not possible.

In the multi-strategy proof planner MULTI the choice points that are subject to heuristic guidance are not restricted to local decisions, i.e., the choice of the next goal and the next method. Heuristics are also employed for the decisions for when and which steps to backtrack and how to deal with certain failures. They are explicitly represented in so-called control rules, which guide the meta-reasoning at several levels.

In extensive experiments we applied MULTI to several mathematical domains. The analysis of MULTI's proof attempts revealed typical failure situations as well as meta-reasoning patterns. It turned out that during the automated proof construction with MULTI such failure reasoning patterns do not only guide "clever" steps that ease the derivation of a solution proof plan but are often necessary to find a solution at all.

In this paper, we shall describe several meta-reasoning patterns that analyze and exploit failures to guide proof plan manipulations and refinements. We explain how the meta-reasoning patterns are realized in MULTI and exemplify their application to proof plan ϵ-δ-problems. Although we use ϵ-δ-problems as application domain all the described meta-reasoning patterns are applicable in other domains as well.

The paper is structured as follows. First, we introduce the basics of proof planning with multiple strategies. Afterwards, we describe failure reasoning patterns to guide the search in MULTI. Section 4 explains the application of MULTI to ϵ-δ-problems. Section 5 exemplifies the use of the failure reasoning patterns with ϵ-δ-problems and section 6 summarizes conducted experiments. The paper concludes with the discussion of the results and related work in section 7.

2 Proof Planning with Multiple Strategies

Proof planning [4] was originally conceived as an extension of tactical theorem proving to implement automated theorem proving at the abstract level of *methods*. Proof planning considers mathematical theorem proving as Artificial Intelligence (AI) planning problem: the initial state of a proof planning problem is specified by the proof assumptions, the goal is specified by the theorem to be proved, methods are the planning operators.

The knowledge-based proof planning developed in the ΩMEGA group [12] employs many AI principles and techniques such as hierarchical planning, knowledge representation in frames, use of constraint solvers, and meta-reasoning to guide the search. In particular, it emphasizes the integration of domain-specific knowledge into the planning process. Methods can encode not only general proving steps but also steps particular to a mathematical domain. Mathematically motivated heuristics guiding the search can be encoded in so-called *control rules*. The control rules are evaluated at choice points in the planning process and can express meta-level reasoning about the current proof planning state as well as about the entire history of the proof planning process and the proof context.

Proof construction may require to construct mathematical objects, i.e., to instantiate existentially quantified variables by witness terms. In proof planning, *meta-variables* are used as place holders for witness terms. When proof planning ϵ-δ-problems, equations and inequalities with meta-variables are passed to $\mathcal{CoSIE}$, a constraint solver for equations and inequalities over the reals. $\mathcal{CoSIE}$ checks the (in)consistency of the constraints and collects consistent constraints in a constraint store. Later, it tries to compute instantiations for the meta-variables that satisfy the collected constraints [14].

The simplest version of proof planning searches at the level of methods only, i.e., it searches for applicable methods and applies the instantiated methods, which are called actions, until all goals are closed. The final sequence of actions forms a solution plan. Operations such as backtracking and meta-variable instantiation are usually hard-coded: backtrack one action in the plan, if and only if no method is applicable and instantiate meta-variables only at the end, when all goals are closed.

Case-studies revealed that this somewhat inflexible proof planning fails for a number of problems [11]. This motivated the development of proof planning with multiple strategies, which decomposes the previously monolithic proof planning process and replaces it by separate *strategies*, which are instances of parameterized algorithms for different proof plan refinements and modifications.

We implemented proof planning with multiple strategies in the proof planner MULTI [11]. Among others, MULTI employs general algorithms for action introduction, meta-variable instantiation, and backtracking. The algorithm for action introduction has parameters for a set of methods and a set of control rules. When MULTI executes a strategy of this algorithm, then the algorithm introduces only actions that use the methods specified in the strategy. The choices during the action computation and selection are guided by the control rules specified by the strategy. The single parameter of the instantiation algorithm is a function that determines how the instantiation for a meta-variable is computed. If MULTI applies an instantiation strategy wrt. a meta-variable mv and if the computation function of the strategy yields a term t for mv, then the instantiation algorithm substitutes mv by t in the proof plan. The single parameter of the backtrack algorithm is a function that computes a set of refinement steps of other algorithms that have to be deleted. When MULTI applies a backtrack strategy, the algorithm removes all refinement steps that are computed by the function parameter of the strategy as well as all steps that depend from these steps. Sample strategies of all three algorithms are discussed in the subsequent sections.

In MULTI, no sequence of strategies is pre-defined or hard-coded in a control cycle. Rather, MULTI's blackboard architecture enables the flexible cooperation of independent strategies guided by meta-reasoning in *strategic control rules*. In a nutshell, MULTI operates according to the following cycle:

Job Offers Applicable strategies post their applicability in form of so-called job offers onto the blackboard.
Guidance Strategic control rules are evaluated to order the job offers.
Invocation The strategy with the highest ranked job offer is invoked.
Execution The algorithm of the invoked strategy is executed with respect to the parameter instantiation specified by the strategy.

Note that the execution of an action introduction strategy can be interrupted (i.e., interruption is a choice point in the action introduction algorithm). In this case, MULTI can first apply some other strategies and then re-invoke the interrupted strategy execution. Failures in the action introduction algorithm, i.e., a goal for which no method is applicable, are also interrupts. A detailed, technical description of the MULTI system can be found in [9].

3 Failure Reasoning

When searching for a solution proof plan, MULTI can encounter impasses, i.e., situations in which there is a goal, but there are no methods or strategies applicable to the goal. MULTI's standard approach to deal with such a failure is to backtrack the step that introduced this goal, i.e., to erase the problematic goal. This *goal-triggered backtracking* involves reasoning about the failure (i.e., for which goal no method is applicable?) and tries to tackle the cause of the failure instead of simply deleting the last introduced step (known as chronological backtracking).

For many situations, however, a different handling of a deadend is advisable. The reasons for this are twofold:
(1) Theorem proving often requires steps whose necessity is difficult to predict. Reasoning about a situation in which a failure occurred can suggest certain recovery or solution steps. Hence, the failures and their productive use can hold the key to discover a solution proof plan.
(2) Goals and applications of methods and strategies can be intertwined in complex ways. In particular, the incorporation of constraint solving into proof planning causes dependencies that make a 'standard' handling of failures difficult. Rather, dependencies have to be analyzed in order to guide suitable reactions.

In the following, we shall discuss several domain-independent and general meta-reasoning patterns on typical failures. The meta-reasoning patterns are declaratively encoded into corresponding control rules. This encoding and the concrete application to ϵ-δ-problems are discussed in section 5.

Guiding Case Splits

Case-split is a well-known technique in mathematics. But when is it useful to apply it and which cases should be considered? The following general pattern describes the need for a case-split: there is a main goal, which can be solved by methods introducing some side goals. These side goals are called conditions. If one of the conditions cannot be solved, then a partial success, i.e., the solution of the main goal, gives rise to consider patching the proof attempt by a case-split on the failing condition. Then, the main goal has to be proved for each case. This approach corresponds to the meta-reasoning pattern:

Case-Split Introduction:
IF failing condition while main goal is solved
THEN introduce case-split on failing condition

In the concrete application of this meta-reasoning pattern to ϵ-δ-problems, see section 5, we shall explain how the main goal and the side goals are determined in this domain.

Unblock Desirable Steps

Lets assume there is a typical combination of several steps to form a solution proof plan or part of a solution proof plan. Then, the application of certain key steps becomes "desirable" during the solution process according to this typical

combination. If such a desirable step should be applied but is blocked, then the application of other steps should be considered, which will unblock the desirable step. In the most general form, we can formulate this approach as the meta-reasoning pattern:

Unblock Desirable Steps:	
IF	particular step is desirable but blocked
THEN	perform other steps to enable this step

In section 5, we shall discuss the concrete application of two instances of this general pattern to ϵ-δ-problems. The two instances differ wrt. the determination of desirable steps as well as wrt. the selection of other steps to enable these desirable steps.

The first instance triggers the backtracking of particular steps to overcome blocked meta-variable instantiations of the constraint solver $\mathcal{CoSIE}$. This backtracking aims at the application of certain desirable steps and makes use of the freedom in MULTI to backtrack any actions in the proof plan under construction.

The second instance triggers the speculation of lemmas. Similar to the introduction of case-splits, in general, the speculation of lemmas is a Eureka step that potentially may introduce an infinite branching point into the search space that is difficult to control in automated theorem proving. The pattern instance speculates lemmas goal-directedly in order to enable a desirable but blocked method application. Then, the lemmas are subsequently proved.

Analysis of Meta-Variable Dependencies

The instantiations of meta-variables and constraints on the meta-variables cause dependencies among goals that share these meta-variables. Take, e.g., two goals G and G' that both contain a meta-variable mv. Now assume that MULTI first creates a partial proof plan for G and binds mv in such a way that G' cannot be proved anymore. The default reaction is the standard goal-triggered backtracking and would remove G'. However, the actual problem is not G' but the selection of an appropriate instantiation for mv. That is, part of the subplan for G has to be removed to introduce another subplan that instantiates mv differently. This approach corresponds to the general meta-reasoning pattern:

Analyze MV-Dependencies:	
IF	failure on goal caused by meta-variable instantiation/constraints
THEN	backtrack meta-variable instantiation/constraints

In the concrete application of this meta-reasoning pattern to ϵ-δ-problems, see section 5, we shall explain how the causal connection of a failure with the instantiation of a meta-variable or constraints on meta-variables is determined in this domain.

4 Proof Planning ϵ-δ-problems

We shall elaborate the usage of these meta-reasoning patterns for ϵ-δ-problems, which prove statements about the limit, the continuity, or the derivative of a

function f at a point a. The standard definitions of limit, continuity, and derivative comprise a dependency of a δ from an ϵ. For instance, the definition of limit and continuity are:

$$\lim_{x \to a} f = l \equiv \forall \epsilon_{\blacksquare}(0 < \epsilon \Rightarrow \exists \delta_{\blacksquare}(0 < \delta \wedge \forall x_{\blacksquare}(|x - a| > 0 \wedge |x - a| < \delta \Rightarrow |f(x) - l| < \epsilon)))$$

$$cont(f, a) \equiv \forall \epsilon_{\blacksquare}(0 < \epsilon \Rightarrow \exists \delta_{\blacksquare}(0 < \delta \wedge \forall x_{\blacksquare}(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon)))$$

An example theorem is the Cont-If-Deriv problem that states that, if a function f has a derivative f' at point a[1], then f is continuous at a. When the definitions of limit and continuity are expanded, then the problem's assumption is

$$\forall \epsilon_{1\blacksquare}(0 < \epsilon_1 \Rightarrow \exists \delta_{1\blacksquare}(0 < \delta_1 \wedge \forall x_{1\blacksquare}(|x_1 - a| < \delta_1 \wedge |x_1 - a| > 0 \Rightarrow |\tfrac{f(x_1) - f(a)}{x_1 - a} - f'| < \epsilon_1)))$$

and the problem's theorem is

$$\forall \epsilon_{\blacksquare}(0 < \epsilon \Rightarrow \exists \delta_{\blacksquare}(0 < \delta \wedge \forall x_{\blacksquare}(|x - a| < \delta \Rightarrow |f(x) - f(a)| < \epsilon))).$$

An ϵ-δ-proof of this problem as well as of similar theorems constructs a real number δ depending on ϵ that satisfies certain inequalities.[2] The usual procedure for discovering a suitable δ is the incremental restriction of the range of values. Proof planning adopts this approach by replacing unknown witness terms such as δ by meta-variables and by cooperating with the constraint solver $\mathcal{CoSIE}$, which collects constraints on the meta-variables.

In the remainder of this section, we describe the strategies employed by MULTI to accomplish ϵ-δ-proofs. A more detailed description of this application of MULTI is given in [9].

Strategies

Central for accomplishing ϵ-δ-proofs with MULTI is the action introduction strategy SolveInequality, see Table 1. It is applicable to goals whose formulas are inequalities. SolveInequality mainly comprises methods that deal with inequalities such as COMPLEXESTIMATE, TELLCS, ASKCS, FACTORIALESTIMATE, and SOLVE*-B. The control rule `prove-inequality` is in the list of control rules of SolveInequality.

When faced with an inequality goal, SolveInequality first tries to apply the methods TELLCS and ASKCS, which both interface $\mathcal{CoSIE}$. TELLCS passes the goal as constraint to $\mathcal{CoSIE}$ (provided it is consistent with the constraints collected by $\mathcal{CoSIE}$ so far), whereas ASKCS asks $\mathcal{CoSIE}$ whether the goal is

[1] That is, if $\lim_{x_1 \to a} \frac{f(x_1) - f(a)}{x_1 - a} = f'$.

[2] The construction of a δ is is a non-trivial task for students as well as for traditional, resolution-based automated theorem provers. Bledsoe proposed several versions of the problem Lim+ as a challenge problem for automated theorem proving [3]. The simplest versions of this problem (problem 1 and 2 in [3]) are at the edge of the capabilities of traditional automated theorem provers but the harder versions are beyond their capabilities. More difficult problems such as Cont-If-Deriv cannot be proved by traditional provers.

Strategy: SolveInequality		
Condition	*inequality-goal*	
Action	Algorithm	Action-Introduction
	Methods	ComplexEstimate, TellCS, Simplify, AskCS, Solve*-B, FactorialEstimate ...
	C-Rules	`prove-inequality`, ...

Table 1. The SolveInequality strategy.

entailed by its current constraints. If an inequality is too complex to be handled by $\mathcal{CoSIE}$, then SolveInequality tries to apply methods that reduce an inequality to simpler inequalities such as Simplify, Solve*-B, ComplexEstimate, and FactorialEstimate. For instance, applications of the ComplexEstimate method exploit the Triangle Inequality and reduce a goal with formula $|b| < e$ to simpler inequalities in case there is an assumption $|a| < e'$ and $b = k * a + l$ holds for suitable terms k and l. The resulting simpler goals are $|l| < \frac{e}{2}$, $e' < \frac{e}{2*mv}$, $|k| \leq mv$, and $0 < mv$, where mv is a new meta-variable. The method FactorialEstimate deals with fractions in inequalities. It reduces a goal of the form $|\frac{t}{t'}| < t''$ to the three subgoals $0 < mv_F$, $mv_F < |t'|$, and $|t| < t'' * mv_F$, where mv_F is a new meta-variable. Applications of Solve*-B exploit transitivity of $<, >, \leq, \geq$ and reduce a goal with formula $a_1 < b_1$ to a new goal with formula $b_2\sigma \leq b_1\sigma$ in case an assumption $a_2 < b_2$ exists and a_1, a_2 can be unified by the substitution σ.

In this way, SolveInequality successively produces simpler inequalities until it reaches inequalities that are accepted by $\mathcal{CoSIE}$. This approach – handle with $\mathcal{CoSIE}$ or simplify – is guided by the control rule `prove-inequality`. This rule first checks whether the current goal is an inequality. If this is the case, it prefers the methods of SolveInequality in the desired order: TellCS, AskCS, Simplify, Solve*-B, ComplexEstimate, FactorialEstimate etc.

To derive ϵ-δ-proofs Multi also employs the domain-independent action introduction strategies NormalizeGoal and UnwrapAss. Both strategies contain general methods for the decomposition of logic connectives and quantifiers. Whereas applications of NormalizeGoal decompose goals, applications of UnwrapAss decompose assumptions.

In order to instantiate meta-variables that occur in constraints collected by $\mathcal{CoSIE}$, Multi employs the two instantiation strategies InstIfDetermined and ComputeInstFromCS. The first is applicable only, if $\mathcal{CoSIE}$ states that a meta-variable is already determined by the constraints collected so far. Then, the computation function connects to $\mathcal{CoSIE}$ and receives this instantiation for the meta-variable. ComputeInstFromCS is applicable to all meta-variables for which constraints are stored in $\mathcal{CoSIE}$. The computation function of this strategy requests from $\mathcal{CoSIE}$ to compute an instantiation for a meta-variable that is consistent with all constraints collected so far.

Application and Cooperation of the Strategies

For proof planning an ϵ-δ-problem Multi typically proceeds as follows: First, it applies NormalizeGoal to decompose the initial goal. Afterwards, it applies

SolveInequality to the resulting inequality goals. Some methods of the strategy SolveInequality can only be applied when suitable assumptions are available (e.g., ComplexEstimate and Solve*-B). In case SolveInequality detects promising subformulas of assumptions, it interrupts (guided by one of its control rules) such that Multi can apply UnwrapAss to unwrap the promising subformula. Afterwards, SolveInequality can proceed and use the new assumption.

The invocation of ComputeInstFromCS is delayed by a strategic control rule until all goals are closed. This delay of the computation of instantiations for meta-variables is sensible since the instantiations should not be computed before all constraints are collected, i.e., only after all goals are closed. However, if the current constraints already determine a meta-variable, then a further delay of the corresponding instantiation is not necessary. Rather, immediate instantiations of determined meta-variables can simplify a problem [11]. To allow for the flexible instantiation of determined meta-variables SolveInequality can interrupt and cooperate with the strategy InstIfDetermined.

5 How Failure Reasoning Works (Examples)

There are default application and cooperation of strategies to accomplish ϵ-δ-proofs (see previous section). In addition, since Multi does not pre-define an order or combination of strategies, control rules can be added, which override the default behavior and implement failure reasoning patterns.

5.1 Guiding the Introduction of Case-Splits

The Cont-If-Deriv problem is an example for an ϵ-δ-problem that needs the introduction of a case-split. When tackling this problem, Multi starts as usual for ϵ-δ-proofs. It decomposes the theorem with the strategy NormalizeGoal and derives the inequality goals $0 < mv_\delta$ and $|f(c_x) - f(a)| < c_\epsilon$ (where mv_δ is a new meta-variable and c_x and c_ϵ are new constants) to which it applies SolveInequality. SolveInequality passes the first goal with an application of the method TellCS as constraint to $\mathcal{CoSIE}$ but fails to reduce the second goal. Since in the initial assumption it detects $|\frac{f(x_1)-f(a)}{x_1-a} - f'| < \epsilon_1$ as a subformula, which could be used, it interrupts. Multi applies the strategy UnwrapAss whose application yields the new assumption

$$|\frac{f(mv_{x_1})-f(a)}{mv_{x_1}-a} - f'| < mv_{\epsilon_1}$$

and the three new goals $0 < mv_{\epsilon_1}$, $|mv_{x_1} - a| < c_{\delta_1}$, and $|mv_{x_1} - a| > 0$ (where mv_{x_1} and mv_{ϵ_1} are new meta-variables and c_{δ_1} is a new constant).

With the new assumption, SolveInequality closes the main goal $|f(c_x) - f(a)| < c_\epsilon$ in several steps. In between, SolveInequality interrupts once and switches to InstIfDetermined, which introduces the binding $mv_{x_1} \rightarrow c_x$. Then, it tackles the new goals from the application of UnwrapAss. It succeeds to solve $0 < mv_{\epsilon_1}$ and $|mv_{x_1} - a| < c_{\delta_1}$ but fails to solve $|mv_{x_1} - a| > 0$, which meanwhile became $|c_x - a| > 0$ wrt. the introduced binding $mv_{x_1} \rightarrow c_x$. Thus, in this situation, Multi

can solve the main goal $|f(c_x) - f(a)| < c_\epsilon$ with an assumption that has some conditions. When MULTI uses the assumption, then it introduces the conditions as new goals. Later, it fails to prove one of these conditions, $|c_x - a| > 0$.

The meta-reasoning pattern *Case-Split Introduction* analyzes the failure and suggests its "repair". Technically, the pattern is realized in MULTI by two control rules, one strategic control rule and one control rule in SolveInequality, which guide suitable backtracking and the introduction of the case-split. This works as follows: if SolveInequality fails to prove a condition of an assumption that was used to prove the main goal, then the strategic control rule triggers the backtracking of all actions following the introduction of the failing condition.

In our example, the application of UnwrapAss and all actions that depend on it are backtracked such that $|f(c_x) - f(a)| < c_\epsilon$ becomes a goal again. When MULTI re-invokes SolveInequality after this backtracking, then the control rule in SolveInequality fires and suggests the application of the method CASESPLIT for the failing condition and its negation. Afterwards, SolveInequality has to prove $|f(c_x) - f(a)| < c_\epsilon$ twice: once under hypothesis $|c_x - a| > 0$ and once under hypothesis $\neg(|c_x - a| > 0)$. For the first case it proceeds as described above. The failing condition $|c_x - a| > 0$ now follows from the hypothesis of the case. The second case is solved differently by SolveInequality. First, it simplifies the hypothesis $\neg(|c_x - a| > 0)$ to $c_x = a$. Afterwards, it uses this equation to simplify the goal $|f(c_x) - f(a)| < c_\epsilon$ to $0 < c_\epsilon$, which follows from an introduced hypothesis.

Other ϵ-δ-problems also require this kind of failure reasoning (see section 6). In other mathematical domains the same pattern occurs and leads to a case-split introduction (see discussion of related work in section 7). Whereas the failure reasoning pattern is domain independent, the actual case-split may depend on the mathematical domain. So far, however, we employ a general case-split into the cases *cond* and $\neg$*cond* only.

5.2 Meta-Reasoning for Repair of Constraint Handling

The problem Lim-Div is an example problem for which backtracking is guided by meta-reasoning on a highly desirable but blocked strategy. To the knowledge of the authors this is a problem that has not been proved by any other system. It states that the limit of the function $\frac{1}{x}$ at point c is $\frac{1}{c}$:

$$\forall \epsilon_{\blacksquare}(0 < \epsilon \Rightarrow \exists \delta_{\blacksquare}(0 < \delta \wedge \forall x_{\blacksquare}(|x - c| < \delta \wedge |x - c| > 0 \Rightarrow |\tfrac{1}{x} - \tfrac{1}{c}| < \epsilon)))$$

The decomposition of the initial complex goal by NormalizeGoal results in the two goals $0 < mv_\delta$ and $|\frac{1}{c_x} - \frac{1}{c}| < c_\epsilon$ (where mv_δ is a new meta-variable and c_x and c_ϵ are new constants). SolveInequality closes the first goal by an application of TELLCS whereas it simplifies the second goal to $|\frac{c - c_x}{c_x * c}| < c_\epsilon$. An application of FACTORIALESTIMATE to this goal results in the three new goals $0 < mv_f$, $|c_x * c| > mv_f$, and $|c - c_x| < mv_f * c_\epsilon$ (with the new meta-variable mv_f). SolveInequality closes these three goals with TELLCS. Now all goals are closed and in the default behavior $\mathcal{CoSIE}$ is supposed to provide instantiations for the

meta-variables mv_δ and mv_f. That is, the strategy ComputeInstFromCS, which asks $\mathcal{CoSIE}$ to compute the instantiations, becomes a highly desirable strategy.

However, $\mathcal{CoSIE}$ fails to compute instantiations here and ComputeInstFromCS does not succeed. What is the problem? So far, $\mathcal{CoSIE}$ collected the constraints

$$\frac{|c_x - c|}{c_\epsilon} < mv_f,\ 0 < mv_f,\ mv_f < |c_x * c|,\ 0 < mv_\delta,\ 0 < c, \text{ and } 0 < c_\epsilon.$$

These constraints are consistent but a solution for mv_f exists only, if $\frac{|c_x - c|}{c_\epsilon} < |c_x * c|$ holds. This, however, does not follow from the collected constraints. In particular, the constraints collected so far are not sufficient for an ϵ-δ-proof since they do not establish a connection between c_ϵ and mv_δ. A possibility to overcome this problem is to refine the existing constraints in order to obtain an extended set of refined constraints for which a solution exists. That is, selected applications of TellCS (and only these selected applications) have to be backtracked in order to enable further refinement of some constraints.

An instance of the meta-reasoning pattern *Unblock Desirable Steps* analyzes the failure and suggests its "repair": *IF* the constraint solver fails to provide instantiations because of insufficient constraints, *THEN* backtrack to create and pass further constraints. Technically, the idea to overcome highly desirable but blocked meta-variable instantiations by the constraint solver is encoded in the strategic control rule `backtrack-to-unblock-cosie`. When all goals are closed, but the strategy ComputeInstFromCS is not applicable since the constraint solver fails to compute instantiations, then this control rule analyzes the constraints passed by applications of TellCS. It triggers the backtracking of actions of TellCS that pass inequalities to $\mathcal{CoSIE}$ that can be refined to simpler inequalities by applications of methods such as ComplexEstimate.[3] Then, these simpler inequality goals are may passed to the constraint solver.

In our example, `backtrack-to-unblock-cosie` triggers Multi to backtrack the application of TellCS that closes $|c - c_x| < mv_f * c_\epsilon$. Then, SolveInequality reduces the re-opened goal with the method ComplexEstimate. This action uses the assumption $|c_x - c| < mv_\delta$, which is created during the application of NormalizeGoal, and reduces $|c - c_x| < mv_f * c_\epsilon$ to the new goals $|0| < \frac{c_\epsilon * mv_f}{2}$, $mv_\delta \leq \frac{c_\epsilon * mv_f}{2 * mv}$, $|-1| \leq mv$, and $0 < mv$ (where mv is a new meta-variable). Afterwards, TellCS passes the new inequality goals to $\mathcal{CoSIE}$. Since $\mathcal{CoSIE}$ also fails on this extended constraint set `backtrack-to-unblock-cosie` guides the backtracking of the application of TellCS that closes $|c_x * c| > mv_f$. Again, SolveInequality reduces the re-opened goal with ComplexEstimate and passes the resulting inequalities to $\mathcal{CoSIE}$. This results in the following constraint store:

$c_\epsilon > 0$	$c > 0$	$mv_f \geq mv' * mv_\delta$	$mv' > c$
$mv_f > 0$	$mv > 1$	$\frac{c_\epsilon * mv_f}{2} > 0$	$mv_\delta > 0$
$mv_\delta \leq \frac{c_\epsilon * mv_f}{2 * mv}$	$mv_f * 2 \leq c^2$		

[3] Currently, the critical constraints are chosen by heuristics encoded in `backtrack-to-unblock-cosie`. It would be more convenient, if $\mathcal{CoSIE}$ would directly point out what the critical constraints are. However, this kind of information is not provided by the $\mathcal{CoSIE}$ system yet.

Now the following bindings consistent with these constraints can be computed: $mv \rightarrow 2$, $mv' \rightarrow c+1$, $mv_f \rightarrow \frac{c^2}{2}$, and $mv_\delta \rightarrow min(\frac{c_\epsilon * c^2}{8}, \frac{c^2}{2*(c+1)})$.

All ϵ-δ-problems in which subgoals with fractions occur need to repair the constraint reasoning (see section 6). In other domains the same meta-reasoning to overcome blocked instantiations of constraint solvers is applicable.

5.3 Lemma Speculation

An alternative that may unblock desirable steps is the speculation of lemmas. As example problem consider:

$$\lim_{x_1 \to c} f(x_1) = l \text{ follows from } \lim_{x \to 0} f(x+c) = l.$$

When the defined occurrences of limit are expanded, the problem consists of the assumption
$\forall \epsilon_{\blacksquare}(0 < \epsilon \Rightarrow \exists \delta_{\blacksquare}(0 < \delta \wedge \forall x_{\blacksquare}(|x-0| < \delta \wedge |x-0| > 0 \Rightarrow |f(x+c) - l| < \epsilon)))$
and the theorem is
$\forall \epsilon_{1\blacksquare}(0 < \epsilon_1 \Rightarrow \exists \delta_{1\blacksquare}(0 < \delta_1 \wedge \forall x_{1\blacksquare}(|x_1 - c| < \delta_1 \wedge |x_1 - c| > 0 \Rightarrow |f(x_1) - l| < \epsilon_1)))$.

The decomposition of the initial complex goal by NormalizeGoal results in the two goals $0 < mv_{\delta_1}$ and $|f(c_{x_1}) - l| < c_{\epsilon_1}$ (where mv_{δ_1} is a new meta-variable and c_{x_1} and c_{ϵ_1} are new constants). SolveInequality closes $0 < mv_{\delta_1}$ by an application of TellCS but fails to reduce the second goal with the current assumptions. Since in the initial assumption it detects $|f(x+c) - l| < \epsilon$ as a subformula, which could be used, it interrupts. Multi applies the strategy UnwrapAss whose application yields the new assumption $|f(mv_x + c) - l| < mv_\epsilon$ (where mv_x and mv_ϵ are new meta-variables) and some additional goals.

Next, SolveInequality should apply Solve*-B to tackle $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with the new assumption $|f(mv_x + c) - l| < mv_\epsilon$. However, this fails since the application of Solve*-B demands to unify $|f(mv_x + c) - l|$ and $|f(c_{x_1}) - l|$, which fails. Since no other method is applicable and there is no further promising subformula to unwrap, Multi would backtrack. The analysis that $|f(mv_x + c) - l|$ and $|f(c_{x_1}) - l|$ are quite similar and that the unification is blocked only because of the residue $mv_x + c = c_{x_1}$ gives rise to consider to patch the proof attempt by speculating the residue $mv_x + c = c_{x_1}$ as a lemma.

An instance of the meta-reasoning pattern *Unblock Desirable Steps* analyzes the failure and suggests its "repair": *IF* the application of a method is not possible because of a unification residue that is promising to be provable in the current context, *THEN* speculate the residue as lemma and use it for rewriting to enable the considered method applications.

The question is, when is a residue promising to be provable in the current context (otherwise residue speculation opens a Pandora's box)? In proofs that use a constraint solver (such as the ϵ-δ-proofs, which use $\mathcal{CoSIE}$) the constraint solver can be exploited to decide whether residues are promising lemmas. Whereas the employed unification and matching are decidable procedures that do not depend on domain-specific knowledge, $\mathcal{CoSIE}$ employs domain knowledge of inequalities

and equations over the field of real numbers. To exploit this domain knowledge as well as the context information passed to $\mathcal{CoSIE}$ so far we query $\mathcal{CoSIE}$ whether it accepts the residue before we speculate it as lemma. In this way, we combine the domain-independent unification and matching with the domain knowledge contained in $\mathcal{CoSIE}$.[4]

Technically, the described productive use of failing unifications and matchings for lemma speculation is encoded in the control rule `choose-equation-residue` in SolveInequality. `choose-equation-residue` analyzes the residue of blocked unifications and matchings and queries $\mathcal{CoSIE}$ whether it accepts the residue. If this is the case, `choose-equation-residue` fires and suggests to speculate the residue as lemma and to rewrite the current goal with this lemma.

In our example: When SolveInequality fails to tackle $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with the new assumption $|f(mv_x + c) - l| < mv_\epsilon$, then the analysis of the failure by `choose-equation-residue` yields the residue $mv_x + c = c_{x_1}$, which is accepted by $\mathcal{CoSIE}$. Hence, the control rule `choose-equation-residue` fires and introduces $mv_x + c = c_{x_1}$ as lemma and guides the rewriting of $|f(c_{x_1}) - l| < c_{\epsilon_1}$ with this equation. This results in the new goal $|f(mv_x + c) - l| < c_{\epsilon_1}$. The application of SOLVE*-B to this goal and the assumption $|f(mv_x + c) - l| < mv_\epsilon$ is now possible. It results in some simpler inequality goals, which SolveInequality passes to $\mathcal{CoSIE}$ by applications of TELLCS.

Other ϵ-δ-problems require this failure reasoning as well (see section 6). In other domains with constraint solvers or other means to decide for promising lemmas the same meta-reasoning to overcome blocked unifications and matchings is applicable.

5.4 Analyzing Meta-Variable Dependencies

As example for an ϵ-δ-problem that needs the analysis of meta-variable dependencies consider the following problem:

$$\lim_{x \to 0} f(a * x) = l \text{ follows from } \lim_{x_1 \to 0} f(x_1) = l \text{ and } a > 0.$$

Unfolding of the occurrences of limit and normalization result in the goal $|f(a * c_x) - l| < c_\epsilon$. Unwrapping the initial assumption yields the new assumption $|f(mv_{x_1}) - l| < mv_{\epsilon_1}$, which can be used to close the goal. Thereby, mv_{x_1} is instantiated by $a * c_x$. The unwrapping of the initial assumption also yields two goals, which become $|a * c_x| > 0$ and $|a * c_x| < c_{\delta_1}$ wrt. the instantiation $mv_{x_1} \mapsto a * c_x$. These two goals can be closed with two assumptions from the normalization of the initial theorem: $|c_x| > 0$ and $|c_x| < mv_\delta$. This works as follows: apply COMPLEXESTIMATE with the first assumption to the first goal

[4] An alternative to this combination is theory unification, which incorporates domain-specific equations into the unification procedures. However, the decidability of theory unification is difficult to determine and depends on the concrete set of domain equations (e.g., see [2]). We prefer decidable unification and matching procedure in order to avoid undecidable application conditions whose evaluation can block the complete proof planning process.

and pass the resulting inequality goals to $\mathcal{CoSIE}$ and apply ComplexEstimate with the second assumption to the second goal and pass the resulting inequality goals to $\mathcal{CoSIE}$.

Since the control rule `prove-inequality` suggests the method Solve*-B before the method ComplexEstimate, Multi does not find this solution directly. Rather, Multi applies Solve*-B to the first goal $|a * c_x| > 0$ wrt. the second assumption $|c_x| < mv_\delta$. This is possible since $|c_x| < mv_\delta$ equals $mv_\delta > |c_x|$ and mv_δ can be trivially unified with $|a * c_x|$. This results in the instantiation $mv_\delta \mapsto |a * c_x|$ and the new goal $|c_x| > 0$, which follows directly from the first assumption. Next, Multi tackles the second goal $|a * c_x| < c_{\delta_1}$ but fails, since with the introduced instantiation of mv_δ no solution is possible. However, not the second goal is problematic in the end, but the instantiation of mv_δ introduced during the solution of the first goal.

The meta-reasoning pattern *Analyze MV-Dependencies* analyzes the failure and suggests its "repair". Technically, the pattern is realized in Multi by the strategic control rule BackTrackLastBinding, which guides the backtracking of steps that introduce instantiations or constraints for meta-variables instead of the standard backtracking. In this case, the strategic control rule analyzes that the instantiation $mv_\delta \mapsto |a * c_x|$ is very unlikely to be part of a solution of an ϵ-δ-problem since the meta-variable for δ is supposed to be constrained during the proof planning process but not to be instantiated by different means than $\mathcal{CoSIE}$. Hence, the control rule guides the backtracking of the Solve*-B step that closed the first goal. As result, Multi has to tackle the first goal differently, which finally results in the solution of both goals sketched above.

Note that the control rule BackTrackLastBinding can also guide the successive trial and error of meta-variable instantiations. When Multi fails to solve a goal under a particular instantiation, then the instantiation of the meta-variable has to be backtracked (in order to try the next instantiation), rather than the goal for which Multi actually fails. The suitable backtracking is guided by BackTrackLastBinding, which overwrites the standard goal-triggered backtracking in this case. A domain, where BackTrackLastBinding is used for such a trial and error of meta-variable instantiations are residue class problems (see [10, 8]).

6 Experiments

All ϵ-δ-problems discussed in section 5 are taken from the analysis textbook [1]. We systematically applied Multi to solve the ϵ-δ-problems from the chapters 3, 4, 5, and 6 of this book. Currently, Multi can solve about 60 ϵ-δ-problems. Table 2 lists the problems from [1] whose solution requires the meta-reasoning discussed in this paper. Many similar problems could be formulated.

Since the knowledge engineering for proof planning is pretty difficult, the number of mathematical domains and problems successfully tackled by proof planned so far is growing only slowly. However, if not quantitatively then at least qualitatively, there is striking evidence for the need to meta-reason about failures in mathematics since the identified meta-reasoning patterns rely upon

Meta-Reasoning Pattern	ϵ-δ-problems
Case-Split Introduction	Theorem 4.3.3 (part 3), Theorem 6.1.2, Exercise 5.2.6
Unblock Desirable Steps	Theorem 3.2.3(b), Example 4.1.7(c)+(d), Exercise 4.1.3 (part 1), Exercise 4.1.10(a)–(d), Exercise 4.1.12(b), Theorem 4.2.4(b), Theorem 5.2.1(a), Theorem 6.1.3(a)–(c), Theorem 6.1.2
Analyze MV-Dependencies	Exercise 4.1.12(a) (+ residue class problems)

Table 2. Proof planning problems whose solution requires meta-reasoning about failures.

common techniques in mathematics. As evidence for this statement consider that failure reasoning in the proof planner CLAM (see related work in section 7) exploits similar failures in a completely different mathematical domain (proving theorems by mathematical induction) to guide similar proof plan modifications.

7 Conclusion and Related Work

We described three meta-reasoning patterns by which the multiple-strategy proof planner MULTI productively exploits failures to guide the subsequent proof planning process. They represent heuristics suggesting how to handle a failure that occurs in conjunction with a pattern of partially successful steps. The meta-reasoning patterns do not only circumvent failures, they hold the key to the construction of a solution proof plan.

The described failure reasoning and the repair modifications are possible since MULTI does not enforce a pre-defined systematic backtracking. Rather, when a failure occurs, then strategic control rules in which our heuristics are declaratively encoded can analyze the failure and can dynamically guide promising refinements and modifications of the proof plan. All the meta-reasoning patterns are generally applicable rather than over-specific as shown in the experiments (see section 6). Further meta-reasoning that exploits the flexible control in MULTI is discussed in [8].

Related Work

Related to the unblocking of desirable steps in MULTI is the control reasoning in elaborate blackboard systems, e.g., see [5] and [6]. When a highly desirable knowledge source is not applicable, then reasoning on the failure can suggest the invocation of knowledge sources that unblock the desired knowledge source.

Failure reasoning in the proof planner CLAM is closely related to the introduction of case-splits and lemmas in MULTI. In [7], Bundy and Ireland describe critics as a means to patch failed proof attempts in CLAM by exploiting information on failures. The motivation for the introduction of critics is similar to our motivation for failure reasoning: failures in the proof planning process often hold the key to discover a solution proof plan.

Critics in CLAM extend the hierarchy of inference rules, tactics, and methods. A critic is associated with one method – mostly with the wave method – and captures patchable exceptions to the application of this method. Critics are

expressed in terms of preconditions and patches. The preconditions analyze the reasons why the method has failed to apply. The patch suggests a change to the proof plan.

The situations that trigger case-split introduction and lemma speculation in CLAM and MULTI are very similar: unprovable premises of conditional facts from the context trigger case-split introduction, whereas missing premises in the current context trigger lemma speculation. However, the critics mechanism in CLAM and failure reasoning in MULTI considerably differ not only in minor technical issues but also in their conceptual design. Critics are a method-like entity directly bound to failing preconditions of a particular method. Moreover, part of a critic is a patch of the failure, which is a special procedure that changes the proof plan. In contrast, failure reasoning in MULTI is conducted by declarative and separate control rules. These control rules are not associated with a particular method but rather test for particular situations that can occur during the proof planning process (independent of the strategy or method that caused the situation). The control rules can reason about the current proof plan and about other information such as the history. The patch of a failure is not implemented into special procedures but is carried out by methods and strategies whose application is suggested by the control rules.

References

1. R.G. Bartle and D.R. Sherbert. *Introduction to Real Analysis*. John Wiley& Sons, New York, 1982.
2. K.H. Bläsius and H.J. Bürckert, editors. *Deduktionssysteme*. Oldenbourg, 1992.
3. W.W. Bledsoe. Challenge Problems in Elementary Analysis. *Journal of Automated Reasoning*, 6:341–359, 1990.
4. A. Bundy. The Use of Explicit Plans to Guide Inductive Proofs. In *Proceedings of CADE–9*, volume 310 of *LNCS*, pages 111–120. Springer, 1988.
5. D.D. Corkill, V.R. Lesser, and E. Hudlicka. Unifying Data-Directed and Goal-Directed Control. In *Proceedings of AAAI-82*, pages 143 – 147. AAAI Press, 1982.
6. E.H. Durfee and V.R. Lesser. Incremental Planning to Control a Blackboard-Based Problem Solver. In *Proceedings of AAAI-86*, pages 58 – 64. AAAI Press, 1986.
7. A. Ireland and A. Bundy. Productive Use of Failure in Inductive Proof. *Journal of Automated Reasoning*, 16(1-2):79–111, 1996.
8. A. Meier. MULTI – *Proof Planning with Multiple Strategies*. PhD thesis, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, 2004.
9. A. Meier. The Proof Planners of ΩMEGA: A Technical Description. Seki Report SR-04-XY, FR Informatik, Saarland University, Saarbrücken, Germany, 2004.
10. A. Meier, M. Pollet, and V. Sorge. Comparing Approaches to Explore the Domain of Residue Classes. *Journal of Symbolic Computation*, 34(4):287–306, 2002.
11. E. Melis and A. Meier. Proof Planning with Multiple Strategies. In *Proceedings CL-2000*, volume 1861 of *LNAI*, pages 644–659. Springer, 2000.
12. E. Melis and J. Siekmann. Knowledge-Based Proof Planning. *Artificial Intelligence*, 115(1):65–105, 1999.
13. A.H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, New York, 1985.
14. J. Zimmer and E. Melis. Constraint solving for proof planning. *Journal of Automated Reasoning*, 2004. accepted.

Nonlinear System Identification Using ANFIS Based on Emotional Learning

Mahdi Jalili-Kharaajoo

Young Researchers Club, Azad University, Tehran, Iran
mahdijalili@ece.ut.ac.ir

Abstract. Neural networks and Neurofuzzy models have been successfully used in nonlinear time series prediction. Several learning methods have been introduced to train the Neurofuzzy predictors, such as ANFIS, ASMOD and FUREGA. Many of these methods, constructed over Takagi Sugeno fuzzy inference system, are characterized by high generalization. However, they differ in computational complexity. A practical approach towards the prediction of real world data such as the sunspot number time series profound a method to follow multiple goals. For example predicting the peaks of sunspot numbers (maximum of solar activity) is more important due to its major effects on earth and satellites. In this paper, the emotional learning method, which has been used in control applications to provide multiple objectives, is proposed to train neurofuzzy predictors. The emotional learning based fuzzy inference system (ELFIS) has the advantage of low computational complexity in comparison with other multi-objective optimization methods. The efficiency of proposed predictor is shown in two examples of highly nonlinear time series. Appropriate emotional signal is composed for the prediction of solar activity and price of securities. It is observed that ELFIS performs better predictions in the important regions of solar maximum. It is also a fast and efficient algorithm to enhance the performance of ANFIS predictor in both examples.

1 Introduction

Predicting the future has been an interesting important problem in human mind. Alongside great achievements in this endeavor there remain many natural phenomena the successful predictions of which have so far eluded researchers. Some have been proven unpredictable due to the nature of their stochasticity. Others have been shown to be chaotic: with continuous and bounded frequency spectrum resembling white noise and sensitivity to initial conditions attested via positive Lyapunov exponents resulting in long term unpredictability of the time series. There are several developed methods to distinguish chaotic systems from the others, however model-free nonlinear predictors can be used in most cases without changes.

Comparing with the early days of using classical methods like polynomial approximators, neural networks have shown better performance, and even better are their successors: Neurofuzzy models [1], [2], [11], [19],[20]. Some remarkable algorithms have been proposed to train the neurofuzzy models [1], [7], [8], [10], [11], [14]. The pioneers, Takagi and Sugeno, presented an adaptive algorithm for their fuzzy inference system [14]. The other methods include adaptive B-spline modeling [8] and adaptive network-based FIS [7]. These learning methods fulfill the principle

of network parsimony which leads to high generalization performance in predictions or estimations. The parsimony principle says that the best models are those with the simplest acceptable structures and smallest number of adjustable parameters.
The next generation of neurofuzzy networks includes locally linear models which are piecewise linear approximations of the nonlinear function and can be interpreted as extensions of radial basis function networks [10], [11], [2]. In recent years RBF networks have shown good results in various fields of prediction and estimation; therefore, one can expect good results in predicting by locally linear neurofuzzy models. Of course these models have noticeable characteristics like low prediction errors, relatively low computational complexity and high generalization.
Following the directions of biologically motivated intelligent computing, the emotional learning method has been introduced as a kind of reinforcement learning. This approach is based on an emotional signal which shows the emotions of a critic about the overall performance of the system. The distinctive feature of emotional signal is that it can be produced by any combination of objectives or goals which improve the estimation or prediction. The loss function will be defined just as a function of emotional signal and the training algorithm will be simply designed to minimize this loss function. So the model will be trained to provide the desired performance in a holistic manner.
The emotional learning algorithm is a model-free method which has three distinctive properties in comparison with other neurofuzzy learning algorithms. For one thing, one can use very complicated definitions for emotional signal without increasing the computational complexity of algorithm or worrying about differentiability or renderability into recursive formulation problems. For another, the parameters can be adjusted in a simple intuitive way to obtain the best performance. Besides, the training is very fast and efficient. As can be seen these properties make the method preferable in real time applications like control tasks, as have been presented in literature [3], [4], [5], [9], [12], [18].
In this research the emotional learning algorithm has been used in predicting some real world time series: the sunspot number and the price of securities. The main advantage of introducing an emotional signal in prediction is adjusting some important features. For example, in predicting the sunspot number the peak points which are related to the maximum of eleven-year cycle of solar activity are more important than the others due to their strong effects on space weather, earth and satellites. Therefore, in order to achieve good predictions in these points the error and delay of predicting the peaks may be included in the definition of emotional signal. Additional achievements are fast training of model and low computational complexity.
This method can be used in various forms. A multi objective prediction can be done by introducing a simple definition of emotional signal at the first steps of training and exchanging it with a more accurate one to improve the accuracy in important regions. As an interpretation at the first stage of prediction the critic will just give attention to the overall performance of prediction and after some time, when an acceptable accuracy is obtained, it will incline to more important goals.
The main contribution of this paper is to provide accurate predictions using emotional learning algorithm in Takagi Sugeno neurofuzzy model. The results are compared

with other neural and neurofuzzy models like RBF network and ANFIS based on the prediction error in important regions and computational complexity.
The paper consists of five sections. The main aspects of Takagi-Sugeno fuzzy inference system along with associated learning methods are described in the second section. The third section deals with the various forms of utilizing emotional learning in the prediction problem. The results of applying the proposed prediction method to benchmark time series are reported and analyzed in section four. Finally, the last section includes some concluding remarks.

2 Neurofuzzy models

Two major approaches of trainable neurofuzzy models can be distinguished. The network based Takagi-Sugeno fuzzy inference system and the locally linear neurofuzzy model. The mathematical description of these models will be described in this section. It is easy to see that the locally linear model is equivalent to Takagi-Sugeno fuzzy model under certain conditions, and can be interpreted as an extension of normalized RBF network as well.
The Takagi-Sugeno fuzzy inference system is based on fuzzy rules of the following type

$$\begin{aligned} &Rule_i: \ If\ u_1 = A_{i1}\ And\ \ldots\ And\ u_p = A_{ip} \\ &then\ \hat{y} = f_i(u_1, u_2, \ldots, u_p) \end{aligned} \tag{1}$$

where $i = 1 \ldots M$ and M is the number of fuzzy rules. $u_1, \ldots, u_p$ are the inputs of network, each A_{ij} denotes the fuzzy set for input u_j in rule i and $f_i(.)$ is a crisp function which is defined as a linear combination of inputs in most applications

$$\hat{y} = \omega_{i0} + \omega_{i1}u_1 + \omega_{i2}u_2 + \ldots + \omega_{ip}u_p \tag{2}$$

Matrix form $\hat{y} = a^T(\underline{u}) \cdot W$
Thus, the output of this model can be calculated by

$$\hat{y} = \frac{\sum_{i=1}^{M} f_i(\underline{u})\mu_i(\underline{u})}{\sum_{i=1}^{M} \mu_i(\underline{u})} \quad ; \quad \mu_i(\underline{u}) = \prod_{j=1}^{p} \mu_{ij}(u_j) \tag{3}$$

where $\mu_{ij}(u_j)$ is the membership function of j^0 input in the i^{th} rule and $\mu_i(\underline{u})$ is the degree of validity of the i^{th} rule.
This system can be formulated in the basis function realization which leads to relation between Takagi-Sugeno fuzzy model and normalized RBF network. The basis function will be

$$\phi_i(\underline{u}) = \frac{\mu_i(\underline{u})}{\sum_{j=1}^{M} \mu_j(\underline{u})} \tag{4}$$

as a result

$$\sum_{j=1}^{M} \phi_j(\underline{u}) = 1 \tag{5}$$

This neurofuzzy model has two sets of adjustable parameters; first the antecedent parameters, which belong to the input membership functions such as centers and deviations of Gaussians; second the rule consequent parameters such as the linear weights of output in equation (2). It is more common to optimize only the rule consequent parameters. This can be simply done by linear techniques like least squares [1]. A linguistic interpretation to determine the antecedent parameters is usually adequate. However, one can opt to use a more powerful nonlinear optimization method to optimize all parameters together.
Gradient based learning algorithms can be used in the optimization of consequent linear parameters. Supervised learning is aimed to minimize the following loss function (mean square error of estimation):

$$J = \frac{1}{N} \sum_{i=1}^{N} (y(i) - \hat{y}(i))^2 \tag{6}$$

where N is the number of data samples.
According to the matrix form of (2) this loss function can be expanded in the quadratic form

$$J = W^T R W - 2W^T P + Y^T Y / N \tag{7}$$

Where $R = (1/N) A^T A$ is the autocorrelation matrix, A is the $N \times p$ solution matrix whose *ith* row is $a(\underline{u}(i))$ and $P = (1/N) A^T y$ is the p dimensional cross correlation vector.
From

$$\frac{\partial J}{\partial W} = 2RW - 2P = 0 \tag{8}$$

The following linear equations are obtained to minimize J:

$$RW = P \tag{9}$$

and W is simply defined by pseudo inverse calculation.
One of the simplest local nonlinear optimization techniques is the steepest descent. In this method the direction of changes in parameters will be opposite to the gradient of cost function

$$\Delta W(i) = -\frac{\partial J}{\partial W(i)} = 2P - 2RW(i) \tag{10}$$

and

$$W(i+1) = W(i) + \eta \cdot \Delta W(i) \tag{11}$$

where η is the learning rate.

Other nonlinear local optimization techniques can be used in this way, e.g. the conjugate gradient or Levenberg-Marquardt which are faster than steepest descent. All these methods have the possibility of getting stuck at local minima.

Some of the advanced learning algorithms that have been proposed for the optimization of parameters in Takagi-Sugeno fuzzy inference system include ASMOD (Adaptive spline modeling of observation data) [8], ANFIS (Adaptive network based fuzzy inference system) [7] and FUREGA (fuzzy rule extraction by genetic algorithm) [11].

ANFIS is one of the most popular algorithms that has been used for different purposes, such as system identification, control, prediction and signal processing. It is a hybrid learning method based on gradient descent and least square estimation.

The ASMOD algorithm is an additive constructive method based on *k*-d tree partitioning. It reduces the problems of derivative computation, because of the favorable properties of B-spline basis functions. Although ASMOD has a complicated procedure, it has advantages like high generalization and accurate estimation.

One of the most important problems in learning is the prevention of overfitness. It can be done by observing the error index of test data at each iteration. The learning algorithm will be terminated, when the error index of test data starts to increase.

3 Emotional learning

Most of new learning algorithms like reinforcement learning, Q-learning and the method of temporal differences are characterized by their fast computation and in some cases lower error in comparison with the classical learning methods. Fast training is a notable consideration in some control applications. However, in prediction applications, two more desired characteristics of a good predictor are accuracy and low computational complexity.

The Emotional learning method is a psychologically motivated algorithm which is developed to reduce the complexity of computations in prediction problems. Using the distinctive properties of this method one can follow multiple goals in prediction. In this method the reinforcement signal is replaced by an emotional cue, which can be interpreted as a cognitive assessment of the present state in light of goals and intentions. The main reason of using emotion in a prediction problem is to lower the prediction error in some regions or according to some features. For example predicting the sunspot number is more important in the peak points of the eleven-year cycle of sun activity, or predicting the price of securities with better approximation of variance may be desired. Of course these objectives can be pursued by other methods too, but by emotional learning any complicated multi objective problem can be solved using a fast, efficient computation.

This method is based on an emotional signal which shows the emotions of a critic about the overall performance of predictor. The distinctive feature of emotional signal

is that it can be produced by any combination of objectives or goals which improve estimation or prediction. The loss function will be defined just as a function of emotional signal and the training algorithm will be simply designed to decrease this loss function. So the predictor will be trained to provide the desired performance in a holistic manner. If the critic emphasizes on some regions or some properties, this can be observed in his emotions and simply affects the characteristics of predictor.
Thus the definition of emotional signal is absolutely problem dependent. It can be a function of error, rate of error change and many other features. Then a loss function is defined based on the emotional signal. A simple form can be

$$J = \frac{1}{2} K \sum_{i=1}^{N} es(i)^2 \tag{12}$$

where es is the emotional signal.
Learning is adjusting the weights of model by means of a nonlinear optimization method, e.g. the steepest descent or conjugate gradient.
With steepest descent method the weights will be adjusted by the following variations:

$$\Delta\omega = -\eta \frac{\partial J}{\partial \omega} \tag{13}$$

where η is the learning rate of the corresponding neurofuzzy controller and the right hand side can be calculated by chain rule:

$$\frac{\partial J}{\partial \omega} = \frac{\partial J}{\partial es} \cdot \frac{\partial es}{\partial y} \cdot \frac{\partial y}{\partial \omega} \tag{14}$$

According to (12): $\frac{\partial J}{\partial es} = K.es$

and $\frac{\partial y}{\partial \omega}$ is accessible from (3) where $f_i(.)$ is a linear function of weights.

Calculating the remaining part, $\frac{\partial es}{\partial y}$, is not straightforward in most cases. This is the price to be paid for the freedom to choose any desired emotional cue as well as not having to impose presuppose any predefined model. However, it can be approximated via simplifying assumptions. If, for example error is defined by

$$e = y_r - y \tag{15}$$

where y_r is the output to be estimated, then

$$\frac{\partial es}{\partial y} = -\frac{\partial es}{\partial e} \tag{16}$$

can be replaced by its sign (-1) in (14). The algorithm is after all, supposed to be satisficing rather than optimizing.
Finally the weights will be updated by the following formula:

$$\Delta\omega = -K \cdot \eta \cdot es \cdot \frac{\partial y}{\partial \omega} = -K \cdot \eta \cdot es \cdot \frac{\sum_{i=1}^{M} u_i \mu_i(\underline{u})}{\sum_{i=1}^{M} \mu_i(\underline{u})} \tag{17}$$

The definition of emotional signal and the gradient based optimization of the emotional learning algorithm in neurofuzzy predictors is clarified among two examples in next section.

4 Predicting the Sunspot number

Solar activity has major effects not only on satellites and space missions but also on communications and weather on earth. This activity level changes with a period of eleven years, called solar cycle. The solar cycle consists of an active part, the solar maximum, and a quiet part, solar minimum. During the solar maximum there are many sunspots, solar flares and coronal mass ejections. A useful measure of solar activity is the observed sunspot number. Sunspots are dark blemishes on the face of sun and last for several days. The SESC sunspot number is computed according to the Wolf sunspot number $R=k(10g+s)$, where g is the number of sunspot groups, s is the total number of spots in all the groups and k is a variable scaling factor that indicates the conditions of observation.
A variety of techniques have been used in the prediction of solar activity and its effects by sunspot number. The sunspot number shows low dimensional chaotic behavior and its prediction is a challenging problem for researchers. However, good results are obtained by methods proposed in several articles [6], [13], [15], [16], [17], [19], [20].
In this research, both the monthly and the yearly averaged sunspot number are used to predict. Fig.1 shows the history of solar cycles based on yearly sunspot numbers.

The error index in predicting the sunspot number in this article, the normalized mean square error (NMSE), is defined as follow

$$NMSE = \left(\frac{\sum_{i=1}^{n} (y - \hat{y})^2}{\sum_{i=1}^{n} (y - \bar{y})^2} \right) \tag{19}$$

In which $y, \hat{y}$ *and* $\bar{y}$ are observed data, predicted data and the average of observed data respectively.
At first the emotional learning algorithm has been used to enhance the performance and accuracy of a neurofuzzy predictor based on ANFIS. The emotional signal is computed by a linguistic fuzzy inference system with error and rate of error change as inputs. Figure 2 presents the target and predicted outputs of the test set (from 1920 to 2000). The lower diagram shows the results of best fitted data by ANFIS. The training is done with optimal number of fuzzy rules and epochs (74 epochs) and has been

continued until the error of test set had been started to increase. The other diagram shows the target and predicted values after using emotional learning. The emotional algorithm is just used in fine tuning of the weights of neurofuzzy model which has been initiated and adjusted by ANFIS. The error index, NMSE, has been decreased from 0.1429 to 0.0853 after using emotional learning. It's interesting that training ANFIS to the optimum performance takes approximately ten times more computation effort than the emotional learning to improve the prediction. Thus combining ANFIS with the emotional learning is a fast efficient method to improve the quality of predictions, at least in this example.

The next results are reported as a comparison of the quality of predicting the monthly sunspot number by emotional learning with other learning methods, especially the RBF network and ANFIS. All methods are used in their optimal performance. Over fitness is prevented by observing the mean square error of test data during training. Three regressors are used as the inputs of models. The specifications of methods, NMSE of predictions and computation times (on a 533 MHz Celeron processor) are presented in Table 1. Note that ELFIS is an abbreviation of the proposed predictor for Emotional Learning based Fuzzy Inference System.

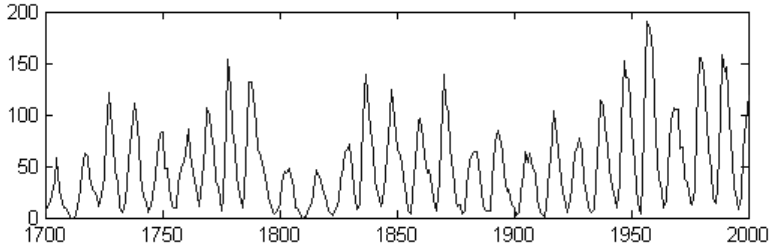


Fig. 1. The yearly averaged sunspot number.

Tab. 1. Comparison of predictions by selected neural and neurofuzzy models

	Specifications	Computation Time	NMSE
ANFIS	8 rules and 165 epochs	89.5790 sec.	0.1702
RBF	7 neurons in hidden layer	84.7820 sec.	0.1314
ELFIS	3 Sugeno type fuzzy rules	22.3320 sec.	0.1386

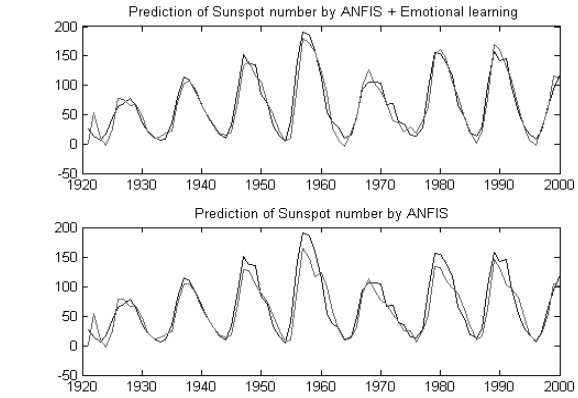


Fig. 2. Enhancement in the prediction of sunspot number by emotional learning, applied to ANFIS.

ELFIS is based on Takagi Sugeno fuzzy inference system, the emotional signal is computed by a fuzzy critic whose linguistic rules are defined by means of weighted error, rate of error change and the last targeted output. Thus the critic shows exaggerated emotions in the solar maximum regions, especially the peak points, where the prediction is more important due to its effects on earth and satellites. The weights of neurofuzzy model (equation 2) are adjusted by emotional learning via 17.
According to Table 1 learning in ELFIS is at least four times faster than the others and is more accurate than ANFIS. It is remarkable that using a functional description of emotional signal rather than the fuzzy description will generate faster algorithm, But finding such a suitable function is not easy.
Figures 3 to 5 show the predictions by RBF network, ANFIS and ELFIS respectively. These diagrams are a part of test set, especially the cycle 19 which has an above average peak in 1957. It's observable that ELFIS generates the most accurate prediction in the solar maximum; however the NMSE of RBF is the least. Noticeably it's more important to predict the peak points with small errors rather than the points in minimum regions. This is a result of the emotions of critic in the solar maximum.

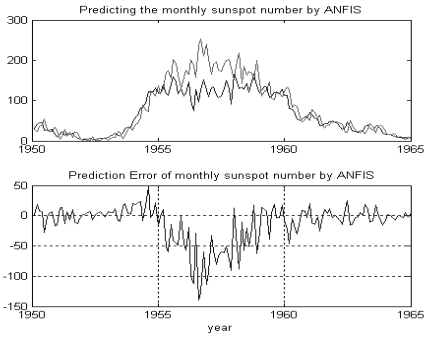


Fig. 3. Predicting the sunspot number by ANFIS

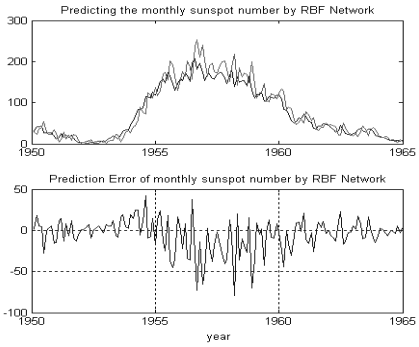


Fig. 4. Predicting the sunspot number by RBF Network

RBF network generates more accurate prediction through the test set; however it is observed that emotional learning provides better results in the solar maximum, especially at the above-average peak of 1957. Other test sets are chosen to stop the training in order to avoid over fitness. In this case even better NMSE can be obtained by RBF, in expense of higher prediction error especially in 1957.

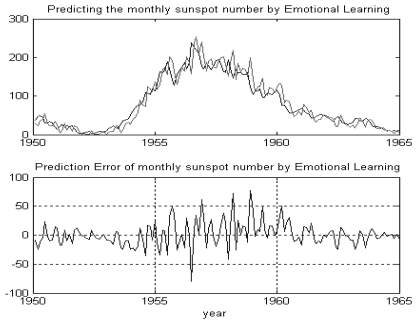


Fig. 5. Predicting the sunspot number by Emotional learning based fuzzy inference system

5 Conclusion

In this paper, the proposed emotional learning based fuzzy inference system (ELFIS) has been used as a predictor in two well known examples; In the prediction of solar activity (the sunspot number time series) the emotional signal is determined with emphasis on the solar maximum regions (the peak points of sunspot number) and it has shown better results in comparison with RBF network and ANFIS. In the prediction of security price, the emotional learning algorithm, defined by emotions of a fuzzy critic, results in good predictions. In fact the use of a combination of error and rate of error change leads to late overtraining of neurofuzzy model and thus more accurate predictions have been obtained. The definition of emotional signal is an important aid in emotional learning algorithms, which provides high degrees of freedom. In the problem of security price prediction, better performance can be obtained through the use of variables in addition to the lagged values of the process to be predicted (e.g. fundamentalist as well as chartist data).

Reference

1. Brown M., Harris C.J. (1994), *Neuro fuzzy adaptive modeling and control*, Prentice Hall, New York.
2. Eppler W., Beck H.N. (1999), "Peicewise linear networks (PLN) for function approximation," *Proc. of IEEE Int. Con. on neural networks*, Washington.
3. Fatourechi M., Lucas C., Khaki Sedigh A. (2001), "An Agent-based Approach to Multivariable Control," *Proc. of IASTED International Conference on Artificial Intelligence and Applications,* Marbella, Spain, pp. 376-381.
4. Fatourechi M., Lucas C., Khaki Sedigh A. (2001) "Reducing Control Effort by means of Emotional Learning," *Proceedings of 9th Iranian Conference on Electrical Engineering, (ICEE2001),* pp. 41-1 to 41-8, Tehran, Iran.
5. Fatourechi M., Lucas C., Khaki Sedigh A. (2001) "Reduction of Maximum Overshoot by means of Emotional Learning," *Proceedings of 6th Annual CSI Computer Conference*, Isfahan, Iran, pp. 460-467.
6. Izeman A. J. (1985), "Wolf J.R. and the Zurich sunspot relative numbers," *The Mathematical Intelligence*, Vol.7, No.1, pp. 27-33.
7. Jang J.R. (1993) "ANFIS: Adaptive network based fuzzy inference system," *IEEE Tran. On systems, Man and Cybernetics*, 23(3), pp. 665-685.
8. Kavli T. (1993), "ASMOD: An algorithm for adaptive spline modeling of observation data," *Int. J. of Control*, 58(4), pp. 947-967.
9. Lucas C., Jazbi S.A., Fatourechi M., Farshad M. (2000) "Cognitive Action Selection with Neurocontrollers," *Third Irano-Armenian Workshop on Neural Networks*, Yerevan, Armenia.
10. Nelles O. (1997) "Orthonormal basis functions for nonlinear system identification with local linear model trees (LOLIMOT)", *IFAC symposium for system identification (SYSID)*, Fukuda, Japan, pp. 667-672.
11. Nelles O. (2001), *Nonlinear system identification*, Springer Verlag, Berlin.
12. Perlovsky L.I. (1999), "Emotions, Learning and control," *proc. of IEEE Int. symp. On Intelligent control/Intelligent systems and semiotics,* Cambridge MA, pp. 132-137.
13. Schatten K.H., Pesnell W.D. (1993), "An early solar dynamo prediction: Cycle 23 ~ Cycle 22," *Geophysical research letters*, 20, pp. 2257-2278.

14. Takagi T., Sugeno M. (1985), "Fuzzy identification of systems and its applications to modeling and control", *IEEE Tran. On systems, Man and Cybernetics*, vol. 15, pp. 116-132.
15. Thompson R.J. (1993), "A technique for predicting the amplitude of the solar cycle", *Solar physics*, pp. 148, 383.
16. Tong H., Lim K. (1980), "Threshold Autoregressive limit cycles and cyclical data," *J. Roy. Statistics. Soc. B*, no.42, pp. 245-292.
17. H. Tong (1996), *Nonlinear time series: A dynamical system approach*, Oxford press, UK.
18. Ventura R., Pinto Ferreira C. (1999), "Emotion based control systems," *proc. of IEEE Int. symp. On Intelligent control/Intelligent systems and semiotics,* Cambridge MA, pp. 64-66.
19. Weigend A., Huberman B., Rumelhart D.E. (1990), "Predicting the future: a connectionist approach," *Int. J. Of Neural systems*, vol. 1, pp. 193-209.
20. Weigend A., Huberman B., Rumelhart D.E., (1992), "Predicting sunspots and exchange rates with connectionist networks," in *Nonlinear Modeling and Forecasting*, Casdagli, Eubank: Editors, Addison-Wesley, pp. 395-432.

Fault Diagnosis Features Extraction and Rules Acquisition Based on Variable Precision Rough Set Model

Qingmin Zhou [1, 4] Chenbo Yin [2, 3] Yongsheng Li [3] Thomas Rathgeber [2]

1. Institute of Computer Application in Planning and Design, Karlsruhe University, 76128 Karlsruhe, Germany
zhou@rpk.uni-karlsruhe.de
2. Institute of Mechanical Design and Automobile Engineering, Karlsruhe University, 76128 Karlsruhe, Germany
yin@mkl .uni-karlsruhe.de
3. College of Mechanical and Power Engineering, Nanjing University of Technology, 210009Nanjing, P.R. China
yinchenbo@njut.edu.cn
4. College of Information Science and Engineering, Nanjing University of Technology, 210009Nanjing, P.R. China
mse@njut.edu.cn

Abstract. Rough set theory is a new mathematical tool to deal with vagueness and uncertainty. But original rough sets theory generates only deterministic rules and deals with data sets in which there is no noise. This drawback has limited the applications of original rough set model for noisy or dirty data. The variable precision rough set model (VPRSM) was presented to handle uncertain and noisy information. Because the information of fault diagnosis is obtained directly from vibration signal system, there are often the existence of noisy data and uncertain information. Therefore, in this paper, a method based on VPRSM is proposed to apply to fault diagnosis feature extraction and rules acquisition for industrial applications. By selecting proper precision level b and using knowledge reduction, the redundancy in power spectrum data is reduced. The fault feature is effectively extracted and accurate diagnostic rule is acquired from the set of incomplete fault power spectrum samples by using the approach. An example for fault diagnosis of rotary machinery is given to show that the method is very effective.

1 Introduction

Rough set theory introduced by Zdzislaw Pawlak in 1982 has been described as a mathematical tool to deal with vagueness and uncertainty [1][2]. In rough set theory, the approximation region is determined through the indiscernible relations and classes. By the knowledge reduction, the classified knowledge rules are given. This approach seems to be of fundamental importance to artificial intelligence and cognitive sciences. Rough set based methods have been applied to machine learning,

knowledge acquisition, decision analysis, knowledge discovery from databases, expert systems, decision support systems, inductive reasoning, and pattern recognition etc.

However, the original rough set only generates deterministic rules and only deal with data sets in which there is no noise. But in practice the noisy data is inevitable. Therefore, a model development on the original rough set theory is needed. As an extension of original rough set model, the variable precision rough set model (VPRSM) is defined by W. Ziarko [3, 4]. This model inherits all basic mathematical properties of the original rough set model but allows for a predefined precision level b . This is an important extension, which will give us a new way to deal with the noisy data.

In industry application, a fault will cause economic losses and even human casualties. Generally, fault diagnosis can be treated as a pattern classification task. The traditional fault diagnosis based on models and residual analysis is not very effective if there are a few errors in the modeling of the system. Therefore, for a long time man has been looking for new efficient theories and intelligent methods to fault diagnosis. The rule-based diagnostic expert system is the most mature and the most promising [5]. However, many diagnostic rule-based expert systems suffer from the diagnosis inefficiency problem [6]. To guide the diagnosis, expert systems rely on an inference engine to derive the conclusions from the knowledge base. But the diagnosis time increases significantly when the number of queries grows, and the diagnostic process slows down because of the inefficient search of the knowledge base. At present, there are some intelligent diagnosis methods based on genetic algorithm, fuzzy sets theory, neural network and so on. However, in the reality these intelligent systems are hard to establish the mathematical models, in which the physics implication of each parameter can be not easily determined. An approach to fault diagnosis based on rough sets theory has be applied. But because the data information of fault diagnosis is obtained directly from vibration signal system, the noisy data is inevitable in fault diagnosis. If noisy data exists, the lower and the upper approximations cannot normally be formed. Therefore, the misjudgment of the diagnosis rule is often caused.

In this paper, a method based on VPRSM is proposed to apply to fault diagnosis feature extraction and rules acquisition for industrial applications. The damage extent of fault is evaluated by energy distribution of frequency. And the power spectrum data is smaller influenced by noise than the time series data. From this reason the power spectrum data is used as fault diagnosis signal. Typical fault of rotating machines were simulated in our rotor test-bed. For inconsistent data and noise data in power spectrum, the VPRSM allow a flexible region of lower approximations by precision variables. By selecting proper precision level b and using knowledge reduction, the fault feature is effectively extracted and accurate diagnostic rule is acquired from the set of fault power spectrum samples by this approach. An example for rotary machinery fault diagnosis is given to show that the method presented in the paper is very effective.

2 Variable Precision Rough Set Model

2.1 Information System

An information system is composed of a 4-tuple: $S=(U,A,V,f)$, in which $U=\{x_1,x_2,...,x_n\}$ is a finite non-empty universe; $A=\{a_1,a_2,...,a_n\}$ is a finite nonempty set of attributes; For each $a\in A$, V is value set of a; $f:U\times A\to V$ is the information function such that $f(x)\in V$ for every $a\in A$ and $x\in U$. The special case of information system is called decision table expressed in the form of relation table. In the decision table each row stands for a member of *U*, also known as a decision rule. Each column stands for the attribute and its value. $A=C\cup D$, $C\cap D=\phi$, *C* is set of condition attributes, *D* is set of decision attributes.

2.2 Approximation of Sets

In rough set theory, uncertainty is defined by the lower approximation and upper approximation. In information system $S=(U,A,V,f)$, each non-empty subset $B\subseteq A$ determines an indiscernibility relation as follows:

$$R_B=\{(x,y)\in U\times U: a(x)=a(y)\forall a\in B\} \quad (1)$$

$$U/R_B=\{[x]_B: x\in U\} \quad (2)$$

R_B partitions U into a family of disjoint subsets U/R_B. U/R_B denotes all the equivalence classes of U. For $x\in U$, its equivalence class is denoted by

$$[x]_B=\{y\in U:(x,y)\in R_B\} \quad (3)$$

For an arbitrary subset $X\subseteq U$, we can associate two subsets with *X*:

$$\underline{R}_B(X)=\cup\{[x]_B:[x]_B\subseteq X\} \quad (4)$$

$$\overline{R}_B(X)=\cup\{[x]_B:[x]_B\cap X\neq\phi\} \quad (5)$$

$\underline{R}_B(X)$ and $\overline{R}_B(X)$ are respectively called the R_B lower and R_B upper approximation of *X*. R_B boundary region of *X* is defined as:

$$BNR_B(X)=\overline{R}_B(X)-\underline{R}_B(X) \quad (6)$$

Meanwhile, $posR_B(X) = \underline{R}_B(X)$ is defined as R_B positive region of *X*, $negR_B(X) = U - \overline{R}_B(X)$ as R_B - negative region of *X*.

If $\overline{R}_B(X) = \underline{R}_B(X)$, then *X* is R_B -definable sets. If $\overline{R}_B(X) \neq \underline{R}_B(X)$, *X* is R_B -rough sets. In this case, for a given concept *X*, we can only know that *X* contains at least all elements in $\underline{R}_B(X)$ and does not contain any element outside $\overline{R}_B(X)$. The boundary region contains those results that are possible, but not certain.

2.3 Rough Approximation Analysis

The above-mentioned approximation sets of original rough set model are quite sensitive to noisy data. For instance, a decision table $S = (U, C \mathbf{U} D)$, *E1* and *E2* are two equivalence classes of condition attributes *C*; *Q* is an equivalence class of decision attributes *D*. There are 100 elements in *E1*, *E2* respectively. For equivalent class *Q*, only an element in *E1* belongs to *Q* , and only an element in *E2* does not belong to *Q* . Based on approximation sets of original rough set model, *E1* and *E2* belong to same boundary region of *Q*. That is to say, it is difficult to make a judgment for two results. However, in *E2* only an element that does not belong to *X* can be caused by noise. Thus in the above-mentioned approximation sets, the date classification of 99% consistent and 1% inconsistent can not be distinguished.

To solve this problem, the VPRSM was proposed. The VPRSM was aimed at handling uncertain and noisy information and was directly derived from the original rough set model without any additional assumptions. The VPRSM extends the original one by relaxing its strict definition of the approximation boundary using a predefined precision level b . Hence some boundary regions are included in the positive region.

2.4 Variable Precision Rough Set Model

The precision level b denotes the proportion of correct classifications, in this case the domain of b is $0.5 < b \leq 1$. For a given information system $S = (U, A, V, f)$, $X \subseteq U$, $B \subseteq A$, lower approximation, upper approximation and boundary region are defined with precision level b .

$$\underline{R}_B^b(X) = \mathbf{U}\{[x]_B : P(X/[x]_B) \geq b\} \tag{7}$$

$$\overline{R}_B^b(X) = \mathbf{U}\{[x]_B : P(X/[x]_B) > 1 - b\} \tag{8}$$

$$BNR_B{}^b(X) = \overline{R}_B{}^b(X) - \underline{R}_B{}^b(X) \tag{9}$$

$$= \mathbf{U}\{[x]_B : P(X/[x]_B) \in (1-b, b)\}$$

$\underline{R}_B^b(X)$ and $\overline{R}_B^b(X)$ are respectively called the R_B lower and R_B upper approximation of X with precision level b . Here, $P(X/[x]_B)$ is referred as conditional probability function :

$$P(X/[x]_B) = \frac{|X \mathbf{I} [x]_B|}{|[x]_B|} \quad (10)$$

where $|X|$ is the cardinality of the set X .

The rough degree of uncertainty can be measured by b -accuracy $a_B^b(x)$. It can be defined as:

$$a_B^b(x) = \left|\underline{R}_B^b(X)\right| \Big/ \left|\overline{R}_B^b(X)\right| \quad (11)$$

As b decreases, the boundary region of the VPRSM becomes narrower. Namely, the size of the uncertain region is reduced. Hence, the VPRSM have some tolerance to the noise [9,10]. The VPRSM can come back to the original rough set model when $b = 1$.

2.5 Knowledge Dependability and Approximate Reduction

For a given information system $S = (U, A, V, f)$, $A = C \mathbf{U} D$, C is called the condition attribute set, while D is called the decision attribute set . $X \subseteq U$, $B \subseteq C$. The measure of classification quality is defined by b -dependability of knowledge as follows:

$$g^b(B, D) = \left| \mathbf{U} \left\{ [x]_B : \frac{|X \mathbf{I} [x]_B|}{|[x]_B|} \geq b \right\} \right| \Big/ |U| \quad (12)$$

The b -dependability $g^b(B, D)$ measures the proportion of objects in the universe for which classification is possible with the precision level b .

In VPRSM, knowledge reduction is aimed to select the minimum attribute subsets of C which don't change the quality of classification with the precision level b [11]. Assumed $red^b(C, D)$ is b approximate reduction, then

$$g^b(C, D) = g^b\left(red^b(C, D), D\right) \quad (13)$$

No proper subset of $red^b(C,D)$ at the same b value can also give the same quality of classification. In other words, if any one attribute from $red^b(C,D)$ is eliminated, the formula (13) will be not valid.

A decision system may have many b - reductions. The intersection of all reductions is called Core. People always pay attention to the reduction that have the least attributes—minimal reduction sets, because the minimal rule sets of the decision system can be obtained through the minimal reduction sets. However, it has been proved that the minimal reductions of the decision system are the NP-hard [12]. The most important thing is to find a reduction algorithm with low computation cost.

2.6 Rules Acquisition

For decision system $S=(U,A,V,f)$, $A=C \cup D$, any expressions $q \to y$ is called a decision rule, where q expresses cause and y expresses effect. If $q \to y$ is true, then $q \to y$ is consistent in *S*. Otherwise, it is inconsistent. For $a \in C$, if $C-\{a\}$ and *D* is consistent, then attribute *a* is called dispensable. Otherwise, *a* is indispensable. If all attributes of *C* are indispensable, then *C* and *D* are independent.

Let $D=\{d\}$, $V_d=\{1,2,...,n\}$, then $U/D=\{D_1,D_2,...,D_n\}$, the decision class $D_i=\{x \in U, d(x)=i\}$. In VPRSM, for each $x \in \underline{R}_B^b(D_i)$ a decision rule may be generated as follows

$$\bigwedge_{r \in C}\left(r, r(x)\right) \to d=i \qquad \textbf{(14)}$$

3 VPRSM Based Approach for Fault Diagnosis

3.1 Features Extraction to Compose Decision Table

In fault diagnosis, many feature parameters can be used to reflect working state of equipment. But it is impossible to use all parameters to judge whether equipments work normally. We can only extract the necessary fault feature attributes and from these features obtain diagnosis rules. For fault diagnosis of rotating machines such pumps and motors, time domain, frequency domain and amplitude value domain can regard as fault features. Because the vibration signals of rotating machines in the frequency domain are obvious, the frequency domain feature of vibration signals is regarded as main fault feature. The damage extent of fault is evaluated by energy distribution of frequency. And the power spectrum data is smaller influenced by noise

than the time series data. Hence in this paper, power spectrum data is used as fault diagnosis signal. Power spectrum represents the distribution of signal energy with frequency.

In fault diagnosis of rotating machines, rotor unbalance, bearing oil whip and misalignment are the common faults. The three typical faults were simulated in our rotor test-bed. The features of power spectrum obtained from the rotor test-bed are used for the attributes of decision table. We take the three typical faults to study the steps of the diagnosis rules acquisition. In application of VPRSM, a decision table must be first built. Then attributes of decision table are discretized because the VPRSM cannot deal with continuous attributes. Fault feature is selected by removing unreasonable features with narrow intervals. Suppose E_V is the average value of vibration energy for each feature frequency spectrum, then

$$E_V = \frac{\sqrt{\sum_{j=1}^{n} A_j^{\,2}}}{\sqrt{N_{BF}}} \tag{15}$$

Where A_j is the amplitude of FFT spectrum; N_{BF} is the width of noise band with window (for Hamming window, $N_{BF} = 1.5$). By calculating the average value of vibration energy E_V and choosing a proper threshold m [13], the attribute value C_i of decision table can be calculated.

If $E_V \geq m$ Then $C_i = "1"$

Else $C_i = "0"$

$C_i = "1"$ represents abnormal spectrum; $C_i = "0"$ represents normal spectrum.

3.2 Fault Features Reduction and Diagnostic Rules Acquisition

In fault diagnosis, there are not one to one relations between the fault reasons and the fault symptoms with relatively high information redundancy. Some of the diagnostic information is interrelated with each other. Others are independent. The VPRSM can effectively eliminate the redundant information to reveal the diagnosis rules by means of reduction and mining.

For decision table $S = (U, A, V, f)$, $A = C \cup D$, U is the data set of the historic fault samples; Condition attribute C is the fault symptom sets; Decision attribute D corresponds with the result of the fault classification. Firstly, we can ascertain the appropriate precision level b . For noisy domains or inconsistent data, b takes smaller value, otherwise b takes bigger value. For the discretized decision

table, we calculate its upper and lower approximation sets with precision level b and its b -dependability [7]. The ability of approximate classification of the decision table can be evaluated at precision level b . Subsequently, reduction of decision table is made, which includes the condition attribute reduction and the diagnostic rule reduction. For condition attribute reduction, check the condition attributes whether all the attributes are indispensable. For a decision table, if the quality of classification with the precision level b does not changed by eliminating an attribute, then this attribute can be deleted, otherwise it can't be deleted.

Similarly, we can reduce diagnostic rule from an attribute reduction to get a set of tidy [8]. The maximum number of condition attribute values of a rule is removed without decreasing the classification accuracy of the rule. Generally, we select the least attributes and the simplest diagnostic decision rules.

4 Example

The fault diagnosis for rotating machines is used as an example. By rotor experiments 50 groups sample data of rotor are collected such as table 1. $U = \{x_1, x_2, ..., x_{50}\}$ is a finite sets of the fault samples; The condition attribute $C = \{c_1, c_2, ..., c_8\}$ is the fault symptom sets, und $C_1, C_2, ..., C_8$ indicate respectively frequency range, here f_0 is rotating frequency. Decision attribute D means the fault type. $D = \{1,2,3\}$ shows that the sample have respectively the rotor unbalance fault, oil whip fault and rotor misalignment fault symptom. Because the reference point of power spectrum obtained from the rotor test-bed does not lie in zero point, so it must be converted to zero reference point. The maximum point of absolute value of negative number is as zero point during computing. Then the attributes of decision table are discretized, the attribute value is {1, 0}, which represents respectively abnormal spectrum and normal spectrum. The decision table of fault diagnosis is built, see table 2.

Table 1. Fault diagnosis cases

U	0.01~0.39 f_0	0.40~0.49 f_0	0.5 f_0	0.51~0.99 f_0	f_0	2 f_0	3 f_0	5 f_0	D
	C_1	C_2	C_3	$C4$	C_5	C_6	C_7	C_8	
x_1	-32.20	-28.03	-30.15	-23.27	8.09	-20.21	-22.43	-16.84	1
x_2	-29.11	-30.65	-32.76	-11.20	9.68	-21.07	-19.18	-18.50	1
x_3	-20.19	24.05	22.86	19.06	0	-14.43	-28.81	-17.66	2
…	…	…	…	…	…	…	…	…	…
x_{50}	-44.37	-48.10	-49.28	-36.22	-18.79	-8.17	-10.01	-22.94	3

Table 2. The decision table of fault diagnosis

U	0.01~0.39 f_0	0.40~0.49 f_0	0.5 f_0	0.51~0.99 f_0	f_0	2 f_0	3 f_0	5 f_0	D
	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	
x_1	0	0	0	0	1	0	0	0	1
x_2	0	0	0	1	1	0	0	0	1
x_3	0	1	1	1	0	0	0	0	2
…	…	…	…	…	…	…	…	…	…
x_{50}	0	0	0	0	1	1	1	0	3

Table 3. The core value table of diagnostic rule for { c_2, c_5, c_6}

Rule	0.40~0.49 f_0	f_0	2 f_0	D
	C_2	C_5	C_6	
1	0	1	*	1
2	*	1	*	1
3	1	*	*	2
4	1	*	0	2
5	*	*	*	3
6	*	1	1	3

Table 4. The diagnostic rules of reduction for { c_2, c_5, c_6}

Rule	0.40~0.49 f_0	f_0	2 f_0	D
	C_2	C_5	C_6	
1	0	1	*	1
2	0	1	*	1
2’	*	1	0	1
3	1	*	0	2
3’	1	0	*	2
4	1	*	0	2
5	0	*	*	3
5’	*	1	*	3
5”	*	*	1	3
6	*	1	1	3

The steps of the rotor fault feature extraction and diagnostic rules acquisition based on VPRS are as following:

Step 1 The decision table is simplified, and the same samples are combined.

Step 2 Computing respectively all equivalence classes for fault symptom sets *C* and fault type sets *D*, namely $U/C=\{X_1,X_2,...,X_i\}$ and $U/D=\{Y_1,Y_2,...Y_j\}$.

Step 3 For inconsistent equivalence class X_i, computing respectively conditional probability function $P(Y_j/\ X_i)$.

Step 4 The appropriate precision level b is ascertained. For a given probability value b, the b-positive region corresponding to a concept is delineated as the set of objects with conditional probabilities of allocation at least equal to b. For noisy domains or inconsistent data, b takes smaller value, otherwise, b takes bigger value. Because the power spectrum data has smaller influence by noise than time series data, by computing we take $b=0.8$.

Step 5 Lower approximation $\underline{R}_B^b(X)$ and upper approximation $\overline{R}_B^b(X)$ are calculated with precision level $b=0.8$. Then calculate its b-accuracy $a_B^b(x)$ and b-dependability $g^b(B,D)$. It results that $a_B^b(x)$ is equal to 1.0 and $g^b(B,D)$ to 1.0. This indicates that the approximate classificatory ability of decision table with precision level $b=0.8$ is accord with request.

Step 6 The fault symptom sets (condition attribute) are reduced. The Core of the condition attribute reduction is calculated, here the Core is empty. There are four simple reduction subsets of condition attribute *C* relatively to decision attribute *D*. Those are { c_2, c_5, c_6},{ c_4, c_5, c_7},{ c_3, c_6, c_7 } and { c_3, c_4, c_6, c_7 }.

Step 7 The diagnostic rules are reduced and minimization of the diagnostic rules is acquired. Limited by the length of the paper, we only provide the rule sets of reduction subsets {c_2, c_5, c_6}. The Core value of each diagnostic rule can be calculated, see table 3. In table 4, the diagnostic rules of reduction are given.

Step 8 The redundant knowledge rule of the decision table is eliminated. The corresponding rules are synthesized. The diagnostic rules are acquired as follows:

$C_2[0]\ C_5[1] \rightarrow D[1]$
$C_5[1]\ C_6[0] \rightarrow D[1]$
$C_2[1]\ C_5[0] \rightarrow D[2]$
$C_2[1]\ C_6[0] \rightarrow D[2]$
$C_5[1]\ C_6[1] \rightarrow D[3]$

The rules above can be also expressed:

1) If the spectrum in f_0 is abnormal, and the spectrum in (0.40~0.49) f_0 and in

$2f_0$ are normal, then exists the rotor unbalance fault.

2) If the spectrum in (0.40~0.49) f_0 is abnormal, and the spectrum in f_0 and in $2f_0$ are normal, then exists the rotor oil whip fault .

3) If the spectrum in f_0 and in $2f_0$ are abnormal, then exists the rotor misalignment fault.

The diagnostic rules above are correct and consistent with the actual experiments result. These rules can be regard as diagnostic knowledge to diagnose other fault sample data.

5 Conclude Remarks

In fault diagnosis there is no one to one relation between the fault reasons and the fault symptoms. Therefore the feature extraction and rules acquisition from redundant fault information data are always difficult problem in the traditional fault diagnosis. In the paper, a method based on VPRSM is proposed to deal with fault diagnosis feature extraction and rules acquisition. The data information of fault diagnosis is obtained directly from vibration signal system in which there are the existence of noisy data and uncertain information. The method can effectively eliminate the redundant fault attributes and acquire the correct diagnosis rules from noisy fault information. Simulation results for rotating machines show that the reduction and classification of the diagnostic knowledge can be realized and the correct diagnosis rules can be acquired by the presented method. This method is more effective to rule mining of the fault diagnosis. It provides foundation in theory and application for the intelligent fault diagnosis.

References

1. Pawlak, Z.: Rough sets. International Journal of Computer and Information Sciences. 11(5) (1982) 341-356
2. Pawlak, Z.: Why rough sets. Proceedings of the Fifth IEEE International Conference on Fuzzy Systems, 2, (1996) 738-743
3. Ziarko, W.: Analysis of Uncertain Information in the Framework of Variable Precision Rough Sets. Foundations of Computing And Decision Sciences , 18(3-4), (1993)381-396
4. Katzberg, J.D., Ziarko, W.: Variable Precision Extension of Rough Sets. Fundamental Informatics 27, (1996)155-168
5. Burrell, P., Inman, D.: An Expert System for the Analysis of Faults in an Electricity Supply Network: Problems and Achievements. Computers in Industry, 37,(1998) 113-123
6. Durkin, J.: Expert Systems. Design and Development, Macmillan, (1994)
7. Zhang, D.-F., Wang, Z.-Q.: VPRS Model Approach to Fault Feature Selection and Diagnostic Rules Extraction. Journal of system simulation, 15(6),(2003) 793-796
8. Zhang,W.-X., Wu, Z.-W., Liang, J.-Y.: Rough Set Theory and its Method. Science Press, Beijing(2001)

9. Chen, X.-H, Zhu, S.-J, Ji, Y.-D, Li, Y.-M.: Generalized Rough Set Model and its Uncertainty Measure. Journal of Tsinghua University, 42(1) , (2002)128-131
10. Mi, J.-S, Wu, W.-Z, Zhang,W.-X.: Approaches to Knowledge Reduction Based on Variable Precision Rough Set Model. Information Sciences 159(2004) 255-272
11. Beynon, M.: Reducts within the Variable Precision Rough Set Model. A Further Investigation. European Journal of Operational Research 134 (2001) 592-605
12. Skowron A, Rauszer C.: The Discernibility Matrices and Functions In Information Systems. Intelligent Decision Support-Handbook of Applications and Advances of the Rough Sets Theory, Kluwer Academic Publishers, (1992)331-362
13. Hu, T., Lu, B.-C., Chen, G.-J.: A Gas Turbo Generator Fault Diagnosis New Approach Based on Rough Set Theory. Journal Of Electronic Measurement And Instrument, 15(1), (2001) 12-16

An Intelligent Agent to Support Collaboration within a Distributed Environment

Bogdan-Florin Marin[1], Axel Hunger[1], Stefan Werner[1], Sorin Meila[1], Christian Schütz[1]
[1] University of Duisburg-Essen, Oststrasse 99, 47057 Duisburg, Germany
{bmarin, hunger, swerner, meila, c.schuetz}@uni-duisburg.de

Abstract. The motivation for the usage of software agents in University-Education can differ. The Institute for Multimedia and Software Engineering at the University of Duisburg-Essen (UDE) is involved in a research project which aims at the development of new education concepts in virtual learning environments and also at the internationalization of university education which plays an important role nowadays. This paper goal is to show how to integrate agent technology to support collaborative learning in distributed environments. The aim of this research is to provide the first steps to define a method for creating a tutor agent which can partially replace human-teachers and assist the students in the process of learning.

1 Introduction

The communication and interaction between the people, intending to work together on related tasks using a computer from different locations has been improved by terms of Computer Supported Collaborative Work (CSCW) systems. The advancements in the CSCW technologies had led a further research into computer supported collaborative learning (CSCL) for the educational purposes. CSCL supports multiple students learning in the collaborative process across the same networked servers to communicate the ideas and information, access information and documents, provide feedback and facilitate the group activities by the help of computer supported systems.

Basic research in Computer-Supported Collaborative Learning (CSCL) and educational psychology focused on identifying potential indicators of effective collaborative learning teams ([1], [2] and [3]). Also, CSCL research explored the types of problems that may result from insufficient group interaction and support ([4], [5], [6] and [7]). This resulted in several research approaches for observing, analyzing and assessing collaborative learning interactions as well as for developing methods and tools that provide guidance and support to on-line collaborative learning teams ([8], [9], [10], [11], [12] and [13]).

Due to the use of collaborative learning in classroom which has proven to be effective in improving academic achievement, motivation for learning and social interaction [14], the interest in collaborative environments has increased considerably

[15] therefore studies into creation of collaborative learning environments on the Internet have shown a topic of great interest for many researchers [16].

In this paper, "collaborative learning" does not mean only "Computer Supported Collaborative Learning (CSCL)" based on computers and information networks. It also refers to a new method of group learning which is effective not only for knowledge acquisition, but also for meta-cognitive skills [17], [18] and communication skill acquisition [19].

Distance educators and their students benefit greatly from the use of online collaborative techniques. During the last years, the traditional text-based conferencing methods have been complemented by audio/video-conferencing methods, whiteboard, polling, and instant messaging techniques. These operate in synchronous and asynchronous communication modes, on a variety of computer platforms, and in stand-alone and integrated designs. Our research is also motivated by a real problem: distance education.

The development and the setup of a new international degree course are described in [20]. German students are required to spend at least one semester abroad. With the usage of the new media and communication technologies it should be easier for foreign students to follow the studies in Germany. Further, it provides german students the opportunity to take part in Software-Engineering Course during their abroad period of study, but also have the chance of getting knowledge in systems known as CSCW environments. This was the main motivation for the development of a synchronous groupware called Passenger and its application in Software-Engineering-Education.

The traditional "Computer Supported Cooperative Work and Software Engineering" lab at UDE is conducted as a project setup of student teams, each consisting of four persons: three students and one tutor, where the same tutor can be in several virtual teams. That can cause problems in terms of tutors' availability if the virtual teams meet at the same time but also if the teams meet at times outside the tutor consultation hours. To make sure that at least a virtual tutor is always available agent technology shall be used. Therefore, a novel idea and a framework of intelligent tutor agents conducting a Software Engineering lab using Passenger is presented in this paper.

This paper is structured as follows: the next section evaluates the need of a new educational environment to support spatially distributed teams and also presents basic design concepts of the Passenger groupware, the third section highlights several aspects of software agents and also introduces the reasons for choosing agents as tutoring knowledge elements. This section provides also an overview of related work in this research field. Section 4 presents the tutor based agent and its functionality within the Passenger groupware. Final section presents the outcome of this research.

2 The Synchronous Passenger Groupware

Computer support for group learning is augmenting by the dissemination of computer technology networks. There is a great variety of new applications and possibilities, although one of the major challenges is the transition period required for transporting the applications, methods, methodologies and techniques of the real world to the virtual world. This is where the work in the fields of Computer Supported Cooperative Work (CSCW) and Software Engineering join up. Through the studies for group work and the methods for developing software it is possible to model and design software that is proper for supporting groups which is called groupware.

Within the framework of the lab for the lecture "CSCW and Software-Engineering" at UDE, the students are trained using the virtual lab concept to work in spatially distributed teams. The virtual laboratory concept is quite general encompassing a range of technologies and human factors that are necessary for operation in any remote environment, whether remote in time, scale or distance.

The organization and execution of such a practical course/lab with spatially distributed participants require tools for support in different phases:

- Most importantly, the organization of the practical course must be supported. For example that the students are to announce within the prescribed period and that the supervisors can generate the group arrangements and schedules from these data.
- In meetings, relevant meeting information must be announced such that the participants must be invited explicitly to a meeting. The relevant meetings information consists of information such as dates and times of the meetings as well as connecting information.
- During a meeting tools to support one or several forms of co-operation are necessary: the synchronous group work, the asynchronous group work and the individual work. These tools must ensure additionally the conflict-free access to common resources.

Mainly the application of public available tools like Microsoft Netmeeting or CUSeeMe and their usability on the basis of conducting a distributed software engineering lab can be found [21]. These tools are rated as more or less suitable for the application in Software Engineering labs, which are conducted in the above mentioned manner due to the special requirements resulting from the educational perspective.

For the special case of the software engineering lab regarded here, no complex problems are to be solved concerning the organization of the practical course or the invitation to a meeting. This could rather be solved by using standard Internet technologies such as HTML (Hyper Text Markup Language) and web-databases. The emphasis lies on the development of a new educational environment for the support of the students during a meeting in the context of a Software Engineering lab.

An educational environment must be constructed effectively so that students who participate into distributed teams learning process through Internet or information

network could interact with other team-members timely and friendly. Namely, the collaboration is the most important subject with a view to designing an advanced computer supported educational environment. In order to attain this subject, it is necessary to discuss an architectural framework from the required resources points of view:

- Technologically mediated communication channel
- Shared workspace for a team/group
- Personal workspace
- Learning materials/ learning tools

In order to create an educational environment for the spatially distributed teams, a synchronous groupware called “PASSENGER” was developed at the University of Duisburg – Essen throughout the last years. A client-/server architecture has been chosen whereby the server is located at the university, due to the fact that the university plays a major role in this configuration.

The specified requirements for a groupware used in a Software-Engineering-lab give a direction how the function- and application-classes of the Passenger-Client were defined. Therefore the Passenger-Client consists of a communication component, a cooperation component and several shared tools and resources to carry out Software Engineering tasks.

The Passenger Client user interface (see Figure1) contains video screens of each member and a whiteboard that is divided in a public window for common process on the outline documents and a private window for individual process on the documents.

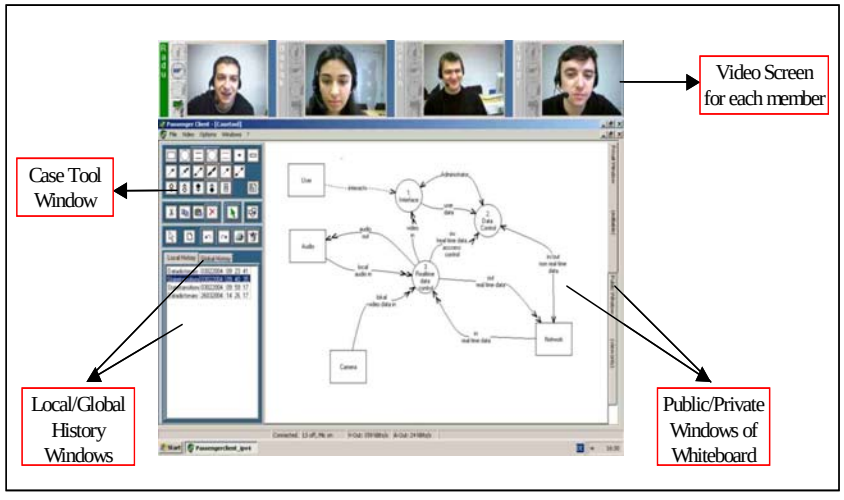


Figure 1 Passenger Client

Each member has the same view of the public window according to the What You See Is What I See (WYSIWIS) principle, but only one of them can alter the document at a certain time. Each member is also equipped with a private working window to try out own ideas and to work simultaneous on an individual solution.

Three essential differences of our groupware compared to publicly available solutions can be specified:

a. *Passenger floor control [22]*: the advantages of the developed Floor-Control protocol are to guarantee a defined fairness and to prevent the mutual exclusion and blocking. Thereby, the fairness definition is based on a theoretically equal distribution of the Floor-holding concerning the occurrence. Anyway this type of equal distribution is not forced. Since the defined roles in the group process model are opposed to any kind of equal distribution, the arrangements to guarantee the equal distribution of the Floor were renounced. In particular, the Passenger Floor Control does not limit Floor-holding duration.

b. *A user interface designed to support group awareness [23]*: The design is based on common requirements and the special requirements from the analyzed group behavior. For all design decisions thought has been given to the requirement for measures to increase the group-awareness. Therefore a concept for the positioning and resizing of the communication windows was developed and implemented. Especially solutions for the Floor-Control and the group-awareness were developed during the design of the user interface. Group awareness functions are implemented by means of providing all needed information for a late coming in participant to discover the actual conference state. This is implemented by highlighting the video screen of the person who has access to the shared resources.

c. *The whiteboard concept materialized in tools to carry out software engineering tasks*. The implemented PASSENGER-CASE-Tool for the software design features its concept for realizing the private and public work area and its process specified support for software engineering. The separated realization between the work and the display area enables the Floor-Holder firstly to simultaneously access the last two design documents. Due to this fact he is simply able to compare the two schemes by switching between the work and display window. The rest of the participants have the same possibility under condition that they transfuse the content of the display field of their work area. The transfer of arbitrary documents to the work area of the Floor-Holder and there to the display area of the others, should not be commented, in order to set the process rights and the access to the speaking channel in a shareable Floor-Control.

The Passenger groupware contains also a Telepointer which serves to elucidate and present the facts. The Telepointer is implemented as a collaborative service which can be used by the Floor-Holder. This user can lead the attention of the other participants to some objects or screen areas of the public window during a discussion.

A more detailed description of the functionality of this groupware or its usage in Software Engineering can be also found in [24].

3 Software Agents

In the last few years, software agents' technology has become an interesting field of research. The concept of *agent* has become important in both Artificial Intelligence and mainstream computer science. Many researchers consider agent technology as the translation of social theories into computer programs [26].

Three key issues can be identified concerning software agents:

- *Agent theory* represents essentially specifications, which can define an agent and its properties.
- *Agent architecture* describes the step from specification to implementation.
- *Agent languages* are programming languages which allow to control or to interact with the agents

Starting from a simple comparison between a human agent - which is a person who acts autonomously and behaves intelligently - and a software agent, several issues can be noticed:

- Agents as intentional systems[1]: is it legitimate or useful to attribute human characteristics like beliefs, desires to artificial agents? Being an intentional system seems to be a necessary condition for an agent, but is it a sufficient one?
- Knowledge is a pre-condition for an agent's actions: what an agent needs to know in order to perform several actions?
- When building intelligent agents it is important that a rational balance is achieved between the complexity (hardware resources) of an agent and the tasks that the agent should perform.

There are several basic characteristics, which any intelligent software agent should have:

- *autonomy* is the ability of an agent to operate without the direct intervention of humans or others, and have some kind of control over its actions
- *social ability* represents the possibility to interact with humans (maybe with other agents) via some agent-communication languages
- *reactivity* represents the ability of an agent to perceive its environment and respond to the changes that occur in it

[1] Human behaviour and activity are predicted and explained through the attribution of attitudes such as believing (She took her umbrella because she *believed* that it was going to rain), wanting (He worked hard because he *wanted* to posses a new car), fearing, hoping and so on. An intentional system describes entities whose behaviour can be predicted by the method of attributing belief, desires [25].

- *pro-activeness* is the ability of an agent to exhibit goal-directed behaviour by taking the initiative.

3.1 Why Software Agents as Tutors?

Unfortunately, there is not yet an effective evaluation of the impact of agent-based architectures properties such as mobility, autonomy, distribution and homogeneity. Who should develop and build intelligent agents? Should one use static or mobile agents? What is the adequate autonomy degree? How can the agents be integrated into existing learning environments? How many agents should be used in order to partially replace a human teacher? How different should those agents be? How much will future intelligent learning environments cost, and what kind of resources and support will they require? The answers to these questions would provide the first step for setting a framework of pedagogical-software agent-based solutions to remove all the obstacles from distance learning environments. These and other questions related to the design, development, integration, implementation, maintenance, and cost of intelligent learning environments are dominating most of the researchers in this field.

The application of agents in the educational sector comes about mainly in the form of personal assistants, user guides, alternative help systems, dynamic distributed system architectures, human-system mediators and others. As a result of all of the changes that have taken place in the educational system, one now sees the increasing emergence of complex and dynamic educational infrastructure that needs to be efficiently managed. Corroborating this, new (types of) educational mechanisms and services need to be developed and supplied.

In particular these services need to satisfy a series of requirements such as personalization, adaptation, support for user mobility, support for users while they are dealing with new technologies, among others. Agents emerge to provide solutions for these requirements in a way that is more efficient when compared to other existing technologies [27].

According to Aroyo and Kommers [27], agents can influence different aspects in educational systems. They supply new educational paradigms, support theories and can be very helpful both for learners and for teachers in the task of computer-aided learning.

Lees and Ye [28] believe that the application of the agent paradigm to CSCW potentially can exchange information more fluid among the participants of groupware systems (as decision-making systems), help in control of the process flows and also supply groupware interfaces. These ideas also are applicable to other domains, such as is the case of interactive learning.

According to Kay [29], in the first computer-assisted teaching environments the idea was to build "teachers" who could transmit knowledge to the learners. Currently, these types of environments are more geared up for exploration on the part of the learners, designing, building and using adaptive systems as tools. These environments also are being built to give greater responsibility to the learners regarding aspects of the learning process, and especially regarding control of its model, which is the central aspect in the adaptability of the tools.

For McCalla & all [30], learner models may have a variety of purposes depending upon the type of knowledge that needs to be stored and processed. For them, the computation of all of the learner (sub-)models of an environment can be computationally expensive and not always necessary. In the work cited four purposes are presented for a model: reflection, validation, matchmakers and negotiation.

For Kay [29], there are several problems from the learners' point of view. One is the increase in the power of choice and control over the model. This could increase the learners' workloads or even turn into a distraction. In this case, the learners should take advantage of the moments such as the end of a course or a topic to evaluate and reflect upon their participation and the learning process. Another potential problem is incorrect data being supplied by the learners. The solution adopted in this work for that problem was to store the type of information learners are providing and the type the environment extracts.

For Jennings & all [31], autonomous agents and multiagent systems represent a new modality of analyzing, designing and implementing complex software. The agent concept has a wide area of applications, ranging from the creation of personal assistants to air traffic control systems, electronic commerce and the group work support.

Our research aims at showing how to integrate agents in a synchronous collaborative environment in order to help students in the process of learning and the benefits from this human-computer interaction.

The fundamental reason for introducing agents as tutoring knowledge elements is their capabilities of communication and interaction. These characteristics are fundamental for agent's usage in an educational environment.

Tutoring agents are entities whose ultimate purpose is to communicate with the student in order to efficiently fulfill their respective tutoring function, as part of the pedagogical mission of the system [32].
Some of these tutoring functions are:

- to present a topic
- to explain the topic
- to give an example
- to answer a student's question
- to ask a student a question and evaluate the student's answer
- to examine and diagnose a student's behaviour during the learning process

This means that the agent should also have the roles of a human tutor within a group of students, which are:

- *Interrogator* – poses questions and the students of a collaborative group then provide answers. The questions should provide help for the students to reach a common learning goal.

- *Reviewer* – analyzes the students' answers, including whether it is correct or not.
- *Monitor* – records the answers from all the students and the communications among students during the collaborative learning process.
- *Instructor* – gives individualized instructions and helps those students who cannot keep up with the progress of their group-mates.

All together: the tutor-agent should be able to present and explain a learning subject, to pose questions about it, to evaluate the learner answers and also to provide specific feedback. The tutor-agent should generate relevant replies from a knowledge base in response to the questions posed by the learner. If the tutor-agent cannot generate an adequate response to one question then it should communicate with other tutors (human or agents) in order to accomplish its task.

4 Agent-Tutor in Passenger

Choosing an environment is an important decision for agent researchers and developers. A key issue in this decision is whether the environment will provide realistic problems for the agent to solve, in the sense that the problems are true issues that arise in addressing a particular research question.

Due to the fact that a large number of Passenger sessions can run simultaneous at the same time, it appeared the need of a system to monitor and manage all these sessions. The first step in this direction was a web-based system called Watchdog which monitors the activities of all Passenger Servers (see Figure2).

The following scenario can be assumed: three students would like to meet on-line using Passenger and learn together or discuss about the topics learned from their previous lecture. Also, besides the fact that the students might need supervision the need of a tutor may appear if there is a disagreement between the students. Therefore, during that session if help of a tutor is needed, one of the students can press "Call Tutor" button. This request will be send to the Watchdog and the system will provide this request also with the exact parameters of the session (IP address of the Passenger server where that session is hosted) to any available human-tutors.

If there are no human-tutors available, then a tutor-agent will connect to the current session and it will try to provide necessary help to the participants. In Passenger learning environment the presence of an intelligent-agent, that can perform a kind of tutoring role, could be benefic in helping the students to reach their common goal.

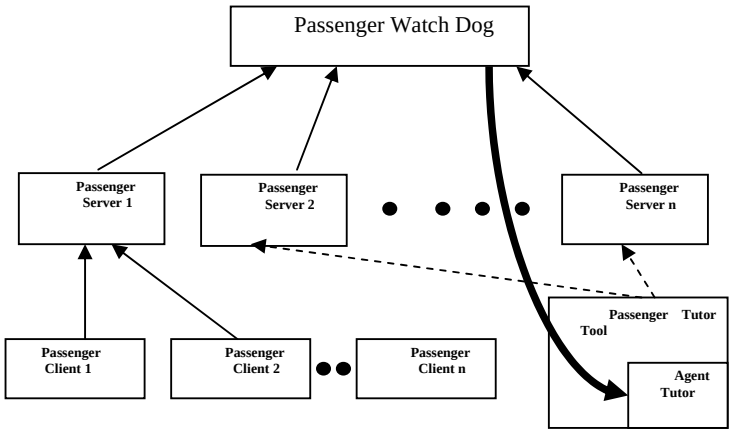


Figure 2 Passenger WatchDog-Server

The roles of the agent – tutor within the Passenger groupware are:

- *Selects a model (topic) for session/discussion*: it is through this negotiation of meaning and understanding that learning occurs. Therefore each topic has a tree structure, with nodes that are: first question for the participants, possible answers by participants, agent response to each of these answers. Topics are designed to attract participants into an interactive dialogue and to avoid the "silence" during a Passenger session.
- *Assigns roles to the students:* during the semester the student-teams will experience the entire life-cycle of Software Engineering. The students start with a requirement analysis following the Ward & Mellor [33] approach during the modeling phase. The given problem for the practical training is chosen in such a way, that it cannot be solved by one student on its own. Therefore, each topic is divided in sub-topics which can be assigned by the tutor agent to one of the participants.
- *Provides help for toolbox buttons:* each tutor-agent is able to provide students basic help regarding the usage of the Passenger Client. Within the tutor-agent's architecture there is implemented a pattern recognition algorithm. Using this algorithm the agent can provide adequate answer to students' questions like: "*How (1) can I draw (2) a control transformation (3)?*" where (1)(2) and (3) ~How… draw… control transformation…~ represent a pattern example. After recognizing a pattern the agent will search its knowledge database for a proper answer and will provide this answer to the student. For this example the answer is "*You should press the third button of the CaseTool buttons from the first row, and then go with the mouse in your working area and click where you want to have a control transformation…* "
- *Supports and gives hints on awareness functions:* the tutor agent has the ability to provide to participants proper feedback on awareness issues like:

"Why can't we see Jack? (*Answer: Jack should press F3 or select send video from Video, or maybe Jack does not have a video-camera*)"

- *Controls and gives hints on floor control mechanism or selects floor passing method (adaptive):* the agent can provide answer to questions like: "Why my colleagues cannot hear me? (*Answer*: *you must be the actual floor holder in order that the others can hear you, therefore you should request the rights. There is a button on* …)" or it should be able to avoid the deadlock situations like: one student which is the floor holder leaves the session but she/he forgets to pass the floor, therefore the other participants cannot modify the common artifact or they cannot communicate. One of the remaining participants can ask the floor from the agent-tutor. The agent can notice that the actual floor holder is inactive (e.g. he hasn't made any changes to the common document for more than 10 minutes). Therefore the agent has the ability to take the floor from the inactive participant and to give it to the one that has requested for it.
- *Gives hints for next steps in modeling:* during a session it can occur that the students might reach a deadlock- the students do not know how to continue their work to fulfill their common task. The agent analyzes the current state of the students' work and to provide hint for the next steps. If the agent cannot accomplish this task then it should communicate with other tutor-agents from another Passenger sessions. If the other agents cannot provide a proper answer then the analyze evaluation should be communicated to a human-tutor. The human-tutor if he is available can replace that agent within its session or he can provide the agent the adequate answer. The communication and cooperation with other agents, or human-tutors is realized by specific language as KQML [34], [35].

To make the tutor agent design process easy and accessible (see Figure3), we proposed a framework where the agent is composed of a set of categories of components:

- *navigational components*: that are responsible for agent's travel itinerary (within a network) when it needs to communicate with other tutors (humans or agents) in order to receive help for accomplishing its task or with the Passenger Watch Dog server.
- *Core components* which are divided into:
    - *performing components*: that are responsible for executing one or more learning tasks. Also these components compose the agent decision system – agent can choose his actions.

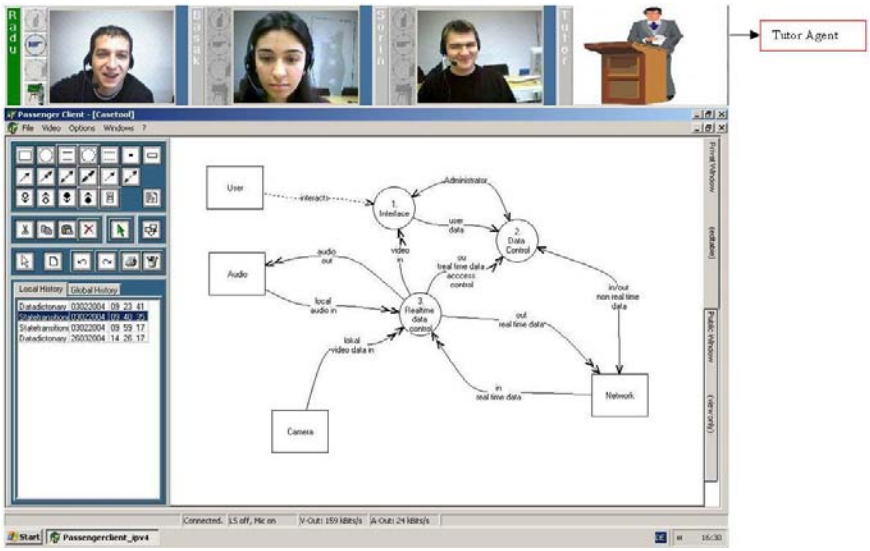


Figure 3 Tutor-Agent in Passenger

- *reporting components*: that determine how the results are collected and reported back. The agent has the ability to monitor the activity of each participant in a Passenger session and to evaluate the level of a student's understanding and provides help on those topics which the student didn't understand. This evaluation and also the agent-student interaction is recorded in a database, so that any human teacher can later check.

- *Agent's interface* is an animated cartoon with human like gestures. We choose to design and implement our own animated agent instead of using the Microsoft Agent to ensure Passenger platform independence and extensibility.

Besides all of these described components the agent contains also a self learning mechanism. Learning is a very important aspect of intelligent agents. The Passenger tutor – agent can learn to improve its performances from its previous interactions with students or with other tutors: humans or agents. It interacts with students to provide the tutoring materials, helps and hints, to pose questions and to analyze students' answers. Based on these interactions, the agent chooses that accommodates not only the students' needs or preferences but also the tutoring goals. The agent's goals might be locally achievable which means that tutoring material can be retrieved from a local knowledge database, or require interaction with another tutor-agent from a different session or a human tutor in case of a word pattern for which it cannot find it in its own knowledge database.

The Passenger-tutor-agent's collaborative learning process is based on both the students' direct and indirect feedback. A simple learning algorithm is applied in the

monitoring component to record students' requests or behavior. When the same request has been recorded more times and also in different sessions, the agent notifies a human-tutor about this and also updates his parameters in order to perform future actions according to students' request. It also monitors and evaluates the student indirect feedback or reaction following a suggestion. As an example the following scenario can be assumed: the students are asked whether they need help/hints or not after taking more time on a topic than the time allocated. When the students choose a different solution/action than the suggested one or they refuse the help, the agent records this behavior as indirect feedback. Also a report concerning the students' behavior during a training session is send to the tutor after the session is over.

Most current intelligent tutoring systems are electronic page turning, with some hyperlinks and multiple-choice exercises (e.g. see Web_Soc [36]). Even if this electronic delivery provides self-paced learning it is not truly interactive. Passenger is a synchronous groupware to support spatially distributed teams, in other words is a truly interactive environment which emulates a real face-to-face discussion.

Therefore our research is based on an autonomous agent paradigm instead of an intelligent tutoring system, because of intelligent agents' capabilities of communication and interaction. These characteristics are fundamental for agents' usage in a collaborative educational environment.

5 Conclusions

In terms of the geographical distribution of the participants, which is one of the most publicized advantages of the Web-based education environments, there is much to gain through the use of the agent paradigm.

This paper goal was to show how to integrate agent technology to support collaborative learning in distributed environments. The aim of this research is to provide the first steps to define a method for creating a tutor agent which can partially replace human-teachers and assist the students in the process of learning.

Our tutor agent tries to replace partially the human teacher, in assisting the students at any time of their convenience and in the meantime the agent can evaluate the results of the students' activity during the learning process. This evaluation also provides a great benefit for a human teacher or tutor.

The basic contribution for this research is conception of a Distance Learning System in which human and artificial agents can collaborate to achieve a common learning goal.

Current commercial learning management systems could also benefit from the development of such agent – based capabilities. The resulting intelligent learning systems might use a variety of intelligent agents to offer dynamic – and smart – teaching and learning environments.

References

1. Johnson, D., Johnson, R., Holubec, E. J.: Circles of learning: Cooperation in the classroom 3rd edn. Edina, MN: Interaction Book Company (1990).
2. Webb, N.: Testing a theoretical model of student interaction and learning in small groups. In R. Hertz-Lazarowitz and N. Miller (Eds.), Interaction in Cooperative Groups: The Theoretical Anatomy of Group Learning. NY: Cambridge Univ. Press (1990), 102-119.
3. Koschmann, T., Kelson, A., Feltovich, P., Barrows, H.: Computer-supported problem-based learning. In T. Koschmann (Ed.), CSCL: Theory and Practice of an Emerging Paradigm. Mahwah, NJ: Lawrence Erlbaum (1996), 83-124.
4. Kiesler, S., Sproull, L. S. (Eds.). Computing and change on campus. New York: Cambridge Press, (1987).
5. Dobson, M., McCracken, J.: Problem based learning: A means to evaluate multimedia courseware in science & technology in society. In T. Muldner & T. C. Reeves (Eds.), Educational Multimedia & Hypermedia 1997. Calgary: AACE (1997).
6. Cameron, T., Barrows, H. S. Crooks, S. M.: Distributed Problem-Based Learning at Southern Illinois University School of Medicine. In C. Hoadley & J. Roschelle (Eds.). Computer Support for Collaborative Learning. Designing New Media for a New Millenium: Collaborative Technology for Learning, Education, and Training. Palo Alto: Stanford University (1999), 86-94.
7. Thomas, R.: Evaluating the Effectiveness of the Internet for the Delivery of an MBA programme. Innovations in Education and Training International (2000), 97-102.
8. Nurmela, K., Lehtinen, E., Palonen, T.: Evaluating CSCL Log Files by Social Network Analysis. Proceedings of the Int. Conference on Computer Support for Collaborative Learning (CSCL 1999), Stanford, California, USA (1999), 434-442.
9. Barros, M., Verdejo, M.: Analysing student interaction processes in order to improve collaboration. The DEGREE approach. Int. Journal of Artificial Intelligence in Education, Vol.11. (2000), 221-241.
10. Soller, A.: Supporting Social Interaction in an Intelligent Collaborative Learning System. Int. Journal of Artificial Intelligence in Education. Vol. 12 (2001), 40-62.
11. Martínez, A., Dimitriadis, Y., Rubia, B., Gómez, E., Garachón, I., Marcos, J.A.: Studying social aspects of computer-supported collaboration with a mixed evaluation approach. Proc. of the Int. Conf. on CSCL, Boulder, Colorado, USA (2002), 631-632.
12. Reiser, B.: Why Scaffolding Should Sometimes Make Tasks More Difficult for Learners. In Gerry Stahl (Ed.), Computer Support for collaborative learning: Foundations for a CSCL community. Hillsdale, NJ: Erlbaum (2002), 255-264.
13. Zumbach, J., Mühlenbrock, M., Jansen, M., Reimann, P., Hoppe, H.-U.: Multidimensional Tracking in Virtual Learning Teams. In G. Stahl (Ed.), Computer Support for Collaborative Learning: Foundations for a CSCL community. Hillsdale, NJ: Erlbaum (2002), 650-651.
14. Slavin, R.E.: Research on cooperative learning and achievement: What we know, what we need to know. Contemp. Educ. Psychol., vol21, (Jan 1996).
15. Pinto, M., Amor, M., Fuentes, L., Troya, J.M.: Collaborative Virtual Environment Development – An Aspect-Oriented Approach, 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01), Mesa Arizona (April 16-19, 2001).
16. Jennings, N.: "On Agent-based Software Engineering, Artificial Intelligence" No. **117**, Elsevier Press (April, 2000), 277-296.
17. Pintrich, P., De Groot1, E.,: Motivational and Self-Regulated Learning Components of Classroom Academic Performance, Journal of Educational Psychology, No.82 (1990), 33-40.

18. Spiro, R.J., Coulson, R.L., Feltovich, P.J., Anderson, D.K.: Cognitive flexibility: Advanced knowledge acquisition ill-structured domains, Proceedings of the 10th Annual Conference of Cognitive Science Society, Erlbaum, Hillsdale, NJ (1988), 375-383.
19. Endlsey, W.R.: Peer tutorial instruction Englewood Cliffs, NJ: Educational Technology (1980).
20. Hunger, A.: The concepts of the internationally oriented degree course, computer science and communication engineering, In Proceedings of the International Conference on Engineering Education, Rio de Janeiro, Brazil (August 17-18, 1998).
21. Sarjoughian, H.S., Zeigler, B.P., Ham, M., Parris, J.: Conducting distributed group software-engineering projects, challenges to state-of-the-art collaboration technologies, In Proceedings of WMC 99, SCS Publishing (1999).
22. Dommel, H.-P., Garcia-Luna Aceves, J.J.: Group coordination support for synchronous Internet collaboration, IEEE Internet Computing, (March-April 1999), 74-80.
23. Dourish, P., Bellotti, V. (Eds.): Awareness and coordination in shared workspaces", In J. Turnier and R. Kraut, Proceedings Of CSCW'92- Sharing Perspectives, ACM Press Toronto Canada (1992), 107-114.
24. Werner, S., Hunger, A., Schwarz, F., Schütz, C., Jung, M.: A Synchronous Groupware and Some Scenarios for Conducting a Software Engineering Lab with Distributed Teams, Proceedings of Iasted International Conference Computers and Advanced Technology in Education, Greece (June2003).
25. Dennett, D. C.: The Intentional Stance, The MIT Press (1987).
26. Ekdal, B., Davidsson, P.: A workable definition of computerized agents, 3rd World Multiconference on Systemic, Cybernetics and Informatics, Florida, USA (1999).
27. Aroyo, L., Kommers, P.: Preface - Intelligent Agents for Educational Computer-Aided Systems. Journal of Interactive Learning Research, Vol.10 (3/4), (1999), 235-242.
28. Lees, B., Ye, Y.: Preface of the Proceedings of ASCW01 – Workshop of Agent-Supported Cooperative Work, In: The 5th International Conference on Autonomous Agents, Montreal, Canada (2001).
29. Kay, J.: Learner Control. User modeling and User Adapted Interaction, 11, Kluwer Academic Publishers, Netherlands (2001),111-127.
30. McCalla, G., Vassileva, J., Greer, J., Bull, S.: Active Learner Modeling, In: Gautier, Frasson & Vanlehn (Ed.). Proceedings of ITS'2000, Springer LNCS 1839 (2000), 53- 62.
31. Jennings, N., Sycara, K., Wooldridge, M.: A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems, 1, Kluwer Academic Publishers (1998), 7-38.
32. Viccari, R.M., Martins-Giraffa, L.M.: The use of Agents techniques on Intelligent Tutoring Systems, IV Congresso RIBIE, Brasil (1998).
33. Ward, P.T., Mellor, S.J.: Structured development for real time systems. Prentice-Hall Internat (1985).
34. Finin, T., Fritzson, R., McKay, D., McEntire, R.: KQML as an Agent Communication Language, Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94), ACM Press, USA (1994).
35. Labrou, T., Finin, T.: A Proposal for a new KQML Specification, Technical Report TR CS-97-03, University of Maryland, USA (1997).
36. Chang, K.E., et al.: Web_Soc: A Socratic-Dialectic-Based Collaborative Tutoring System on the World Wide Web, IEEE Transactions on Education (2003 February), 69-78.

Acquiring Emotion Knowledge of Anger from WWW

Xi Yong[1,2], Cungen Cao[1]

[1]Key Lab of Intelligent Information Processing,
Institute of Computing Technology,
Chinese Academy of Sciences,
Beijing 100080, China
[2]Graduate School of the Chinese Academy of Sciences
{yongxi, cgcao}@ict.ac.cn

Abstract. Emotion knowledge is a special part of human knowledge. This paper focuses on acquiring and representing knowledge of one single emotion, i.e. anger. Our method consists of three parts. First, we extracted numerous scenarios of anger from Web pages. Second, we analyzed those scenarios, in which we considered the agent and anger-eliciting context from several perspectives. Third, we manually abstracted the scenario details away and formalized them into a knowledge representation model to obtain an anger knowledge base. We believe that this method can easily be extended for other emotions.

1. Introduction

Human emotion knowledge is a special kind of human knowledge [9], and this knowledge is very important in applications such as human-computer interaction, affective computing, natural language understanding, and psychoanalysis [11].

In this paper, we focus on one single human emotion, i.e. anger, which is one of the basic human emotions [9]; and discuss the methods of acquiring and formalizing anger scenarios from WWW.

In the psychological literature, psychologists have extensively investigated the antecedents of anger, as well as its waxing and fading. Lakoff proposed an interesting study of anger through the use of metaphors [6]. He examined a large number of metonymies for anger and classified them, which led to the extraction of high-level metaphors for anger, and finally presented a prototypic cognitive model of anger which he claims is a central model with a few variants, and upon which all the metaphors he worked out were converged. But the model is over-simplified and too abstract, and it does not account for differences in various individuals.

The model proposed by Averill is a well-known and comprehensive study on anger [1]. Averill constructed a model of anger according to a number of self-report surveys, which he used to explore various aspects of anger such as motives, responses, targets and so on. And he worked out a very comprehensive list of rules and norms associated with anger. His research offers a rich picture of anger and provides us with much inspiration.

On the basis of previous work, Chris Vogt presents a practical method for representing anger in a general way [13], and frames are adopted as the representation. Each slot of a frame described a specific aspect of an anger event, such as Person, Caused-by, Results and so on. While the frame has included all the key points of anger, it can only be used to store some discrete instances of anger when it has been identified. The identification method and the organization of these instances are still unsolved problems.

In this paper, we present a method for acquiring and representing commonsense knowledge from anger situations extracted from Web pages. Through careful semantic analysis, we have constructed a formal representation framework for anger and summarized numerous anger-eliciting patterns which can give rise to anger, according to which we can give a reasonable explanation to the production of anger in each represented situation and determine whether a given situation will make a given agent feel anger or not. This model has incorporated most of the key characteristics of anger as described by leading emotion researchers and effectively revised a number of shortfalls existing in previous work. Furthermore, it is designed in a very general way and could easily be modified to accommodate other emotions.

This paper is organized as follows: Section 2 shows our knowledge acquisition method. Section 3 introduces the underlying emotional theory behind our model and analyzes the semantic structure of anger situation. In Section 4 we describe our representation framework and explain it in more detail by some instances. Finally, section 5 concludes the paper.

2. Extracting Scenarios of Anger from WWW

The lack of commonsense knowledge about emotion has highly limited the ability of intelligent systems to interact with users more naturally and socially. It is one of the most pressing issues to build a systematic, structural and operational knowledge base that contains various formalized commonsense knowledge about emotion. And this depends on a large-scale real-world knowledge corpus about social emotion-eliciting situations.

We believe that Web pages are a particularly important source of emotion knowledge because large bulks of postings are Internet-based; and with the rapid development of the network industry today, there are countless information resources in textual form available on the Internet. This kind of knowledge can avoid the partial opinions toward emotion-eliciting situations caused by individual considerations of single researcher or small research group and the biases which people are likely to make towards stereotypical or socially desirable answers in some general acquisition methods such as questionnaire. After comprehensive analysis of all kinds of situations and reactions in people's emotional experiences, the emotional knowledge represented can become commonsense in deed.

Therefore, we firstly used a large amount of textual Web pages from WWW (about 1.2T), and then extracted text segments, in proper length and with proper keywords (e.g. angry and rage), from these pages as our initial corpus. At present the approaches of textual affect sensing commonly include keyword spotting, lexical

affinity, statistical methods and hand-crafted models [4]. Because there is such a great bulk of Web pages that we can only employ the easiest and fastest method of keyword spotting to extract the segment containing some emotional information from them. So, we adopted all the words listed in the 'anger' category in a Chinese Synonymy Thesaurus edited by J.J. Mei as our keywords, which includes most of the words and phrases we used to represent anger in everyday life in different forms such as emotional concepts (e.g. "anger", "rage", "indignation"), expressions and actions (e.g. "stamp with fury", "catch fire", "brawl") and so on. By locating these keywords in Web pages we picked them up along with relative context in proper length that we experientially assume as sixty words. These textual segments provide us with a rich corpus.

As we see, the content of the corpus is very rough and cursory, full of incomplete or irrelevant information, but they are good heuristic clues to infer real-world emotion-producing processes. We revised and complemented them after careful analysis and as a result gained 21334 anger scenarios, each of which contains a piece of commonsense knowledge about a special emotion-eliciting situation. Some of the refined pieces are shown in table 1.

Table 1. Some anger scenarios (translated from Chinese)

1	After reading the report in the newspaper, he thought it stabbed his reputation and got very angry. Then he decided to bring an accusation against the writer of the report.
2	If you reproach a man for his naive behavior or words, he must fly into a rage at once.
3	Father flew into a rage when he heard that I failed in my English test again.
4	She got so angry when he missed the date that she was unwilling to see him again since then.
5	Mother was so angry that her face was drained of blood when she knew his son had committed the serious crime.
6	He knew only I could help him to escape the punishment, so he flew into fury when I said I wouldn't help him and hit me with the club.
7	Teacher ordered me to clean the whole classroom. I was indignant because I felt that I had been punished unfairly. I want to revenge the person who maligned me to the teacher no matter whoever he/she is.
8	The atrocity of the governor caused widespread indignation and social criticisms.
9	When I was reviewing my lessons before the exam, these flies were annoying me.
10	He is such an arrogant man and we're annoyed at his cavalier treatment to his old friends.

3. An Appraisal-Based Analysis of Anger

In the literature, dominant theories of emotion are appraisal theories. They suggest that emotions are generated by appraisal of a subjectively important event by the person who is experiencing the emotion. While the original cause of one's emotion is derived from the external stimulus, the appraisal process has the special nature of subjectivity, which is due to the subject's particular goals and the like.

For example, if a person is in a situation which could drive him closer to some important goal, the emotion experienced would be one of a pleasant type, and to determine which the particular pleasant type is actually generated and what its intensity is would require more information about the situation and the person's present internal mental structure which include his/her beliefs, goals, and so on.

Reisenzein [8] views appraisal processes as components of a continuously operating mechanism that monitors the compatibility of newly acquired beliefs of the person with pre-existing beliefs and desires. The belief-congruence appraisal tests whether newly acquired beliefs are congruent or incongruent with pre-existing beliefs and the desire-congruence appraisal tests whether newly acquired beliefs are congruent or incongruent with pre-existing desires.

Although his work offers a suggestive proposal, it is too simple to explain the cause of a particular emotion generated in real-world situations. After a comprehensive survey on the emotional scenarios acquired from Web pages, we presented a more clearly actual appraisal process, which can give a rational explanation to the occurrence of anger.

The first influencing factor is irritating events. Irritating events are what may trigger anger. They are the most important components of anger-eliciting situations. But, whether anger is really triggered still depends on several other factors, as discussed below.

Second, we consider personal expectations. Expectations refer to all kinds of desires and needs of a person. When the conditions in the external context meet any of one's expectations, he/she will experience some emotion of a positive type. On the other hand, if the conditions are against with the person's expectation, which is caused by some others, then the person will generate some negative emotion such as anger.

Third, to judge whether an irritating event causes someone to feel angry depends on how he/she interprets the event. The judgment is essentially subjective, which is based on his/her beliefs that are accumulated by past experience.

Fourth, when a person highly values something, he/she would struggle to obtain or maintain it. For example, if one thinks that personal reputation or image is highly valuable, he/she will maintain a good reputation.

Fifth, anger is usually accompanies with a certain reaction. Again what the reaction is appropriate depends on how the person interprets the irritating event and how the event affects or violates his/ her expectation and/or personal values.

Sixth, anger may also depend on interpersonal relations. Being insulted by one's children may not necessarily cause one to be angry, but being insulted by an opponent tends to irritate one inevitably.

Finally, but most importantly, we consider the attribution of negative events [14]. When a loss or failure is attributable to others, an emotion of anger is likely to come into being; but when a person attributes the loss or failure to the uncontrollable nature, he/she may not be angry towards others, instead of feeling despaired.

4. A Parametric Representation of Anger

Based on previous analysis, we built a two-layered frame-based representational model of anger knowledge. The first layer defines categories of anger situations based on irritating events. This layer is parameterized in that we define relevant parameters in the categories, and these parameters will be instantiated on the second layer where more concrete anger knowledge is defined.

In each category, parameters are typed. Common types include:

- person. Agents who experience anger or make somebody angry.
- event. (Irritating) events are causes of anger.
- object. Non-personal things, such as bad news, which contribute to anger.
- reaction. The behavior that an angry person tends to take.

In our work, category parameterization is significant for knowledge reuse, and parametric categories are actually "generators" of anger knowledge.

```
defcategory insultation-caused-anger
{
    person: ?x
    person: ?y
    event: insult(?x, ?y, ?m)
    emotion: know(?y, insult(?x, ?y, ?m)) → angry(?y)
    reaction: ?z
}
```

Fig.1. A category of Anger

As illustration, we generalize the first scenario in table 1 into a category (Fig. 1) and an instance frame of anger (Fig. 2). In Fig. 1, the emotion angry(?y) depends on whether ?y knows the event of insult(?x, ?y, ?m). Knowing insult(?x, ?y, ?m) can be inferred by hearing from others or happening to read a newspapers article and the like.

In Fig. 2, we first instantiate the category of insultation-caused-anger by assigning each parameter with an actual value, and two values (i.e. person001 and person002) are further defined in the following frames of the category person.

In the instance frame of person001, we see that he has some personal beliefs and values. These are determinate factors that cause him/her to be angry when getting insulted by person002.

```
instantiate insultation-caused-anger to
{
    m=writing newpaper article
    x=person001
    y=person002
    z=accuse(y, x)
}

defperson person001
{
    gender: male
    beliefs: {insult(x, y, public)→lose(y, reputation), lose(y, reputation)→lose(y,
bussiness)}
    personal-values: {gain(y, reputation), gain(y, bussiness)}
    attribution: attribute(lose(person001, business), person002)
    interpersonal: opponents(person001, person002)
}

defperson person002
{
    belief: lose(y, reputation) → lose(y, bussiness)
    expectation: lose(person001, business)
    interpersonal: hate(person002, person001)
}
```

Fig.2. An Instance of Anger

To demonstrate the flexibility of our representation framework, we define another category of anger in Fig. 3. This category of anger is primarily caused by mistreating. But for mistreating to really provoke anger, we need to define the mistreated person and mistreating person, just as what we have done with insultation.

In Fig. 3, the emotion is simply angry(?y). This is different from that in Fig. 1, where the emotion is know(?y, insult(?x, ?y)) → angry(?y). The reason is that when a person is mistreated, he/she can generally recognize the situation of mistreating.

```
defcategory mistreatment-caused-anger
{
    person: ?x
    person: ?y
    event: mistreat(?x, ?y, ?m)
    emotion: angry(?y)
    reaction: ?z
}
```

Fig.3. Another category of Anger

5. Conclusion

Anger is one of the basic and common human emotions. In this paper, we analyzed anger in several determinate aspects, i.e. irritating events, beliefs, personal values, expectations, reactions, interpersonal relations, attributions. This analysis led to a formal representation of anger.

There may be many situations that make people angry. To obtain a reasonable complete list of the anger situations, we extracted from WWW a long list of anger episodes, and conducted a survey on the list. We argued that the WWW-based emotion knowledge acquisition is more practical and realistic than any others methods based on laboratory interviews.

We believe that the method proposed for acquiring knowledge of anger can be generalized to extract knowledge of other emotions such as joy, sadness and fear. This is on our research agenda.

Acknowledgements

This work is supported by the Natural Science Foundation (#60073017 and #60273019), and the Ministry of Science and Technology (#2001CCA03000).

References

1. Averill, J.R.: Anger and Aggression: An Essay on Emotion. Springer-Verlag (1982)
2. Berkowitz, L.: On the Formation and Regulation of Anger and Aggression. American Psychologist, Vol. 45, No. 4 (1990) 494-503
3. Boeree, C.G.: Anger: A Phenomenological Sketch. Ph. D., Shippensburg University (1998)
4. Carlson, J.G., Hatfield, E.: Psychology of Emotion. Harcourt Brace Jovanovich College Publishers (1992)
5. Kovecses, Z.: Emotion Concepts. Springer-Verlag, New York Inc. (1990)
6. Lakoff, G.: Women, Fire, and Dangerous Things: What Categories Reveal about the Mind. University of Chicago Press (1987) 380-409
7. Liu, L.G., Lieberman, H., Selker, T.: A Model of Textual Affect Sensing using Real-World Knowledge. Proceedings of the 8th International Conference on Intelligent User Interfaces (2003) 125-132
8. Reisenzein, R.: Appraisal Processes Conceptualized from a Schema-Theoretic Perspective: Contributions to a Process Analysis of Emotions. In: Appraisal processes in emotion: Theory, Methods, Research (1999)
9. Shaver, P., Schwartz, J., Kirson, D, O'Connor C.: Emotion Knowledge: Further Exploration of a Prototype Approach. Emotions in Social Psychology (2001) 26-56
10. Scherer, K.R.: Studying the Emotion-Antecedent Appraisal Process: An Expert System Approach. Cognition and Emotion(1993) 325-355
11. Tian, W., Cao, C.G.: Research on the Human Psychological Commonsense. Ph. D., Institute of Computing Technology, Chinese Academy of Sciences (2003)
12. Tian, W., Cao, C.G., Wang, H.T.: Acquisition, Representation and Analysis of Psychological Commonsense Concepts. Computer Science, China (2004)

13. Vogt, C.: Anger and Knowledge Representation. Northeastern University Press (1993)
14. Weiner, B.: An Attributional Theory of Motivation and Emotion. New York: Springer-Verlag (1986)

Preference-based Treatment of Empty Result Sets in Product Finders and Knowledge-based Recommenders

Dietmar Jannach

Institute for Business Informatics & Application Systems
University Klagenfurt, A-9020 Klagenfurt, Austria
dietmar.jannach@ifit.uni-klu.ac.at

Abstract. "Your search returned 0 results" is an undesirable message for customers using an online product-finder or a web-based sales advisory system. Such a situation typically occurs when a filter-based recommender system is used and the user's requirements are unrealistic or inconsistent. In this paper, we present two orthogonal techniques for dealing with such situations, whereby the work is based on a general model of this class of recommendation systems.
First, an algorithm for priority-based filter relaxation is presented where the end user can actively participate in the problem resolution process by specifying the importance of the predefined advisory rules that led to the empty result.
Second, we adapt an algorithm from the field of model-based diagnosis and repair in order to compute a set of action alternatives for the customer. Each of these alternatives corresponds to a possible (small) revision of the originally inconsistent requirements, such that a product proposal can be made.
The paper finally describes practical results from experiences in different real-world application domains and discusses domain-specific heuristics and techniques for further search time improvements for complex problems and multi-user environments.

Introduction

Due to the huge variety of available products and services, many companies run product finders or more intelligent systems like recommender systems or sales advisory systems on their corporate web-sites. Based on the users' inputs, these systems filter out those products that fulfil the given specifications or match the needs and preferences of the customer. In particular for technical domains, where recommendations are not based on the concepts of "quality" and "taste", knowledge-based recommender systems have their specific advantages compared to other prominent approaches based on, e.g., collaborative filtering. Once the knowledge of the expert is made explicit and encoded in a knowledge base, the recommender system will behave like an experienced sales assistant and the quality of the recommendation will be constantly high, even if there are new products or new users involved. Even more, when adopting a knowledge-based approach, such systems will also be able to "explain" their recommendations to the customer. One of the main criticisms of such filter-based approaches [1] is that the undesirable situation can arise

where all products are filtered out and no proposal can be made. "Your search returned 0 results" is the only response of many online systems in such situations. An experienced sales assistant, however, would explain to his customer why there are no results, i.e., which advisory guidelines and rules he did obey. Even more, he could inform the customer what he can do about the situation, for instance reconsidering the potentially conflicting requirements, or he could even propose alternative solutions that satisfy most of the given customer requirements.

In this paper, we present two orthogonal techniques for dealing with such situations. First, we show how filter-relaxation based on priorities can be exploited in order to take the different importance levels of the given advisory guidelines into account and help the user in understanding the proposals. Second, we describe an algorithm for computing a suitable set of alternatives that tries to minimize the differences between the original requirements and other slightly changed requirements such that a solution is possible and respects the strict time restrictions of online recommendation systems.

Example and definitions

For demonstration purposes, we shall use a small example from the domain of digital cameras, a typical problem area where hundreds of different products are available and where online sales advisory systems are already used to assist the customer in finding the right product. We use the following very general model of a filter-based recommender system. Each *product* in the knowledge base is characterized by a set of attributes, whereby in many domains also set-valued attributes must be supported. Next, there is a set of *variables* that correspond to the user preferences, e.g., direct inputs or indirectly computed characteristics. Finally, a set of *filtering rules* describes the relation between the customer characteristics and the fitting product properties.

Our digital cameras are described by the property set *P*, whereby P = {weight, price, interfaces} and the following products exist in the product knowledge-base

PKB = {
{id(p1). weight(100). price(80). interface(usb)}
{id(p2). weight(100). price(150). interfaces(usb). interfaces(firewire).}
{id(p3). weight(200). price(200). interfaces(usb). interfaces(firewire). interfaces(dockingstation).}
}

During the advisory session the customer can specify his preferences by assigning values to the variables from the set *V* that contains *pref_weight (*with the domain "*low, medium, irrelevant")*, *pref_class (cheap, middle, high, premium, irrelevant)* and *pref_interfaces (standard, advanced)*[1]. We represent the actual customer requirements in a set *REQ* of positive ground literals that use the predicate symbols from *V,* e.g.,

REQ = {pref_class(premium). pref_weight(low). pref_interfaces(advanced).}

[1] Note that in many technical domains it is advantageous to question the customer about his preferences and not directly about product properties, in particular when the knowledge level of the customers can be low [2].

The expert's filter rules $FR = \{f1, f2, f3\}$ determining the contents of the result set RS are as follows, whereby the trivial axiom $RS \subseteq PKB$ has to hold.

f1: $pref\_weight(low) \in REQ \Rightarrow \forall X,P: P \in RS: weight(X) \in P \wedge X < 150.$
f2: $pref\_interfaces(advanced) \in REQ \Rightarrow \forall P: P \in RS: interface(firewire) \in P.$
f3: $pref\_class(premium) \in REQ \vee pref\_class(high) \in REQ$
$\Rightarrow \forall X,P: P \in RS: interfaces(dockingstation) \in P \wedge (price(X) \in P \wedge X > 180).$

Given that the customer wants a premium class camera with an advanced set of interfaces and a model that also has a low weight, the application of the rules will result in no suitable product. One approach to deal with the problem is to assign priorities to each filter rule in advance and to iteratively retract the rules until a sufficient number of products remains. Retracting the weight-rule *f1* results in product *p3*, retracting *f2* alone will not help, and retracting *f3* leads to product *p2*. If we assume that the domain expert annotates the filter rules with initial priorities $f3 > f1 > f2$, because from experience he knows that the Firewire – requirement (*f2*) is not that important for most customers, the system will initially come up with solution *p3*, i.e., the rules *f2* and *f1* will be relaxed. If we assume that for each rule an explanatory text is maintained both for the case that the rule is applied and for the case it is relaxed, the system could explain:

- (f3 - positive): Based on your preferences, I propose the premium or high class model "p3" that also ships with a convenient docking station.
- (f1 - negative): Due to your other requirements, I included a camera in the proposal that does not fulfill your requirements with respect to a low weight.

Note that although filter *f2* was retracted, the conditions of the filter are fulfilled for product *p3*. As such, we can include the positive explanation for *f2*.

- (f2 - positive): This camera fulfils your requirements on advanced interfaces.

After this initial proposal and the explanation, the customer can be given the possibility to dynamically change the priorities of the rules, if they differ from the domain expert's predefined priorities and trigger the computation of another result.

Nonetheless, there are domains where there are strict rules that should never be relaxed or situations where the customer interactively states that he explicitly stipulates the application of specific rules. Consequently, a situation with an empty result set still can arise. In this case, it would be helpful for the customer if the system provides a set of alternatives of slight changes in the requirements such that a product can be recommended. If we assume that all filter rules in our example are strict ones, some *good* options for the customer could be as follows.

- (1) If you change your weight requirement from "low" to "middle", I can propose the following products: p3.
- (2) If you change your class requirement from "premium" to "middle", I can propose the following products: p2.

Theoretically, there are lots of other alternatives that differ from the previous ones in their "quality". For instance, all repair-alternatives that contain (1), as well as additional changes in the requirements can be seen as suboptimal. Furthermore,

alternatives where requirements are completely taken back, i.e., changes where the removal of a predicate from *REQ* leads to a solution, are also not optimal.

After the description of the algorithm for preference-based relaxation in the next section, we will describe a technique for efficient computation of suitable repair alternatives in cases where no result exists.

We will use the following general definition of knowledge-based recommendation problems for the subsequent algorithms[2].

***Definition**: A Knowledge-based Recommendation Problem (KBRP) is a tuple <P, V, PKB, FR, REQ>, where P and V are sets of predicate symbols that are used for describing product properties and customer requirements. PKB represents the available products and is a set of sets of positive ground literals using the predicate symbols from P. FR is a set of logical sentences with the structure of "filter rules". REQ is a set of positive ground literals using the symbols from V and correspond to actual customer requirements.*
A"filter rule" is a logical sentence in form of an implication, where the antecedent is a logical expression where predicate symbols from V are allowed, and where the consequent describes restrictions on elements of PBK by the usage of predicate symbols from P.

***Definition**: Given a Knowledge-based Recommendation Problem KBRP<P, V, PKB, FR, REQ>, KBRP is a called a "Valid KBRP" if there exists a set REQ (including the empty set) such that there exists a subset RS of PKB where FR ∪ REQ ∪ RS is satisfiable.*

Generally, a *knowledge-based recommender* in this implementation independent problem formulation is a software system capable of computing a valid recommendation *RS*, given the parameters *PKB, FR,* and *REQ*, or reporting failure of finding a solution.

Priority-based relaxation

The algorithm for priority-based relaxation is straight-forward and from the basic idea similar to the Hierarchical Constraint Satisfaction approach [3]. We extend the KBRP*<P, V, PKB, FR, REQ>* with a function Φ that relates each filter rule from *FR* with a priority value, i.e., a positive integer, whereby higher values stand for lower priorities and zero means that a rule should not be relaxed. The algorithm takes KBRP, Φ and a search limit as input and returns a set of products that remain after a successful relaxation process or otherwise an empty set. In addition, for each product in the result set, the sets of applied and removed filter rules for the explanation

[2] We use first order logic as a representation language in order to facilitate a clear and precise presentation.
In this context, we use the more general term "knowledge-based recommendation problem"; in the literature, the terms "filter-based recommendation" and "filter-based product retrieval" are also used to describe this class of systems.

process is constructed, whereby we include those filter rules in the set of applied filters that were originally removed due to a higher priority, but would hold for a given product, see filter rule *f2* in the example section.

Algorithm RELAX(*PKB, FR, REQ,* Φ, limit)

```
result = ∅                                  // initialize the result
allrelaxed = ∅                              // remember everything we relaxed
relaxable = subset of elements f of FR where Φ(f) > 0
while (| result | < limit ∧ | relaxable | > 0)   // as long as there are relaxable filters
                                            // and the limit is not reached.
   priority = highest value of Φ(f), where f ∈ relaxable
                                            // determine the set of filters to remove.
   relaxset = subset of relaxable, were Φ(f) = priority
   relaxable = relaxable \ relaxset         // remove the set from the original set.
   allrelaxed = allrelaxed ∪ relaxset       // remember the removal
   result = filter(PKB, relaxable , REQ);   // call the recommender/product finder.
end while
// compute the correct explanation sets
// variable "explanations" contains triples of products, applied, and relaxed filters
explanations = computeExplanations(relaxable, allrelaxed, result)
return result;
```

Algorithm 1. Priority-based relaxation

Algorithm COMPUTEEXPLANATIONS(applied, relaxed, result)

```
// The algorithm computes a set of positive and negative explanations
// for each product in the result set by testing each filter in the applied
// and relaxed sets.
explanations = ∅;
for each p ∈ result
   // prepare the real sets for the explanations
   pos_arguments = applied
   neg_arguments = relaxed
   for each f ∈ relaxed                     // test all relaxed individually
      rs = filter(p, f , REQ)               // call the recommender, check if the
                                            // current product fulfils the filter rule
      if (| result | > 0)                   // if filter consistent for that product
         pos_arguments = pos_arguments ∪ f  // update the explanation sets, e.g.,
         neg_arguments = neg_arguments \ f  //  filter f2 in the example
   end for
   // add the new explanation tuple.
   explanations = explanations ∪ <p, pos_arguments, neg_arguments >
end for
return explanations
```

Algorithm 2. Computing adequate explanations

With regard to complexity of the algorithm, the computation of the result set without the explanations takes at most as many iterations as there are different values in the range of Φ. For each element in the result set, the number of iterations for the explanations is |*result*| * |*relaxed*|. Note that in a practical setting we will not pre-compute all explanations in advance but rather on demand, when a user asks for the explanation for a specific product.

Computing repair alternatives

In domains where some of the rules are strict and cannot be relaxed definitely, a situation with an empty proposal can still arise. Then, it would be helpful if the system can propose the customer a set of "repair" actions which corresponds to slight changes and revisions of the initial requirements. In principle, two basic approaches are possible. First, the underlying recommendation knowledge base can be extended with additional domain knowledge, i.e., a set of explicitly modelled repair rules like shown in [4] for the domain of product configuration and reconfiguration. Such approaches, however, are costly in terms of knowledge acquisition and in particular maintenance as for each change in the knowledge base also the repair rules have to be checked and possibly adapted. Even more, such knowledge bases tend to get complex because of the strong interdependencies between the rules. On the other hand, model-based (diagnosis) approaches like, e.g., [5] or [6], that try to minimize the amount of additional needed domain knowledge for the repair task, face the problem of large search spaces for the computation of possible repair alternatives. Applied to real-world problem settings, such systems rely on domain-specific heuristics for discriminating between the alternatives or different forms of (structural) abstractions in order to produce adequate results within limited resource bounds.

If we follow such a model-based approach, the search complexity in our domain is determined by the size of the domain of the variables (i.e., the user inputs) and the property, whether these variables can be multi-valued or not. For each single-valued variable with a domain-size of n, we have $n+1$ possible assignments when we include a special *null*-value with the meaning that the user did not specify a requirement for a variable. For each multi-valued variable, there are 2^n possible answer combinations. If we assume that there are three single-valued and three multi-valued variables involved in the remaining strict rules and we have an average number of four possible answers, there are theoretically more than 60.000 possible answer combinations. Obviously, an exhaustive search for those combinations is not feasible, given the tight time limits of an online recommendation system[3].

In the following, we present algorithms and techniques for efficient computation of a suitable set of alternatives where we use domain-independent heuristics as well as application-dependent variants in order to reduce the required search times for practical settings.

Diagnosing the requirements. Depending on the application domain it can be sufficient that the system presents the user a list of requirements that should be

[3] An offline pre-computation for *all* possible input combinations is not possible, given the vast search space for, e.g., twenty to twenty-five questions in a realistic scenario.

completely retracted for a possible solution, e.g., "*If you skip your requirements on the weight, I can propose the following solution...*" In such cases, the computation of alternatives can be performed by using the standard *Hitting-Set* algorithm that is commonly used in the field of model-based diagnosis ([7], [8]). The algorithm is used in a way that the nodes of the Hitting-Set DAG[4] are labelled with *conflicts* which are in our case subsets of user requirements that cause the result set to be empty. If there is no conflict generation support, it will be the union of all variables that are used in the non-relaxable filters that have to be applied in the current situation.

Definition. *Given a Knowledge-based Recommendation Problem KBRP<P, V, PKB, FR, REQ>, a "conflict" for KBRP is a set C ⊆ REQ such that there is no set RS ⊆ PKB for which FR ∪ RS ∪ C is satisfiable.*

A conflict is "minimal", if there is no proper subset C' of C such that C' is also a conflict for KBRP<P, V, PKB, FR, REQ>.

The outgoing edges of the DAG are labelled with variable names from the nodes which are retracted in the current search phase. The main advantage of this approach is that the diagnoses are computed in ascending order with respect to their cardinality, which is reasonable because we assume that alternatives with fewer changes in the requirements are preferred by the users. Even more, the tree pruning techniques from [7] can be exploited to reduce the search space, as we are typically only interested in *minimal diagnoses*. As an example, if we found that omitting the *weight* requirement results in a solution, we can remove branches in the search space that involve the *weight* requirement and any additional requirement.

As a simple heuristic which helps us to find a first solution more quickly is to try those variables first which are involved in more non-relaxable filter rules.

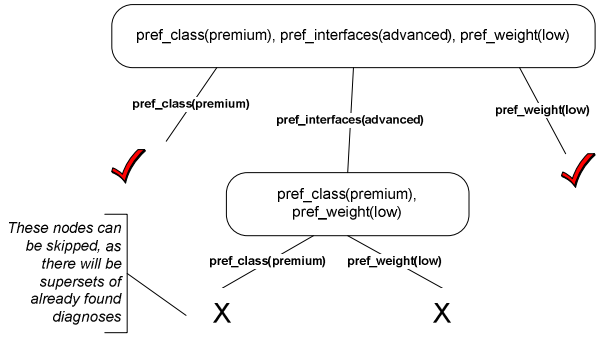


Fig. 1. A simple example for HS-DAG construction

During the breadth-first HS-DAG construction (Fig. 1), each call to the *Theorem Prover (TP)* [7] corresponds to a search for products performed by the underlying knowledge-based recommender, where we leave out all requirements that are on the path from the root node of the DAG to the current node. If this search results in a

[4] Directed acyclic graph.

solution, we can close this node n and add $H(n)$[5] [7] to the set of diagnoses. Note that for each TP-call all the other current user requirements that are not involved in the current set of non-relaxable filters have also to be taken into account. Therefore, in order to guarantee that the algorithm will always find at least one solution, we make the assumption that there are no filter rules in the knowledge base whose antecedent defines a condition on the *non-existence* of a requirement, like

$$\forall X: pref\_x(...) \notin REQ \Rightarrow \ldots.$$

If this – for many domains realistic assumption – holds, we can guarantee that by retracting user requirements no additional filter rules will get active during the search for solutions and we can thus safely prune the search tree.

Optimizations.

1. In general, we cannot assume that the underlying recommender system is capable of generating (minimal) *conflict sets*, such that we start with one single conflict that includes all problematic, i.e., conflicting user requirements. Nonetheless, during the HS-DAG construction, we remove elements from this conflict and check if a solution exists. If we encounter a situation where the removal of one input does not lead to a solution, we know that the remaining user inputs alone cause a conflict. As an online recommendation system will be used by many customers and subsequent customers may have a similar set of requirements, we can cache these already minimized conflicts and reuse them in the next session where a similar repair is needed. A safe reuse of such a conflict is possible if the cached conflict is a subset of the requirements of the next customer (compare, e.g., to [9]).
2. Because we allow disjunctions of predicates in the filters' pre-conditions, it can be the case that there are unassigned variables in the union set of the variables of the non-relaxable filters. Therefore, these variables can be removed from the initial *conflict* – which consequently reduces the search space – such that these variables do not have to be considered during the HS-DAG construction process.

Searching for alternative solutions. In some application domains, just presenting a set of requirements to be removed might not be satisfying for the customer. Even more, when individual requirements are fully removed, the customer might get a proposal that is not "near" to his original preferences. Fully removing a "low price" preference, for instance, could result in the proposal of premium priced products if no other filter rules constrain the price limit. Consequently, it would be preferable if the user can choose among several variants and interactively decide which of his requirements he is willing to give up or relax. In order to cope with such situations, the following value-based variant of the first approach can be used (see Fig. 2). Similar to the first approach, we proceed in a breadth-first approach. However, upon creation of a new node, all alternative settings for the variable under examination are tested individually except for the conflicting assignment. In cases, where more than one variable is tested (e.g., at the second level), we theoretically check all possible combinations of these variables, i.e., the Cartesian product of the possible values.[6]

[5] $H(n)$ equals the set of all edge-labels from the root of the DAG to n.

[6] At the moment, the value-level approach handles user requirements with finite domains of enumeration values. For free-input requirements like a price limit we reduce the search space to respecting the requirement or not but do not search for alternative values.

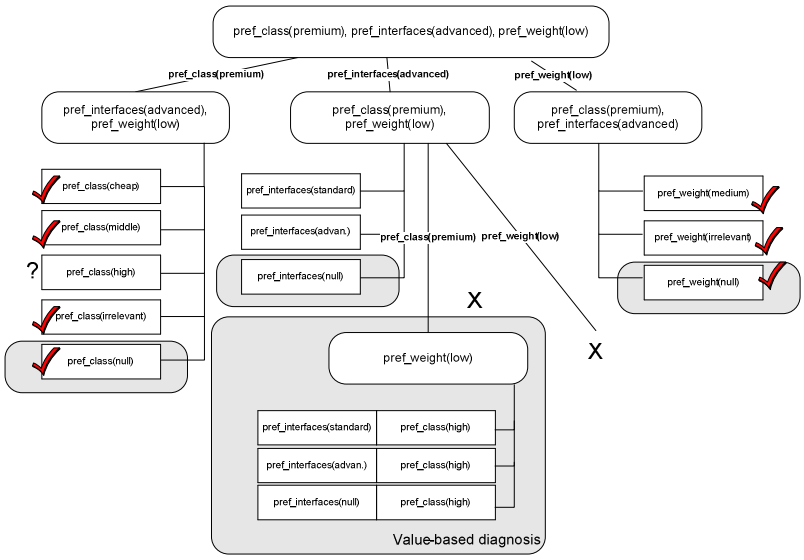


Fig. 2. Value-based algorithm

Note that in the value-based approach, we add a special "null" value (marked grey in Fig. 2) to the domain of each variable as the full removal of a requirement could be a possible option in some domains.

The main advantage of the breadth-first approach is that alternative solutions with fewer changes are found first. In practice, proposing alternatives with more than three or four variables changed does not help the users too much anyway, as the differences to the original preferences are not satisfying for the users. Therefore, in practical situations, the search is stopped when the first few alternatives are found.

When using the value-based approach, pruning has also to be performed on the value level. As we can see in Fig. 2, the node with the path (*pref_interfaces*, *pref_class*) cannot be immediately closed like in the first algorithm. Although we already found several solutions by changing the value of *pref_class* alone, there was no solution when we tried the value "*high*" (indicated by the question mark). Nonetheless, as we make no restrictions on the filter rules except for those on the structure described in the definitions, it could be the case that there exists a combination of *pref_class(high)* and some value of *pref_interfaces* such that a solution with changes in both variables is possible. However, note that we only have to check the combinations with *pref_class(high)* because combinations with other values of *pref_class* would result in a superset of an already found diagnosis. On the other hand, nodes that would involve *pref_weight* can be immediately closed since all alternative assignments to *pref_weight* result in a suboptimal solution.

Nonetheless, additional pruning approaches can be applied to reduce the search space, but come at the cost of loss of solutions. First, we could omit such cases as described in the last paragraph and prematurely prune the node with the path (*pref_interfaces*, *pref_class*), when we have domain-specific knowledge about the

filter constraints. In addition, for each node we can stop examining the possible value combinations, once we found a first solution, e.g., after trying the "cheap" value for *pref_class* we do not check the other alternatives. Such an approach can be useful in cases where we are only interested repair alternatives that involve single changes.

In general, the diagnosis algorithm will always find at least one solution given a valid recommendation problem, as the search space is exhaustively searched and no minimal solutions get lost by the value-based pruning techniques.

Discriminating between diagnoses. In cases when there are lots of solutions, i.e., repair alternatives returned by the search alternatives, these proposals should be ranked such that the customer can choose the alternative that matches his preferences best. A quite intuitive ranking will be based on the number of changes that are required in the requirements. Depending on the domain, other application-specific orderings can be used in order to rank the alternatives with the same number of changes. First, we can prefer those alternatives that maximize the number of products that can be proposed if the changes are applied. Another ordering can take the differences to the original requirements on the value level into account. In many domains, the possible values for a specific variable have an implicit ordering, like a price preference (low, medium, high etc.). As a consequence, from a practical perspective, it can even be better to promote an alternative where two of the requirements are changed to a neighbouring value, than to propose a single-change alternative where a customer's preference has to be inverted, e.g., changed from "yes" to "no". In principle, an initial "cost model" for changes can be computed without further domain knowledge, if we assume that the possible values for a variable are defined in such an implicit order. Before the results are presented to the user, the system can rank the alternatives based on these costs which are determined by the *distance* to the original requirement and the overall number of the possible values[7]. For multi-valued variables, changes where the original requirement statement and some more or fewer values lead to a solution can be for instance preferred over others that have complete different values.

If desired, such a model of "change costs" can also be explicitly defined in the knowledge base. The definition of such a model however can require significant knowledge acquisition efforts as these cost functions have to be defined manually for each variable. Nonetheless, compared with approaches where "repair rules" are explicitly defined, the definition of cost functions has the advantage that changes in the model only have local effects and the overall consistency of the knowledge base is not affected.

Finally, when such a cost model exists, it can be exploited during the search phase in settings where we perform an incomplete search and stop at a certain threshold, e.g., when a defined number of alternatives is found. As an example, we could stop examining other possible values for a price preference, when we found that a change from "low" to "medium" results in a solution, as we know that increasing the price limit further will only result in a suboptimal alternative with respect to the costs, i.e., the quality of the repair alternative.

[7] The number of possible values is important, because the *distance* from "yes" to "no" is small, but there are no other alternatives, so the *cost* estimate should be correspondingly high.

Experimental results

The described algorithms were implemented as add-on to the ADVISOR SUITE [10] framework, a research toolkit for rapid development of personalized online sales advisory systems. For the evaluation process, knowledge bases from several real-world problems from different application domains were tested. The application domains range from complex areas like investment alternatives to technical items like digital cameras or skis, up to "quality and taste" domains like wine or cigars.

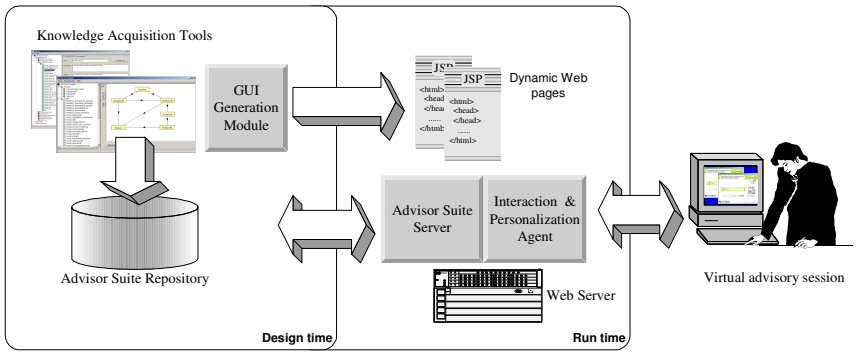


Fig. 3. Overall architecture of ADVISOR SUITE

Figure 3 depicts the overall architecture of the ADVISOR SUITE framework. The required knowledge for recommendation and personalization is captured using graphical knowledge acquisition tools and stored in a knowledge base which is built on top of a relational database system. In the Java-based ADVISOR SUITE SERVER module, the core recommendation logic is implemented; the individual advisory sessions are managed by the INTERACTION AND PERSONALIZATION AGENT. Short response times and high overall performance are key issues for such online Web-based systems. In general, the knowledge-based ADVISOR SUITE framework thus extensively caches and pre-loads the contents of the knowledge base and pre-compiles the recommendation rules into a compact internal format, such that a fast rule-evaluation process is possible.

Problem size. In our real-world advisory problems, the knowledge-base typically comprises twenty to thirty questions (variables) whereby – based on a personalization process – not every user is asked every question [2]. The number of filtering rules that determine the product selection mostly remained manageable and ranges from twenty to sixty expert rules[8]. The number of available products varies with the domain, from a few dozens when "services" are recommended, to several hundred when the domain is investment advisory or digital cameras.

Priority-based relaxation. The computation of explanations in terms of applied and relaxed advisory rules (Algorithm 1) is not problematic, as the search complexity

[8] The overall knowledge base is more complex, as it contains additional knowledge for the personalization of the dialogue flow, the adaptive user interface, as well as for the personalized ordering of the results records.

is linear with the number of priority levels, typically not more then twenty or thirty. Each check whether a sufficient number of results will be available upon relaxation of the filter rules on a certain level, can be performed within 10 to 20 milliseconds on a standard PC and database system depending on the number of available products, the overall response times for all tested cases were below one second. From the end-user perspective, however, this feature was highly appreciated by users in all domains where such an advisory application was implemented. First of all, the existence of a personalized explanation significantly increases the customers' confidence in the proposal, i.e., when reading the natural language explanations of the applied advisory rules and the justification for the products the user implicitly "learns" things about the domain. Furthermore, the possibility to stipulate the application of individual rules or decrease the priority of a rule lets the user express his real preferences in an intuitive and comprehensible way, in cases where the priorities which are predefined by the domain expert do not match his personal interests. Nonetheless, we also found that some of the users were overwhelmed when they are asked to assign relative priorities to the rules, such that in most cases we followed an approach where the only possibilities are to force the application of a rule or completely ignore it and to undo these choices.

Search for alternatives. The search for possible alternatives is computationally complex and has to be fine-tuned for each application domain. In typical applications, the list of explanations of the expert rules leading to the empty result set is presented to the user, followed by the list of repair alternatives. Depending on the domain, the complexity of the problem, and also the skill-level of the users, we have to tune the parameters of the diagnosis process: In some cases, only single-step or two-step repairs are comprehensible for the users; in other domains users want to have a choice of many different and possibly complex alternatives. In most cases, the maximum cardinality of the desired repairs is three or four as repairs that involve more changes are hard to comprehend and asses for the end user. If for instance five or more of the originally fifteen requirements have to be changed, the customer's confidence in the proposal might be low, because the product that is proposed after the change does not match a large part of his preferences.In our test cases, the computation of diagnoses up to cardinality three with single-valued attributes showed acceptable response times below one second for the computation of all possible repair alternatives without any further optimization, even in cases where nearly all of the customer requirements were involved in a conflict. For the computation of diagnoses with higher cardinality, we introduced several optimizations described as follows, such that the response times stay within the acceptable limit of two or three seconds for the hardest real-world problems with cardinality five.

1. *Result caching*. For each node in the search tree, we have to check whether changing a value results in a solution or not which corresponds to a call to the recommender system. In the ADVISOR SUITE framework, the results of previous sessions (together with explanations and repairs) are compactly stored in memory and on disk such that the system constantly "learns" new relations between

requirements and solution[9]. Our experiences show that only a fragment of the theoretically possible input combinations actually occur and many users have same or similar requirements such that the space requirements for this cache are limited. A typical example for such a "pattern" in user requirements is that customers who specify a high price-limit also have a high preference for brand products. We encode such previous results (and also those that origin during the diagnosis process) compactly, such that the check for a solution for an already known combination is then less then one millisecond. Finally, we also cache the possible repair alternatives which can subsequently be accessed without reasoning, when the same set of requirements occurs a second time. The experiences of an application with about fifteen thousand online advisory sessions in the first week of deployment shows that a high "cache-hit" rate can be achieved that justifies the limited overhead of managing such a cache.

2. *Conflicts re-using*. The same principle of such a system-wide cache can also be applied for conflicts, as in particular the computation of minimal conflicts can be a time-consuming task.
3. *Domain-dependent heuristics*. Finally, domain-dependent heuristics can help us in reducing the search space, in particular for multi-valued requirements that significantly enlarge the size of the search space. In many domains, we know for each multi-valued attribute, whether more values in such a requirement reduce the set of possible products or broaden this set, e.g., when the requirements have an implicit "one-of" semantics. Therefore, depending on the semantics, we can limit the search to alternatives with more or fewer values in the requirements and prune out all other constellations, which will not result in good solutions.

In general, the diagnostic approach for the search of repair alternatives has to be more parameterized and fine-tuned for a specific application than the relaxation technique. In particular, special attention has to be paid that the underlying complexity is hidden from the user and the user is not overwhelmed by a large set of complex alternatives. From the user interface perspective it is therefore important that we for instance aggregate the alternatives on the same variable set (e.g., "change the weight requirement to *middle* or *irrelevant*") and give the user the chance to accept and apply the alternative in a single interaction.

Conclusions

We have presented two techniques for dealing with empty result sets in knowledge-based recommender systems, whereby we based our work on a general model of this class of recommender systems that in practice base the search on product filtering.

The technique described for finding alternative requirements follows the tradition of approaches described in ([6],[8],[11], or [15]), where similar algorithms were used for computing action repair actions both for the domains classical model-based

[9] The pre-computation of all possible user input combinations and corresponding results is not feasible both with respect to time and space requirements. The cache has to be invalidated upon the periodic changes in the knowledge base, e.g., when new filter rules are introduced.

diagnosis (e.g., electronic circuits) as well as for the domains of re-configuration and debugging of software and knowledge-bases. The algorithms in this paper are designed in a way that they can be implemented non-intrusively as add-on to existing advisory systems and product finders. The only requirements are that filter rules can be selectively removed and added to the knowledge base (priority-based relaxation) and user inputs can be removed or changed dynamically (search for repair alternatives). The underlying implementation of the product finder has not to be changed or known for the application of the described techniques.

From the end user perspective, the usage of one or the other of the techniques adds a preference aspect ([12], [13]) to the advisory system, where the personalization of proposals is not only limited to preferences expressed on initial user requirements; the end user is also given a certain degree of freedom in his choice of repairs (compromises in the conflicting requirements) and the prioritization of advisory rules. The experiences show that the acceptance of the system and the confidence in the proposal increase, when the end user is able to understand and manipulate the results of the advisory process in these ways.

The presented work also strongly corresponds with the field of "Cooperative Answering" [18] in database systems, where the problem exists, that direct answers to database queries like "yes" or "no" are not always the best answers, i.e., an intelligent system would e.g. allow for the usage of "soft constraints" or preferences and cooperatively provide extra or alternative information. In the work of [19], for instance, automated analysis of the individual parts of a failing query is proposed, such that the system can provide an explanation which parts of the query caused the failure. While the approach in [19] involves high costs for identifying such sub-queries, this division in sub-queries in our application domain is indirectly given by the filter constraints themselves. In addition, our approach allows us to define natural-language explanations for failed sub-queries as well as interactive, preference-based, and non-technical selection of sub-queries to be applied.

Our future work includes the analysis whether similar techniques can be applied to the class of recommendation systems based on Case-based Reasoning ([16], [17]) in particular when the *case base* has to be updated after changes of the recommendation rules or when new products are available. Vice-versa, it will be analyzed if techniques from the CBR field can be adopted for filter-based recommender systems. In addition, we will further evaluate successful approaches from the areas of configuration and constraint satisfaction; in particular techniques based on *local search* [14] whose principle of iterative solution improvement matches the requirements of our application domain. Finally, further work will be spent on personalizing and improving the quality the resulting explanations such that they contain no spurious elements (see, e.g., [20]) or are adapted according to the current user's skills or preferences.

References

[1] D. Bridge. Product recommendation systems: A new direction. In R. Weber and C. Wangenheim, editors, Procs. of the Workshop Programme at the Fourth International Conference on Case-Based Reasoning, p. 79-86, 2001.

[2] L. Ardissono, A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, and R. Schäfer. A framework for the development of personalized, distributed web-based configuration systems. AI Magazine, 24(3):97-110, 2003.

[3] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: Hard and easy problems. In International Joint Conference on Artificial Intelligence, p. 631-639, Montreal, Canada, 1995.

[4] T. Männistö, T. Soininen, J. Tiihonen, and R. Sulonen. Framework and Conceptual Model for Reconfiguration. In Configuration Papers from the AAAI Workshop, AAAI Technical Report WS-99-05. AAAI Press, 1999, p. 59-64.

[5] S. Srinivas and E. Horvitz, Exploiting System Hierarchy to Compute Repair Plans in Probabilistic Model-Based Diagnosis, In: Proceedings of Eleventh Conference on Uncertainty in Artificial Intelligence, Montreal, 1995, Morgan Kaufmann, p. 523-531.

[6] G. Friedrich, G. Gottlob, and W. Nejdl: Formalizing the Repair Process - Extended Report. Annals of Mathematics and Artificial Intelligence, Vol. 11(1-4): 187-201 (1994)

[7] R. Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32(1), Elsevier, 1987, p. 57-95.

[8] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner. Consistency-based diagnosis of configuration knowledge bases, Artificial Intelligence, 152(2), p. 213-234, 2004.

[9] R. Greiner, B.A. Smith, R.W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence, 41(1), Elsevier, 1989, p. 79-88.

[10] D. Jannach and G. Kreutler, Building on-line sales assistance systems with ADVISOR SUITE, In Proceedings: 16th Intl. Conference on Software Engineering and Knowledge Engineering (SEKE'04), Banff, CAN, 2004.

[11] M. Stumptner and F. Wotawa, Reconfiguration using Model-based Diagnosis, in: Proceedings of the International Workshop on Diagnosis (DX99), June 1999.

[12] U. Junker, Preference-Based Search for Scheduling. In Proceedings: AAAI/IAAI, Austin, TX, USA, 2000. p. 904-909.

[13] U. Junker, Preference programming for configuration, In Proceedings IJCAI'01 – Workshop on Configuration, Seattle, 2001.

[14] R. Sosic and J. Gu. Efficient Local Search with Conflict Minimization: A Case Study of the N-Queens Problem. IEEE Transactions on Knowledge and Data Engineering, Vol. 6, 5, p. 661-668, Oct 1994.

[15] L. Console, G. Friedrich, D. T. Dupré: Model-Based Diagnosis Meets Error Diagnosis in Logic Programs. IJCAI 1993, Chambéry, France, p. 1494-1501.

[16] R. Burke, The Wasabi Personal Shopper: A Case-Based Recommender System, In Proceedings: AAAI/IAAI, Orlando, Florida, 1999, p. 844-849.

[17] R. Burke, Knowledge-based Recommender Systems. In A. Kent (ed.), Encyclopedia of Library and Information Systems. Vol. 69, Supplement 32. Marcel Dekker, 2000.

[18] T. Gaasterland, P. Godfrey, and J. Mincker, An overview of Cooperative Answering, Journal of Intelligent Information Systems Vol. 1(2), pp. 123-157, Kluwer, 1992.

[19] J.M. Janas. On the Feasibility of Informative Answers. In: Gallaire et al., Advances in in Database Theory, Vol. 1. Plenum Press, 1981

[20] G. Friedrich, Elimination of spurious explanations, Proc. of 16th European Conference on Artificial Intelligence, Valencia, Spain, 2004.

Query Plan Distribution in a Mediator Environment

Jonathan Gelati

Dipartimento di Ingegneria dell'Informazione
Università di Modena e Reggio Emilia
Via Vignolese 905, 41100 Modena, Italy
jonathan.gelati@unmore.it

Abstract. Mediator systems provide an integrated view over a set of distributed and heterogeneous data sources. They have typically two main functionalities: first they allow to build an integrated representation (or *Global Virtual View*) of the heterogeneous data and secondly allow users to pose queries over it. In this paper we tackle the issues related to the creation and actual execution of a query plan. We discuss how the control and the execution of a query plan can be distributed using the abstraction of software agents and multi-agent systems.

1 Introduction

Mediator systems are meant to provide an homogeneous interface to access a variety of data sources, hiding the lower level complexity to the users and managing the conflicts that could arise while manipulating data coming from heterogeneous sources. [1, 8, 3, 2] are examples of such systems.

The task of a mediator system is usually split into two main functions: the first is the integration of the schema related to the distributed, heterogeneous data sources in order to obtain what we call an integrated *Global Virtual View* (GVV) of the underlying data, the second is the execution of queries posed over the GVV.

We will focus on the second task, when the system has to build a plan and execute distributed queries (i.e. queries that consider data belonging to different sources). We will describe the steps that bring to the definition of the data structures needed for the query solving phase and how query plan can be subsequently implemented in a distributed way.

The paper is organised as follows. In Section 2 we present the techniques we used in the *MIKS* system to integrate heterogeneous schema. In Section 3, we show how, given a query posed by a user, a query plan can be built, starting from the mappings obtained during the integration process. In Section 4 we describe how the control and the execution of a query plan can be distributed. In Section 5 we discuss related work. Finally, Section 6 presents some final observations.

2 Integrating heterogeneous schemas

Information integration of data coming from distributed, heterogeneous data sources takes into consideration two kinds of knowledge: the intensional one and the extensional one. For clarity sake, the following example will be used throughout the paper. We assume we have to integrate the schemas of three data sources (Figure 1).

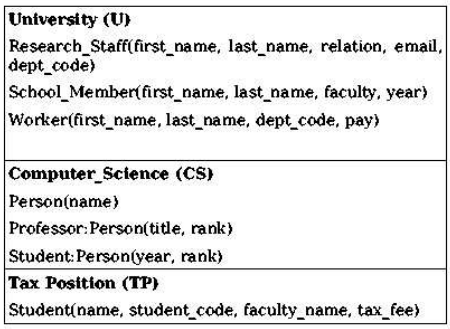


Fig. 1. The schema of the data sources of our running example

The first data source is a relational database comprising of six relations concerning the people working at a university, the department they work in and the rooms they use. The second one is an object database concerning the people who works in the Computer Science department of the university. The third data source is file-based and stores administrative information about the students which apply to the university.

We assume also the following extensional rules (their meaning and relevance is discussed in Section 2.2) holding among the relations of the schemas:

1. U.School_Member SYN_{ext} TP.Student
2. CS.Student NT_{ext} U.School_Member
3. U.Research_Staff NT_{ext} U.Worker
4. CS.Professor NT_{ext} U.Research_Staff
5. CS.Professor $DISJ_{ext}$ U.School_Member
6. U.Research_Staff $DISJ_{ext}$ TP.Student
7. U.Research_Staff $DISJ_{ext}$ CS.Student

2.1 Intensional integration

The intensional integration aims at resolving the conflicts that may arise at schema level when the same concept of the application domain is represented using different terms or structures in different data sources. For example, there

may be two relations on two distinct databases that model the same real world concept and thus have to be considered linked by a synonym relationship. Relationships between two relations or two attributes may be of four types: narrow term, broader term, related term and synonym (for a discussion of the meaning and how they apply to inter-schema and intra-schema integration see [2]). Once the relationships have been stated, classes with a certain degree of affinity (see [2]) are clustered together to produce one global class. The process results in a data structure, which we call *Mapping Table*, that reports how the global attributes of a global class are mapped into the attributes of the local classes (see Figure 2).

GC	*name*	*rank*	*faculty*	*year*	*relation*	*e_mail*	*tax_fee*	*works*	*student_code*	*pay*
U.Research_Staff	first_name AND last_name	"Professor"	NULL	NULL	relation	e_mail	NULL	dept_code	NULL	NULL
U.School_Member	first_name AND last_name	"Student"	faculty	year	NULL	NULL	NULL	NULL	NULL	NULL
U. Worker	first_name AND last_name	NULL	NULL	NULL	NULL	NULL	NULL	dept_code	NULL	pay
CS.Person	name	NULL	"CS"	NULL	NULL	NULL	NULL	NULL	NULL	NULL
CS.Professor	name	rank	"CS"	NULL	NULL	NULL	NULL	division	NULL	NULL
CS.Student	name	rank	"CS"	year	NULL	NULL	NULL	NULL	NULL	NULL
TP.Student	name	"Student"	faculty_name	NULL	NULL	NULL	tax_fee	NULL	student_code	NULL

Fig. 2. The mapping table of our example

With respect to the representation of intensional overlappings used in [12], our mapping table contains information on how intensional conflicts are solved. We in fact apply the methodology not to the attributes taken from the local schemas to be integrated, but to the global classes of the GVV and to their global attributes.

2.2 Extensional integration

Once all the intensional conflicts are solved, we can proceed to the extensional integration. The challenge is to compose all the features that are related to a same modelled entity, these features being originally sparsed over the data sources.

We have to identify the extensional rules holding among the extensions of the local classes composing a global class. As a global class clusters in principle all the objects related to the same abstract entity, there must be no extensional relationships between local classes belonging to different global classes.

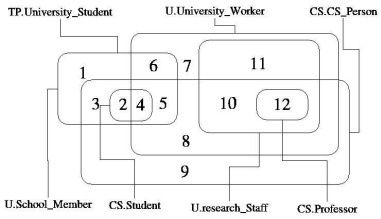
Extensional rules and base extensions *Def. We define the extension of a class C at time t as the set of objects that populate the class C at time t.*

Given two classes A and B we can have four types of extensional relationships:

- Equivalence: $A \equiv B \Leftrightarrow \forall t : Ext^t_A = Ext^t_B$

- Inclusion: $A \subseteq B \Leftrightarrow \forall t : Ext^t_A \subseteq Ext^t_B$
- Disjunction: $A \oslash B \Leftrightarrow \forall t : Ext^t_A \bigcap Ext^t_B = \oslash$
- Overlapping: $A \cap B \Leftrightarrow \forall t : \neg(A \equiv B) \wedge \neg(A \subset B) \wedge \neg(B \subset A) \wedge \neg(A \oslash B)$

If we give a graphical representation of the extensional relationships among the considered local classes, we obtain a partitioning over the union of the extensions. Figure 3 depicts the partitioning for our example, given the set of extensional rules presented in Section 2.



(a)

GC	1	2	3	4	5	6	7	8	9	10	11	12
U.Research_Staff	0	0	0	0	0	0	0	0	0	1	1	1
U.School_Member	1	1	1	1	1	1	0	0	0	0	0	0
U. Worker	0	0	0	1	1	1	1	1	0	1	1	1
CS.Person	0	1	1	1	1	0	0	1	1	1	0	1
CS.Professor	0	0	0	0	0	0	0	0	0	0	0	1
CS.Student	0	1	0	1	0	0	0	0	0	0	0	0
TP.Student	1	1	1	1	1	1	0	0	0	0	0	0

(b)

Fig. 3. The (a) graphical and (b) tabular representations of the resulting base extensions

Each such partition is called a *base extension*. Note that in Figure 3 every instance of a class belongs to only one base extension (whilst, if we consider classes, this is not true in general; for instance, when we have inheritance the instances of a class are instances also of the class it inherits from). In order to identify a correct set of base extensions, we have to consider also the so-called *existence requirements*, conditions that have to hold for the rules to be valid (see Table 1). For instance, if there is a rule of type $A \cap B$ then the following sets must be not empty: AB, $\neg AB$ and $A\neg B$.

Extensional rule	Existence requirement
$A \equiv B$	AB
$A \subset B$	$AB, \overline{A}B$
$B \subset A$	$AB, A\overline{B}$
$A \oslash B$	$A\overline{B}, \overline{A}B$
$A \cap B$	$AB, A\overline{B}, \overline{A}B$

Table 1. The existence requirements for each extensional rule

In [12, 14] the full algorithm for computing the set of base extensions of a set of local classes is reported and commented.

For each base extension belonging to a set satisfying the extensional rules and the existence requirements, we have that its extension is given by the intersection of the extension of the local classes composing the base extension and its intension is given by the union of the global attributes mapped by the local classes composing the base extension.

Extensional hierarchy We now briefly report how to organize the set of base extensions into a hierarchy. The construction of such a hierarchy facilitates the search operation for a base extension containing the attribute needed to solve a query.

To build the extensional hierarchy we use the theory of concept analysis [15] as explained in [12]. A context is defined as a triple (G, M, I), where G is a set of classes, M a set of attributes and $I \subseteq G \times M$ is a binary relation that says that a class $g \in G$ has the attribute $m \in M$ if and only if $(g, m) \in I$. We consider the context where G is the set of base extensions and M the set of global attributes. In order to define I we combine the information of the two tables we have built so far. The first one results from the intensional integration (see Figure 2) and tells how local attributes map to global attributes. The second one tells which local attributes belong to a base extension, for each base extension (see Figure 3(a)). It is now possible to obtain a third table, which reports which global attributes are mapped by each base extension (Figure 4). This is our binary relation I.

Base Extensions / ***Global attibutes***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***
name	1	1	1	1	1	1	1	1	1	1	1	1
rank	1	1	1	1	1	1	0	0	0	1	1	1
faculty	1	1	1	1	1	1	0	1	1	1	0	1
year	1	1	1	1	1	1	0	0	0	0	0	0
relation	0	0	0	0	0	0	0	0	0	1	1	1
e_mail	0	0	0	0	0	0	0	0	0	1	1	1
tax_fee	1	1	1	1	1	1	0	0	0	0	0	0
works	0	0	0	1	1	1	1	1	0	1	1	1
student_code	1	1	1	1	1	1	0	0	0	0	0	0
pay	0	0	0	1	1	1	1	1	0	1	1	1

Fig. 4. The mapping between base extensions and global attributes

The next step is to compute the intent of base extensions and the extent of global attributes.

Def. The intent of a subset of base extensions $A \subseteq G$ is composed by all global attributes $m \in M$ for which exists at least one base extension $g \in A$ such that $(g, m) \subseteq I$.

The intent of a single base extension is given by reading the corresponding column on Figure 4.

Def. The extent of a subset of attributes $B \subseteq M$ is composed by all base extensions $g \in G$ for which exists at least one $m \in B$ such that $(g, m) \in I$.

The extent of one global attribute is given by reading the corresponding row on Figure 4. We can now build the two sets $Int\{intent(\{g\}) : g \in G\}$ and $Ext\{extent(\{m\}) : m \in M\}$, corresponding to the set of intents for each base extension $g \in G$ and the set of extents for each attribute $m \in M$, deriving the two sets $Con_I\{(extent(I), I) : I \in Int\}$ and $Con_E\{(E, intent(E) : E \in Ext)\}$ containing concepts as follows:

The union of the two latter sets gives us a set of new objects, which we call here *virtual classes*. Table 2 lists the set of virtual classes we obtain for our example.

$C1 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, name)$
$C2 = (1, 2, 3, 4, 5, 6, 10, 11, 12, name, rank)$
$C3 = (1, 2, 3, 4, 5, 6, 8, 9, 10, 12, name, faculty)$
$C4 = (4, 5, 6, 7, 8, 10, 11, 12, name, works, pay)$
$C5 = (4, 5, 6, 8, 10, 12, name, faculty, works, pay)$
$C6 = (1, 2, 3, 4, 5, 6, name, rank, faculty, year, tax\_fee, student\_code)$
$C7 = (4, 5, 6, name, rank, faculty, year, tax\_fee, works, student\_code, pay)$
$C8 = (10, 11, 12, name, rank, relation, e\_mail, works, pay)$
$C9 = (10, 12, name, rank, faculty, relation, e\_mail, works, pay)$

Table 2. The existence requirements for each extensional rule

Over the set of virtual classes we can compute the binary relation that gives us the specialisation relation between two classes. The matrix representing the subset relations is illustrated in Figure 5 (a).

This matrix can be further refined by eliminating the transitive specializations as shown in Figure 5 (b).

The set of virtual classes can be organized in an inheritance hierachy following the specialisation relations synthetised in the non-transitive matrix. Figure 6 depicts the inheritance hierarchy for our running example.

3 Building a query plan

In this section, we show how we can use the intensional and extensional information to support query execution.

We first review the process that leads to the creation of a query plan. A query plan is needed whenever a query is posed on the GVV (thus called *global query*) and must be solved in terms of the integrated schema.

<	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0
C3	1	0	0	0	0	0	0	0	0
C4	1	0	0	0	0	0	0	0	0
C5	1	0	1	1	0	0	0	0	0
C6	1	1	1	0	0	0	0	0	0
C7	1	1	1	1	1	1	0	0	0
C8	1	1	0	1	0	0	0	0	0
C9	1	1	1	1	1	0	0	1	0

(a)

<	C1	C2	C3	C4	C5	C6	C7	C8	C9
C1	0	0	0	0	0	0	0	0	0
C2	1	0	0	0	0	0	0	0	0
C3	1	0	0	0	0	0	0	0	0
C4	1	0	0	0	0	0	0	0	0
C5	-1	0	1	1	0	0	0	0	0
C6	-1	1	1	0	0	0	0	0	0
C7	-4	0	-1	0	1	1	0	0	0
C8	-1	1	0	1	0	0	0	0	0
C9	-5	0	0	-1	1	0	0	1	0

(b)

Fig. 5. The (a) matrix and (b) its non-transitive form representing the specilisation relations among base extensions

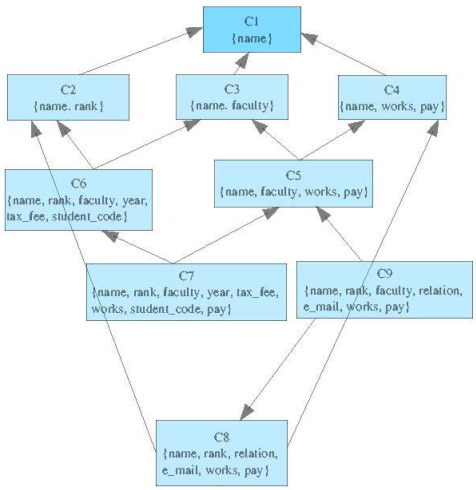


Fig. 6. The resulting inheritance hierarchy for our running example

Given a global query, the building of a query plan can be ideally decomposed into two actions: the identification of the local classes to be queried and the rewriting of the global query into queries executable on the identified local classes.

We take as our example query the following one: `SELECT name, rank, faculty from worker`.

3.1 Identification of the local classes

The first decomposition step is to ensure that a global query is splitted into a set of queries, each addressing one single global class. We call this kind of queries *basic queries*. In general, a global query is equivalent to a set of basic queries and a set of operations to join the answers.

For each basic query, the following steps are required to identify the set of local classes:

1. identification of the set AG_Q union of the set of global attributes contained in the *select-list* and the set of global attributes contained in the *where-clause*;
2. identification of the target virtual classes: exploiting the extensional hierarchy, we need to find out the most general virtual classes that hold all of the desired attributes:

$$TVC = \{V \in V | (AG_Q \subseteq INT(V)) \wedge (\neg \exists V' \neq V | AG_Q \subseteq INT(V) \wedge VISA_{EXT} V')\} \quad (1)$$

   The goal is to identify the *most general* virtual classes, i.e. we start examining from the root of the extensional hierarchy and go down to a lower level only if the considered virtual classes do no hold the complete set of attributes.
3. identification of the set of base extensions: the identified set of target virtual classes implies a set of base extensions, composed by the union of the extensions of the target virtual classes:

$$BE_{in} = \bigcup_{V \in TVC} EST(V) \quad (2)$$

   In order to compute a base extension we execute joins between the classes that compose it. Executing joins actually produces a superset of a base extension. We thus introduce the concept of dominance between two base extensions.
   Def. Given two base extensions B_1 and B_2 and a set of attributes $A = \{a_1, a_2, ..., a_n\}$, B_1 dominates B_2 with respect to A if and only if A is included in the intension of B_1 and in the intension of B_2 and the classes which compose B_1 are a subset of those which form B_2:

$$A \subseteq A_{BE}(B_1) \wedge A \subseteq A_{BE}(B_2) \wedge F(B_1) \subset F(B_2) \quad (3)$$

4. the set of base extensions BE_{in} identifies a set of local classes CL_{in} as follows:
$$LC_{in} = \{L \in SG(G) | \exists B \in BE_{in}, L \in F(B)\} \quad (4)$$
The set of local classes can be reduced in the following cases:
   (a) given two local classes having the same extension, we can delete from the set the one that has less attributes present in the query;
   (b) given two classes belonging to the same data source and a specialisation relationship between the two, we can delete the superclass from the subset.

In our case, the list of attributes is $name, rank, faculty$ and the most general virtual classes comprising these global attributes are $C2$ and $C3$. The set of corresponding base extensions is $BE = \{1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12\}$ which, considering the dominance of base extension 1 over base extensions 2 and 3, of base extension 6 over 4 and 5 and of 10 over 12, must be transformed into $BE_{opt} = \{1, 6, 8, 9, 10, 11\}$. The set of local classes identified by BE_{opt} is *LC*={ *U.Research_Staff, U.School_Member, U.Worker, CS.Person, CS.Professor, CS.Student, TP.Student*}.

3.2 Generation of local queries

The data structure obtained from the extensional integration allows us to determine which local classes have to be queried. At this stage, the *Mapping Table* resulting from the intensional integration (see Figure 2) comes into play.

For each identified local class, the original basic query must be rewritten. This requires both the translation of the global attributes refered by the query (in the *select-list* and *where-clause*) into local attributes and the transformation of the predicates of the *where-clause*. Particular attention must be paid to map global operands into operands that can be understood by the data source each local query refers to. How to perform this operation has been discussed in [14].

In our example, for the local class *U.Research_Staff* we can rewrite the basic query as `SELECT first_name, last_name from Research_Staff`. For the local class U.School_Member the query becomes `SELECT first_name, last_name, faculty from School_Member`. Analagously for the other local classes.

4 Implementing the query plan

In section 3 we have described how a plan can be obtained for a given basic query. Due to its unified nature, the plan is directly executable in a centralised way, under the control of a single module. This approach however does not seem to be very practicable for a number of reasons. First, a mediator system typically manages a large amount of data and offers its services to a potentially high number of users. Scaling up the performance of such a system is difficult. Secondly, operating in a distributed environment is risky, in the sense that some things may go wrong during execution.

Due to these constraints, we need a technique that permits the modification of the original query execution plan to cope with the unpredictable conditions of the environment. Distributing the control and the execution of a plan brings in general the following advantages:

- avoidance of a single point of failure by reducing dependence on centralized control;
- localisation of communication;
- local problems solved through local planning without traversing higher levels of control or execution;
- high-level planning and communication is involved only when local planning fails.

To shift from a centralised plan to a distributed version we propose to use the abstraction of agents and multi-agent system. A multi-agent system is a logical environment where software agents execute, interact and can possibly move. We call this logical environemnt a MAS platform. A MAS platform may comprise of a number of virtual places or container (usually linked to some physical host) and make them accessible in a seamless way to software agents. As discussed in [16, 13], software agents are entities that hold some nice features. For our application we restrict our attention to mobility, coordination and communication (see [7]).

The abstraction of a multi-agent system is particularly suitable to model distributed processes (see [9]). Rather than viewing the system as based on the client-server paradigm, we consider a mediator environment as a system composed by peers, each bringing data and possibly resources. We can decompose the problem in smaller ones and encapsulate the capabilites to solve each of them in a particular type of software agent. Agents will then communicate with each other in order to organize or better coordinate their actions. Agents can exploit the resources made available by the hosts participating to the agent platform to scale up the system.

To exploit the distribution of the environment, we need to reformulate the original query plan, preserving its meaning and organizing it as composed by many sub-plans. Once we have decomposed it, we can distribute it to a number of agents, each responsible for carrying out a portion of the plan, activating the necessary coordination mechanisms. In the following we present how this can be done with a query plan produced according to the steps described in previous sections.

4.1 Quality of service requirements

In order to answer a query we consider two basic properties to be guaranteed: (a) correctness and (b) completeness. Even though a correct and complete answer represents the ideal situation, in a distributed query process we assume correctness to hold more significance than completeness. This is to say that a correct but incomplete answer is to be preferred to a complete but incorrect answer. While presenting the re-planning actions that can be undertaken in our

scenario, we will point out when a modification to the plan impacts on the quality of the obtained answer, i.e. whether either correctness or completeness will be affected by the planned change.

4.2 Plan organisation

A query plan describes how the following steps have to be performed in order to reconstruct the answer to a basic query:

1. execution of local queries
2. reconstruction of base extensions
3. fusion of base extensions

The first two steps are dependent on queries executed on local data sources, while the fusion of base extensions can be perfomed by using the obtained partial results. It follows that until all base extensions are completely formed, we have to deal with the fact that data are distributed over different sources. Once we have the required BEs reconstructed the process becomes independent from the distribution of data sources. We will therefore restrict our attention to the portion of a query plan that consider the first two steps.

Def. A query plan P is of the form $\{LQ, JO\}$ where LQ is the set of local queries to be executed and JO is the set of join operations to be performed among the result of local queries.

Def. The knowledge $K(Q)$ required to build a query plan P is of the form $\{MT, ET\}$ where MT is the mapping table and ET is the extensional hierarchy.

Def. A planning space Ω is of the form $\{P, K(P)\}$, where P is the plan to be implemented and $K(P)$ is the knowledge used to build it.

Our ultimate goal is to distribute the execution and control of our planning space. This implies partitioning the actions to be executed and distribute them to different execution entities and partitioning the knowledge useful to re-planning actions among different control entities. We need to identify a way to associate in a coherent way to a sub-plan, the knowledge that serves to reason about re-planning for that sub-plan.

A rational way to decompose a planning space is to define sub-spaces which are closed with respect to the rest of the planning space. We call these sub-plans regions.

Def. A region α in a planning space $\Omega = \{P, K(P)\}$ is of the form $\{Q, K(Q)\}$ where $Q \subseteq P$ and $K(Q)$ is the knowledge pertaining to the sub-plan Q.

We can now define a partition of the planning space using the concept of region.

Def. Closeness: A region $\alpha = \{Q, K(Q)\} \in \Omega = \{P, K(P)\}$ is closed with respect to planning space Ω if the set Q has no intersection with the set of actions R of any other region $S = \{R, K(R)\} \in \Omega$.

Def. A planning space Ω is cloaseable if it is can be obtained as the union of closed regions.

In our case, this means that within a region, all local queries to be carried out do not depend from local queries belonging to a different region. This defines

a partition over the original planning space. Determining such regions has some nice consequences. The execution of the tasks of a region can be carried out independently from the rest of the plan. The execution of the tasks of a region can be executed remotely with respect to the rest of the plan. The control over the execution of the actions of a region can be demanded to an independet entity that knows all and only the information required to undertake re-planning actions on the sub-plan of the region.

In our case, regions are identified by base extensions. Identifiying regions with base extensions defines the desired partition over the original planning space.

A region will then comprise a sub-plan, including the set of local queries to be solved in order to resonstruct the corresponding base extension and the set of join operations to obtain the base extension from the produced query results, and a knowledge set including the part of the mapping table concerning the local classes corresponding to the local queries to be executed and the extensional relations of the virtual classes related to the base extension to be solved.

In our example query, base extension 1 is included in the set BE_{opt}. From the table mapping base extension on classes (Figure 3), the queries of interest for base extension 1 are those on the classes $U.School\_Memeber$ and $TP.Student$. The join operations have to be performed on attributes in common between the two, and more precisely, reading the $Mapping\ Table$ we have that the join attributes for $U.School\_Memeber$ are $first\_name$, $last\_name$ and $faculty$ and for $TP.Student$, $name$ and $faculty$. The knowledge related to this sub-plan is given by the subset of the $Mapping\ Table$ reporting the mapping for the two local classes and some extensional information. This includes: the subtrees of the extensional hierarchy starting with the virtual classes {C2, C3} and the dominance relations of base extension 1.

4.3 Plan distribution

We focus here on two aspects: the tasks we assign to agents (which define the agent types within the platform) and their organisation. Other issues that go beyond the scope of the paper are central when designing agents (see for instance [11, 17, 10]).

Task allocation The execution of the actions of a region is assigned to Execution Agents (EAs). Different choices can be made to decide which subset of local queries to assign to a particular EA. Among the alternatives, two ways appear feasible: either assigning the execution of a single query to a EA or assigning to a EA the queries that directly join to compose the result. This is a tradeoff to be evaluated. In any case, each local query is assigned to only one agent.

The control over the actions required to reconstruct base extensions is demanded to Control Agents (CAs). A CA agent is responsible for one base extension. A CA receives as input all the knowledge pertaining to one region (what we have defined to be its $K(Q)$) which includes a subset of the $Mapping\ Table$ and a subset of the extensional hierarchy relations.

Organisation Further to the knowledge about which queries to execute, an EA knows to which CAs it must send the obtained results. As a query can be used to compose different base extensions, the EA may be charged to contact more than one CA. On the other hand, decoupling the assignment of queries with the regions, gurantees that a query is executed only once.

Whenever a QA registers that one of its queries cannot be solved, it must alert all CAs which manage the reconstruction of a base extension for which the answer to that query is needed. This is a very critical problem as it means that the affected base extensions cannot be reconstructed. At this stage it is required a re-planning action. A CA which is notified such an event can undertake the following measures:

- intensional re-planning: exploits the ISA_{INT} relationships. Suppose there is a problem executing the local query on CS.Professor. Using the intensional knowledge, we can determine a class that is the father of CS.Professor, i.e. CS.Person. The basic requirement is the father class has all attributes belonging to the *where-clause*. This might imply the loss of the attributes proper to the child class and thus correctness cannot be guaranteed as these attributes cannot be used to perform join operations formerly required to get the base extension. Nevertheless we guarantee the completeness of the answer, as all of the instances of the child class are as well instances of its father;
- extensional re-planning based on ISA_{EXT}: when a base extension is not solvable, then we cannot build the virtual classes which include that base extension. However, it is possible to identify an alternative set of virtual classes that covers the set of attributes AG_Q and does not imply the unsolvable base extension. To determine this set exists, it is sufficient to take the extensional hierarchy and starting from the unsolvable virtual class, we go down until we identify the set of virtual classes covering all the attribues in AG_Q. Note that given a father, each child class in the hierarchy has only a subset of the base extensions of the father and thus it is possible to find a set of childs at some level that does not include the unsolvable base extension. This kind of re-planning guarantees correctness but not completeness as only a particular subset of instances corresponding to the father can be returned. It has also the advantage of reusing the results for the base extensions originally included in the plan and that still belong to the new plan;
- extensional re-planning based on dominance: when a base extension is not solvable, we need to identify an alternative set of base extensions that covers the set of attributes AG_Q. This can be obtained from the dominance relations of the base extension that is not solvable. This is not in general a re-planning measure to be undertaken as the dominated base extensions imply a superset of the local classes of the dominating one and therefore also the unsolvable base extension;
- raw re-planning: the most straightforward solution is to drop the unsolvable query from the list together with all join operations where it is involved. This produces an answer that is incorrect (instances are not filtered on the

missing partial results) and incomplete (the instances of the unsolved query are lost).

According to the quality requirement preferences expressed, a CA can implement an algorithm that takes into consideration the above re-planning measures, applying them according to the priority given to the quality requirements. For instance, if correctness prevails over completenss then the algorithm will try to pursue first an extensional re-planning based on ISA_{EXT}.

5 Related work

How to integrate schemas has been a topic exaustively tackled by previous work, both as far as the intensional integration is concerned (see [2]) and the extensional integration (see [12]). Our plan generation is based on ideas developped in [14].

The issues related to planning and how to undertake corrective or alternative actions to recover from failures pertain to field of distributed planning. A survey of the approaches and the directions the community is taking is reported in [4]. With respect to the classification proposed by DesJardins et al. our approach consider distributed, continual planning. Distributed as the control is spread over a number of entities that execute in different places (in terms of agent platform, containers). Continual is referred to the fact that the software agents exchange information about the ongoing activities and in the case of failure of some action, the community of agents tries to find out whether an alternative execution is practicable, even under the loss of some property (correcteness or completeness). Even though being influenced from other studies, our approach distinguishes from those described in [6, 5] because it is problem-specific, lacking any intention of capturing general aspects of planning outside the domain of distributed query execution for mediator systems.

6 Conclusions and Future Work

We have reviewed the integration process allowing a mediator system to build the GVV given a set of heterogeneous data sources. Both the intensional and extensional knowledge are needed in order to support the subsequent phase of query execution. Thanks to the mapping information produced during the integration process, we are able to construct a unified query plan. We have presented how to distribute control introducing the concept of planning space and regions and how to distribute execution by means of software agents. We have also shown as these two techniques combined allow for re-planning in the case of unsuccessful actions. Future work will address extensions of the proposed mechanism. Our first step will be to consider caching techniques based on the extensional knowledge in order to improve the time of response to queries.

Acknowledgements
I special thank to Sonia Bergamaschi for the revision of the present work.

References

1. Yigal Arens, Chun-Nan Hsu, and Craig A. Knoblock. Query processing in the SIMS information mediator. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 82–90. Morgan Kaufmann, San Francisco, CA, USA, 1997.
2. S. Bergamaschi, S. Castano, D. Beneventano, and M. Vincini:. Semantic integration of heterogeneous information sources. *Special Issue on Intelligent Information Integration, Data and Knowledge Engineering*, 36(1):215–249, 2001.
3. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
4. M. E. desJardins, E. H. Durfee, Jr. C. L. Ortiz, and M. J. Wolverton. A survey of research in distributed continual planning. *AI Magazine*, pages 13–22, Winter 1999.
5. J. Dix, H. Munoz-Avila, D. S. Nau, and L. Zhang. Impacting shop: Putting an ai planner into a multi-agent environment. *Ann. Math. Artif. Intell.*, 4(37):381–407, 2003.
6. E. H Durfee and V. R. Lesser. Partial global planning: A coordination framework for distirbuted hypothesis formation. *IEEE Transactions on Systems, Man and Cybernetics*, 1(1):63–83, 1991.
7. FIPA. Fipa communicative act library specification, 2000.
8. Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an information integration system. pages 539–542, 1997.
9. M. Huget. Agent uml class diagrams revisited, 2002.
10. John Mylopoulos, Manuel Kolp, and Jaelson Castro. UML for agent-oriented software development: The tropos proposal. *Lecture Notes in Computer Science*, 2185:422–??, 2001.
11. J. Odell, H. Parunak, and B. Bauer. Extending uml for agents, 2000.
12. Ingo Schmitt and Gunter Saake. Merging inheritance hierarchies for database integration. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems, New York City, New York, USA, August 20-22, 1998, Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems*, pages 322–331. IEEE Computer Society, 1998.
13. Katia Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph Andrew Giampapa. The retsina mas infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 2001.
14. F. Venuta. Trattamento della conoscenza estensionale nel sistema momis, tesi di laurea, 2000.
15. R. Wille. Concept lattices and conceptual knowledge systems. In *Computer and Mathematics with Applications*, pages 493–515, 1992.
16. Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–78. The MIT Press, Cambridge, MA, USA, 1999.
17. Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

A Self Organising Map (SOM) Sensor for the Detection, Visualisation and Analysis of Process Drifts

Dietmar Zettel[1], Daniel Sampaio[1], Norbert Link[1], Armin Braun[1], Michael Peschl[2] and Heli Junno[3]

[1]University of Applied Sciences Karlsruhe, Moltkestrasse 30, 76133 Karlsruhe, Germany
{dietmar.zettel, daniel.sampaio, norbert.link, armin.braun}@fh-karlruhe.de
[2]Harms & Wende Hamburg GmbH, Grossmoorkehre 9, 21079 Hamburg, Germany
michael.peschl@fh-karlsruhe.de
[3]University of Oulu, ISG, PO BOX 4500, 90014 Oulu, Finland
heli.junno@ee.oulu.fi

The control of processes (enforcement of given state variable values) by means of some driving force, where the state variables are not directly measurable quantities, is a very common problem class. The usual solution approach is by construction of a so-called observer based on an analytical process dynamic model and a statistical model of process and measurement noise. The observer gives an optimum estimate of the state variable values based on a measured history of accessible measured observable values. For cases, in which the necessary models are not available, we propose a SOM sensor which is able to estimate deviations (distance and direction) between states of the process and thus the process deviation from its optimum. The output can then be used as the input of a controller to determine the driving force. The SOM sensor is derived from a process sample containing a set of feature vectors (covering almost all process states) derived from the process observables. As SOM a two-dimensional, discrete Kohonen map is trained with a representative process sample to optimally reflect the topology of the original feature space. It is therefore well suited to visualise the movement of the process by the motion of the winner neuron in the SOM. The resulting path can be analysed in order to detect drift or to gain control relevant information. For the proof of concept a spot welding process consisting of several thousand welds over the whole lifetime of the welding machine electrodes is analysed as a sample application with the objective to reveal the quality loss related to electrode wear.

1 Introduction

The classical approach to solve the control problem in cases where the state variables are not directly accessible but only observables can be measured is to use a state observer, which is used to estimate the values of the state variables (figure 1). For this purpose a model of the dynamic of the process under consideration is needed. The model should reflect all aspects of the process including systematic disturbances like degradation of system components in order to give a correct estimate of the state variable. To arrive at a comparison between estimated state and the measured real

process observable one has to set up a second model, the measurement model, which defines how the state variables transform to the observable variables. The difference between the real and estimated observable values is fed back by the observer to the process model in order to correct the state estimate.

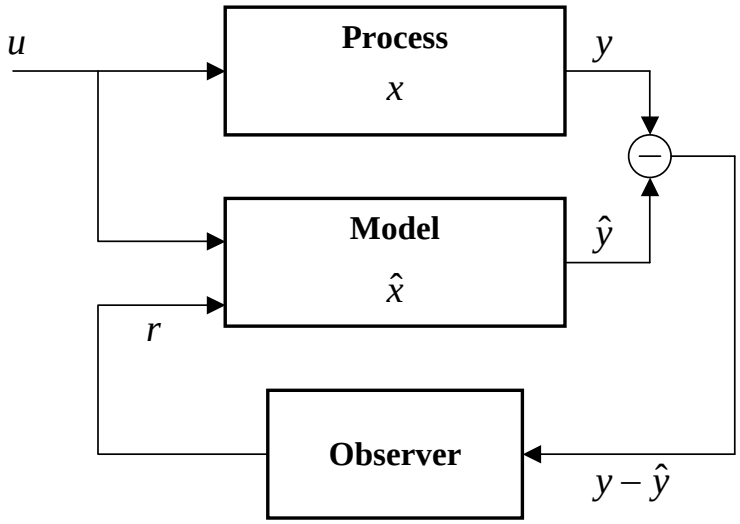


Fig. 1. Observer used to estimate state $\hat{x}$ from measurement $\hat{y}$

The most common observer is the Kalman observer [2]. It consists of a Kalman filter [3] that is responsible for the quantity estimation, a measurement model and a feedback loop with a Kalman matrix for the optimisation of the value estimated which is used as input in a process control (figure 2).

The requirements of such system are:

- Dynamic process model for the estimation of the state variable quantities;
- Model of the measurement system;
- Systematic disturbances (process or measurement) must be previously identified and included in the models or eliminated from the system;
- Process and measurement noise distribution functions must be known.

Many applications of the Kalman observer can be found in recent literature. In vision-based control of motion it is used for predicting the target position [14], in autonomous unmanned vehicle it is used to estimate a submarine position as shown in [15], another application of the Kalman Observer is to receive in a sensorless way the actual values of the rotor position, the rotor velocity, and the rotor flux of an induction motor. This practical example is based on a field orientated control method, where the necessary control variables position, speed, and rotor flux are estimated with a Kalman observer [16].

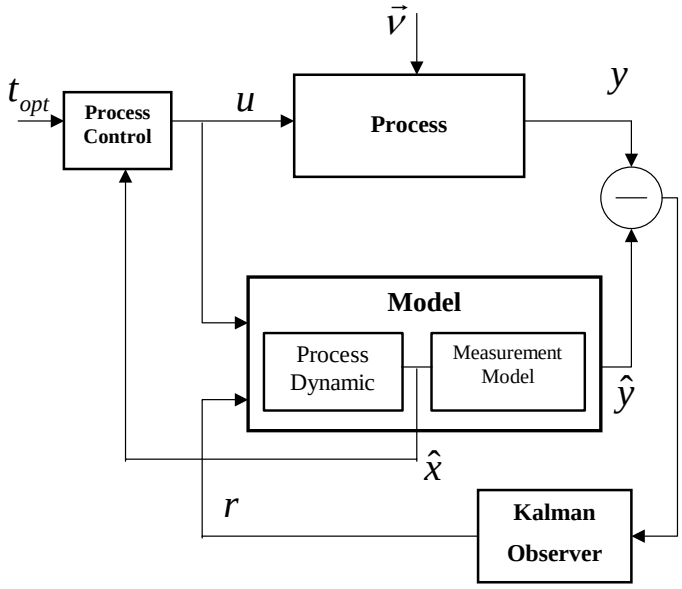


Fig. 2. Closed loop control with Kalman observer to estimate state $\hat{x}$ from measurement $\hat{y}$

Some common points in these applications can be identified. Sudden state changes are not well predictable by the Kalman observer, which possibly reacts with strong oscillations. Non-linear systems with uncertainties in modelling produce large estimation inaccuracies that perhaps can be compensated by the control strategy but has as result a relatively lower performance. Implementing a Kalman Observer is a very complex problem, and it requires a precise model to be calculated in real time. The observer equations must be calculated, which normally means many matrix multiplications and a matrix inversion. In many situations neither a process model is available nor the distribution of the process noise can be assumed to be normal. In these cases a statistical approach has to be developed.

2. Proposed Solution (Method)

In many cases, the processes are highly non-linear with many external influences that can not be modelled and also the noise distribution parameters can not be measured or reasonably be estimated. For such situations, in special for a process drift detection, in this paper the substitution of the Kalman observer by a statistical image of the process using a SOM is proposed (figure 3).

The process observable variables form the input vector $\vec{y}$ for the SOM. From a sample of the high dimensional input vectors $\vec{y}$, representative of the complete state set of the process, the SOM forms a two dimensional image (map) of the process states. By calibration, a map area can be marked, where the process can be identified as optimum. If the actual position of the process in the map (actual relative process

state) is in an area of the image apart from the optimum area, a process optimisation can be made on the basis of distance (d) and angle (φ) between these two areas. Both values (d,φ) are defined as the output of the statistical drift sensor.

This difference information should now be converted to the same quantity as used in the process control. The vector $\Delta\vec{C}$ represents the adjust values and can be used directly as input to the process controller.

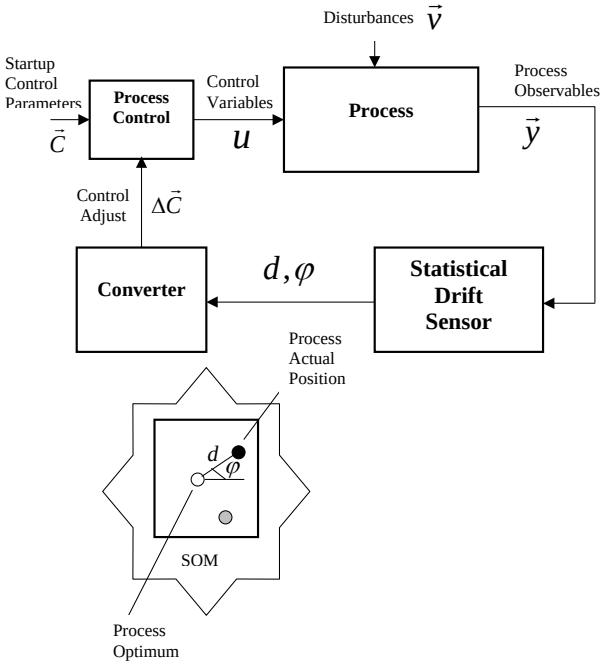


Fig. 3. Process control with SOM drift sensor

From the mapping of the high dimensional space of $\vec{y}$ in a two dimensional space it is also possible to:

- form and analyse the process path by tracking the process and
- visualise the process state and its temporal evolution.

Both information can be used to modify the start-up control parameters $\vec{C}$.

3. Application to Resistance Spot Welding

The characteristics of the statistical drift sensor allow to meet the application needs in the field of resistance spot welding. Resistance spot welding is a process very commonly used in the industry that consists of the joining of two or more metal parts together in a localized area, based on the heating produced according to Joule's Law ($Q = RI^2t$). The parts to be joined are pushed against each other by a tong-like

arrangement of two opposing electrodes. High current is passed through the parts via the electrodes and since heating (Q) is produced mainly at the interface between the parts due to the electrical current, a molten pool is created in this location by the net heating energy flowing in. Thermal expansion occurs and pressure should be applied in the electrodes in order to avoid expulsion of molten material. After switching off the current, this molten material cools down and a solid weld nugget is produced. The complete process consists of a repeating sequence of the production of such welding spots, where all spots should have the same good quality without respect to disturbances. For this kind of process no complete physical model (also taking into account disturbances like material unevenness, electrode wear and others) is available and even the measurement of necessary model parameters is not possible on-line. For example, these parameters are the temperature between the welded metal parts, the pressure over time applied by the electrodes and the conditions of the surface of the welded metal parts. Therefore no observer based in a model (figure 4) is applicable to this kind of process.

Only two quantities are accessible and can be measured during the welding process: the voltage and the current signals for each welding spot. The only material characteristic which can be extracted for each individual spot from the accessible voltage and current signals is the electrical resistance. The latter forms the basis for deriving process features. The desired state value would be (depending on the application) the value of a certain quality measure like the resulting spot diameter or the maximum shear force the spot can withstand.

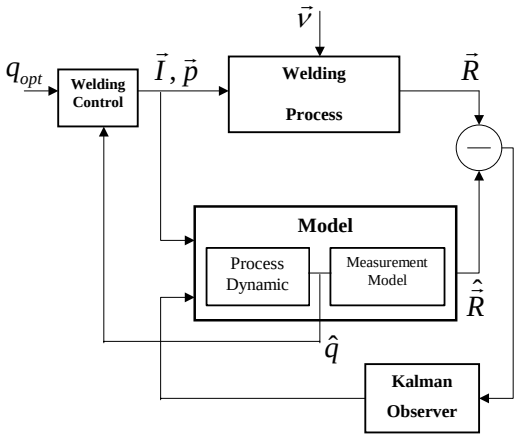


Fig. 4. Scheme of a welding process controller with Kalman observer

For such an application, a welding process controller is to be developed. It should be able to control a process with very short welding times (10-40 ms) and compensate electrode wear and also external disturbances. In this way, the controller should compensate the resulting process drift and keep the quality of the controlled process always in a optimum range.

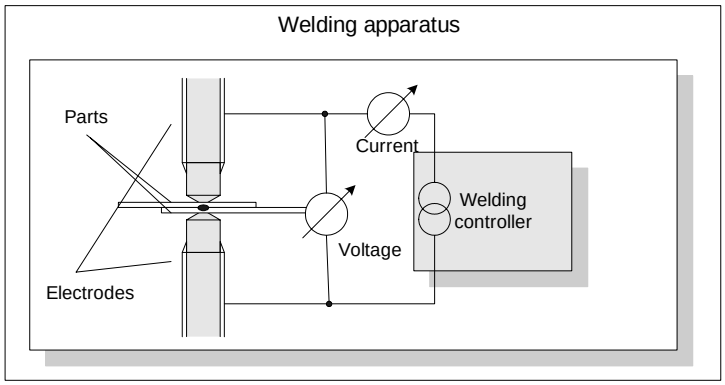


Fig. 5. Scheme of the welding process joining two metal sheets

The possible control variables in this process are the level of the welding current and the welding time given that in this application the electrode force is fixed. As the welding time is permitted to vary only in a very small range in order to maintain the machine's cycle time of 400 parts per minute, it is also kept fix. Therefore the only really controllable variable in this process is the welding current level.

The figure 6 shows the proposed spot welding control system in which the Kalman observer and the model of figure 4 is replaced by a SOM drift sensor connected to a converter.

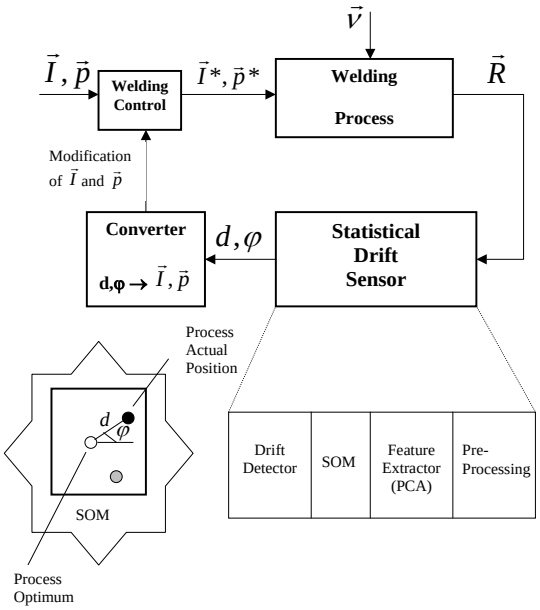


Fig. 6. Spot welding control system with SOM drift

4. The SOM Drift Sensor

The SOM drift sensor is composed of four components: A pre-processing unit to obtain noise filtered curves of the electrical resistance over time („resistance curve“), a feature extractor for a first reduction of the dimensionality, the SOM to finally reduce the process state space to a discrete, two-dimensional map and the drift detector to derive the difference from the optimum process state. The details and set-up of these components are discussed in detail below.

4.1 Pre-processing

In the application area of short-time weldings the middle frequency technology is state-of-the-art. Therefore the resistance curve has to be calculated from the phase-corrected current and voltage signals. For the removal of the high-level noise from these raw signals a non-linear filter is used, which smoothes the curves without eliminating relevant information.

4.2 Feature Extraction

The resistance curves from the pre-processing step are sampled with approximately 600 points. In order to arrive at a reasonable number of features with respect to the training and response time of the SOM, the linear sub-space, which approximates the sample in the best manner, is found via principal component analysis (PCA). The effect is:

- Reduction of the training time and the response time of the SOM during operation,
- Elimination of irrelevant information,
- Saving memory with the operational system.

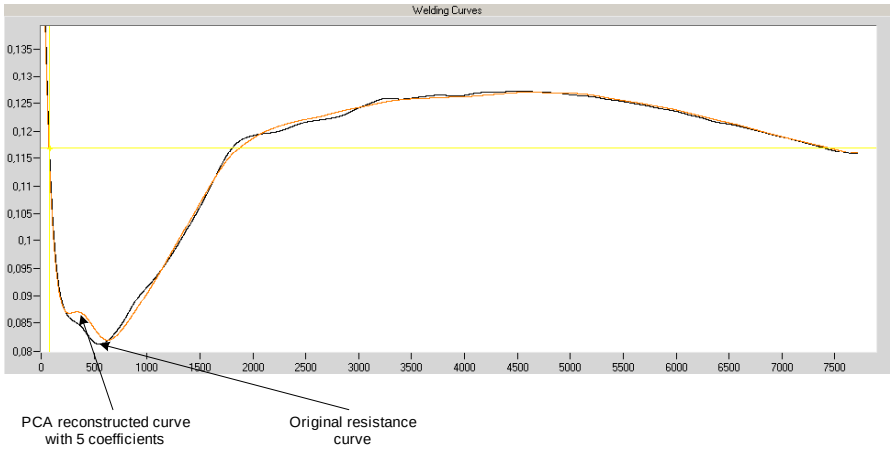


Fig. 7. Original resistance curve compared to resistance curve reconstructed from the first five coefficients

The resulting features are then the projection coefficients onto the first principal components (linear sub-space basis vectors with highest variance), which are calculated as the Eigenvectors with the highest Eigenvalues of the co-variance matrix of the sample vector set.

As shown in figure 7, only five such coefficients are sufficient to represent a resistance curve with originally 600 samples.

4.3 Self-organising Map

Self-organising maps (in our case we use Kohonen maps) are a special kind of neural networks, which consist of one layer of „active neurons“ [4]. They are mainly used for data analysis and data classification.

The basic property of a SOM within the context of our drift sensor is the mapping of the high-dimensional feature space onto an only two dimensional discrete map (grid). Each cell (also called neuron) of the grid is assigned a representative feature vector.

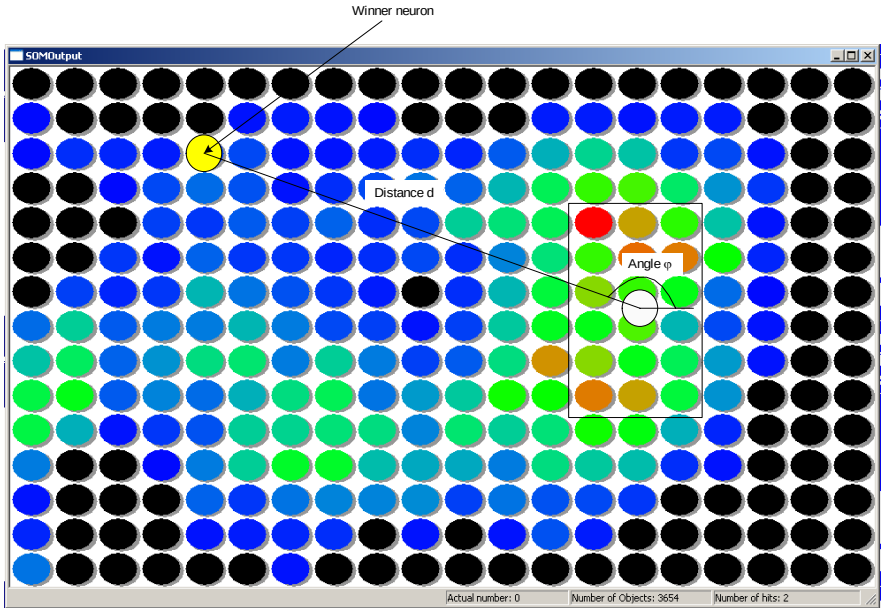


Fig. 8. SOM with process optimum area (rectangle, center: white circle), one winner neuron

These representatives (called weight vectors) are formed during the training from a representative feature vector sample of the process in a way that the neighbourhood relations of the feature vectors are optimally retained in the map. For the training of the SOM no a priori knowledge is required. After the training, each new feature vector is only characterized by the grid location of the most similar representative (the winner neuron) in the map. The grid positions of different such winner neurons define a 2-D difference vector, which can be used to measure a distance and a deviation

direction of the actual process state from the process optimum, once the map area of the optimum is identified (Figure 8).

Later on, a subsequent converter can use the difference measures to calculate increments of the welding parameters, which are fed into the welding controller in order to push the process back to the optimum state.

Once the welding signals have been pre-processed and the PCA coefficients have been calculated for all sample resistance curves, the following three steps are executed to set up the SOM:

1. Training;
2. Calibration of the SOM with respect to the process optimum;
3. Process observation.

4.3.1 Training

Before the SOM can be used as a sensor, it has to learn the process topology. For this purpose a process representative learning sample data set consisting of a set of PCA coefficient vectors (training vectors $\vec{X} = [x_1, .., x_n]^T$, n=number of PCA coefficients) is used. The representatives (weight vectors $\vec{w}_j = [w_{1j}, ..., w_{nj}]^T$, j: cell index) of each cell are initialised with random numbers. When training the SOM, in one training step a training vector $\vec{X}$ is compared with all weight vectors $\vec{w}_j$ to determine the most similar weight vector (winner). As difference measure we choose the mean square difference of the vector components. After the winner neuron has been found, all weight vectors are adjusted by adding a proportion of the difference from the training vector, weighted by a learning function, which depends on the map distance of the corresponding neuron from the winner neuron. Usually a Gaussian or a step function of the radius from the winner neuron position is used. This is repeated for all training vectors (14,900 in our case) in one training epoch. In the subsequent epochs, the variance of the Gaussian or the threshold of the step function as well as the proportionality factor is reduced and the procedure repeated, until the learning rate decreases to zero.

In order to define the training procedure, two steps are necessary:

1. Generation of the training data set;
2. Determination of the optimum learning parameters.

4.3.1.1 Generation of the training data set

Due to the fact that the state variables (the quality measure in our case) can not be monitored and a process representative sample is necessary, assumptions about the process underlying the training data set have to be made and verified, if possible.

The training data set must contain more data close to the process optimum than marginal data. The process optimum is defined as an ideal quality value, which is neither too low nor too high. With a too high quality the tools might wear too fast or the energy is not used most efficiently.

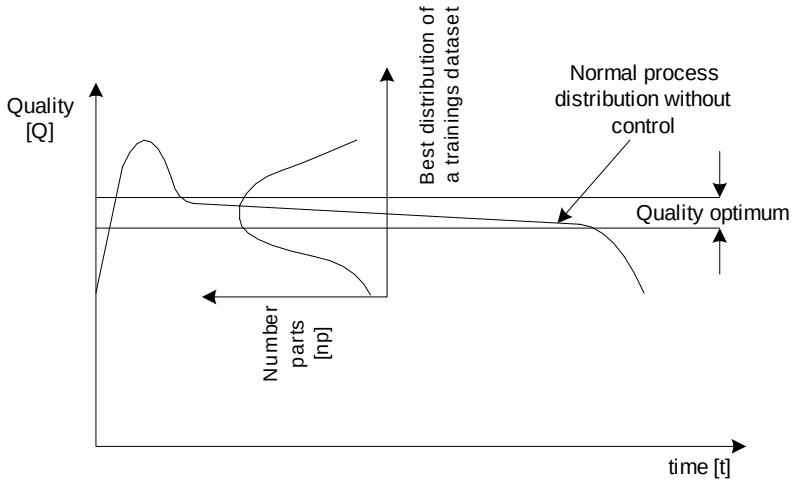


Fig. 9. Quality distribution of the training dataset from a regular industrial process

In a regular industrial production process these assumptions are justified on the average over the lifetime of the production tools used, if there is no interference by adjusting process parameters.

4.3.1.2 Determination of the optimum learning parameters

A common issue during the unsupervised training of a SOM is to find the ideal training, topology and initialisation parameters. There is no way to directly make sure the correctness of the produced SOM.

In order to determine the optimum learning parameters and obtain an accurate SOM, variables that are process-relevant and measurable should be selected. These variables should have also a direct effect on the not measurable actual process state x and can be considered to be their substitute in the context of SOM assessment. A representative dataset $T = \vec{t}_1 \ldots \vec{t}_n$ (n number of measurements) for the process under consideration is then determined. An initial training parameter set is defined as well as its range of variation. For each defined combination of training parameters the SOM is validated with the variables introduced above in order to find the best one.

The validation process consists of the following steps:

- A winner neuron is found for each vector of the dataset T and the value of the chosen relevant variable is associated to it.
- For each neuron, a histogram is created with all associated values belonging to it.
- The associated value with highest frequency in the histogram will define the class of the neuron.

In an ideal trained SOM, the sum of the histogram's highest frequency of each neuron (called here score) should be equal to the size of the training dataset, this means that vectors with the same relevant variable value will be associated to the same neuron (belong to the same class) and a neuron will embrace only vectors with

the same relevant variable value. In a real case this rarely will happen due to the reduced dimension of the SOM, the size of the dataset T and the number of classes present in it. Therefore the combination of training parameters that achieve the highest score will be considered the optimum learning parameter combination due to the best separability of classes.

$$s = \sum_{i=0}^{n} h_{winner} \quad (\mathbf{1})$$

where:

h_{winner} = Maximum value of the histogram,
s = Score for the SOM
n = Number of neurons in the list

The ideal score s is the number of training vectors.

This method of varying the training parameters and obtain for each combination a score has been called “Scoreboard”. The figure 10 shows an example for a scoreboard.

Score: 65 - Directory:
2003-12-10_v002/rect/9x8/bubble/alpha_0.9/radius_3/rlen_1000
Score: 65 - Directory:
2003-12-10_v002/rect/10x7/bubble/alpha_0.9/radius_10/rlen_200000
Score: 65 - Directory:
2003-12-10_v002/hexa/9x8/bubble/alpha_0.9/radius_4/rlen_500000
Score: 64 - Directory:
2003-12-10_v002/rect/9x7/bubble/alpha_0.9/radius_5/rlen_20000
Score: 64 - Directory:
2003-12-10_v002/rect/10x8/bubble/alpha_0.9/radius_5/rlen_1000
Score: 64 - Directory:
2003-12-10_v002/rect/10x8/bubble/alpha_0.9/radius_10/rlen_20000
Score: 64 - Directory:
2003-12-10_v002/rect/10x7/bubble/alpha_0.9/radius_9/rlen_200000
Score: 64 - Directory:
2003-12-10_v002/rect/10x7/bubble/alpha_0.9/radius_9/rlen_100000
Score: 64 - Directory:
2003-12-10_v002/rect/10x7/bubble/alpha_0.9/radius_6/rlen_20000
Score: 64 - Directory:
2003-12-10_v002/rect/10x7/bubble/alpha_0.9/radius_5/rlen_500000

Fig. 10. Example for a Scoreboard

4.3.2 Calibration of the SOM with respect to the process optimum

Under the assumption made for the training data set, the label „process optimum“ defines the region of the map where most of the hits (winner neuron occurrence) can be found. In complement to that, a histogram for the whole SOM is created. During the tests clearly a region showed up where a small group of neurons forms a hit frequency maximum. After labelling the „process optimum“ region, the distance and angle for each other neuron or region in the map can be calculated (see fig. 8).

4.3.3 Process observation

The first tests with a trained SOM showed a very noisy behaviour of the winner neuron position in the map, which reflects the process noise. A Kalman filter might again be the appropriate measure if latency would be a serious problem. In our case the noise correlation times are short compared to the time constant of the system and a much simpler sliding average filter was implemented, where the filtered position x_f and y_f is calculated from the actual and past s-1 positions x and y by:

$$x_f = \frac{1}{s}\sum_{i=0}^{s-1} x_{(p-i)} \tag{2}$$

$$y_f = \frac{1}{s}\sum_{i=0}^{s-1} y_{(p-i)} \tag{3}$$

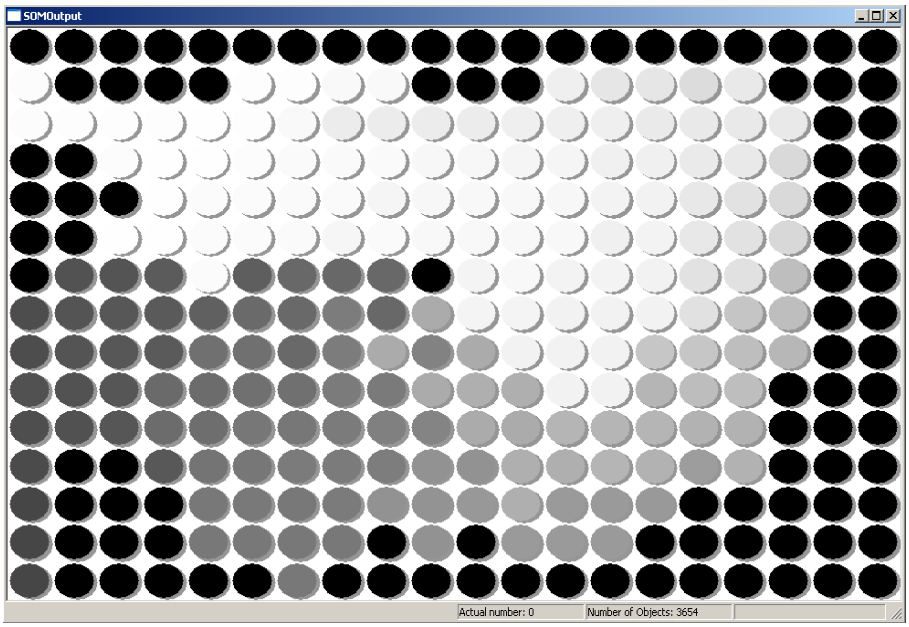


Fig. 11. Process motion through the map (dark gray=begin, middle gray= middle, white=end)

After this filtering with s=10 a clear path of the process becomes visible, where the process (marked by the winner neuron) only moves to neighbouring neurons at a time. This is illustrated in figure 10, where winner neurons are coloured according to their sample number (proportional to process time). Only the last hit was retained in the

map colouring. The counter-clockwise motion of the process can be clearly seen. The observed path from high distance from the optimum in the beginning to a first „island of stability“, then to the process optimum (where the process lasts for the longest period) and then again to high distance at the end is in good agreement with quality observations of the producers.

5. Conclusion

A Self-Organising-Map sensor is proposed to generate input for process controllers, where the state variables are not available and an observer approach is not possible due to lack of appropriate models. A difference between process optimum and present state is derived from the process map, which is obtained by training with process samples and can be used as controller input. The approach was applied to a spot welding process. A general procedure for the training in such cases was set up and successfully applied. The process optimum map area was identified and the motion of the process in the map was visualized and found to be in accordance with quality observations of similar spot welding processes. On going work is to check the reusability of such maps and to close the control loop.

Summarising, the observer approach and the SOM sensor approach are compared in the table 1.

Table 1. Comparison between Observer and SOM Sensor

	Observer	**SOM Sensor**
Estimation of state variables	yes	not possible
Process Model	necessary	not needed
Training (sample)	not needed	necessary
Process Modification	new model must be set up	new training data set must be recorded
Systematic Disturbances	must be modelled or eliminated	must be contained in training data set

The observer has a clear advantage when the process is well defined and a precise dynamical model exists, while in all other cases the SOM sensor is preferable.

6. Acknowledgements

We would like to express our gratitude to our colleagues at the University of Oulu and in the Harms + Wende GmbH & Co.KG and also SBT Stanzbiegetechnik GesmbH Austria for providing the data set and the expertise needed at the various steps of research and for numerous other things that made it possible to accomplish this work. Furthermore, this research has been carried out with financial support from the Commission of the European Communities, specific RTD programme “Competitive

and Sustainable Growth", G1ST-CT-2002-50245, "SIOUX" (Intelligent System for Dynamic Online Quality Control of Spot Welding Processes for Cross(X)-Sectoral Applications"). It does not necessarily reflect its views and in no way anticipates the Commission's future policy in this area.

7. References

1. O. Föllinger: Regelungstechnik, 7. Auflage, Hüthig, Heidelberg, 1992, p.511 ff
2. Arthur Gelb: Applied Optimal Estimation, 9th ed., MIT Press, Cambridge, MA, 1986
3. Mohinder S. Grewal, Angus P. Andrews: Kalman Filtering: Theory and Practice, J. Wiley & Sons, 2001
4. Andreas Zell: Simulation Neuronaler Netze, Addison Wesley 1994
5. Armin Braun: Self-organising maps (SOM's) for analysis and visualisation of resistance sport welding processes, Master Thesis FH-Karlsruhe 2004
6. SIOUX Midterm Report, (EU project G1ST-CT-2002-50245) 2003
7. SQUAW Final Technical Report, (EU project BRPR-CT98-0677) 2001
8. Kohonen, Teuvo; Hynninen, Jussi; Kangas, Jari; Laaksonen, Jorma: SOM_PAK – the self-organising map program package, version 3.1, Helsinki University of technology 1994
9. Kohonen, Teuvo: Self-organising maps, Springer 2001
10. Richard O. Duda, Peter E. Hard, David G. Stork: Pattern Classification, Wiley & Sons 2001
11. Martin Hart: Auswertung direkter Brennrauminformationen am Verbrennungsmotor mit estimationstheoretischen Methoden, Doktorarbeit Universität-Gesamthochschule Siegen 1999
12. William H. Press, Saul A. Teukolsky, William T. Vetterling, Brain P. Flannery: Numerical Recipes in C, Second Edition, Cambridge University Press 1996
13. Dietmar Zettel: Data Mining for Resistance Sport Welding Process. Master Thesis FH-Karlsruhe 2003
14. S. Chroust, E. Zimmer, M. Vincze, Pro and Cons of Control Methods of Visual Servoing, Vienna University of Technology – Institute of Flexible Automation, Proc. 10th Int. Workshop on Robotics in Alpe-Adria-Danube Region, 2001.
15. Kevin J. Walchko, David Novick, Michael C. Nechyba, Development of a Sliding Mode Control System with Extended Kalman Filter Estimation for Subjugator, University of Florida www.mil.ufl.edu/publications/fcrar03/Walchko-1.pdf
16. S. Bejerk, Digital Signal Processing Solutions for Motor Control Using the TMS320F240 DSP-Controller, First European DSP Education and Research Conference, 1996
17. Heli Junno, Perttu Laurinen, Eija Haapalainen, Juha Röning, Resistance spot welding process identification and initialisation based on self-organizing maps, to be published in 1st International Conference on Informatics in Control, Automation and Robotics, ICINCO 2004

Kohonen Networks for Self-organizing Performance of Two Queues Markov Chains

Dimitar Radev[1], Svetla Radeva[2]

[1] Assoc. Professor, Ph.D., Eng., Department of Communication Technique and Technologies, University of Rousse, 7017 Rousse, Bulgaria
dradev@abv.bg
[2] Assoc. Professor, Ph.D., Department of Computer Aided Engineering, University of Architecture Civil Engineering and Geodesy, 1046 Sofia, Bulgaria
svetla_fce@abv.bg

Abstract. The paper is devoted on implementation of Kohonen networks for modeling of two node queues, presented with embedded Markov chains. Vector quantization is implemented for performance of stochastic networks with parallel and tandem two node queues. Probability density estimation for feasible set of arrival entrance process and holding time services is modeling as combination of single queues with random distribution. Kohonen learning rules are applied for two-dimensional vector quantization. The performance estimation of steady state tandem Jackson queue with infinite first queue is realized with self-organizing map. Simulation and numerical results for communication networks with Poisson, Gamma and exponential distributions are shown.

1 Introduction

Stochastic simulation of queuing networks is using for investigation of the probabilistic processes at entrance distribution and link occupancy distribution, call blocking, continuous arrival processes, remaining time services phases, overflow probability, consecutive cell loss performance, packet loss, holding and remaining time services, which determine quality of service and guarantees call level of communication networks. The queuing theory studies systems in which customers randomly arrive at a service station in order to be served, since there may be other customers ahead of them and they may need wait in a buffer. For studying the properties of queuing models most often are searched solving of the problem for homogeneous Markov arrival processes with the help of Discrete Time Markov Chains (DTMC) [1], [2], [7]. The steady state analysis of Markov chains is formulated as the solution of a special linear system with direct, iterative and projection methods [13]. There are several algorithms for the numerical steady state analysis of Markov models with more than 10^4 states, which are no general [4].

In this investigation is suggested one generalized approach with Kohonen networks for steady state clustering of probability distribution of two-dimensional (2D) DTMC with variety of input data.

2 Vector Quantization for Discrete Time Markov Chain Models

Let the stationary stochastic process is presented with time series $\{\mathbf{x}\}$ for which $X_0,X_1,\ldots,X_i,\ldots, X_n, \in S$, is a series of discrete non-negative random variables of DTMC, when the Markov property holds for all X_i. Than for the conditional distribution **P** is valid (1) for all finite state space $S = \{1,2,\ldots,M\}$.

$$\begin{aligned}&\mathbf{P}\left(X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_0 = x_0\right) = \\ &= \mathbf{P}\left(X_n = x_n \middle| X_{n-1} = x_{n-1}\right)\mathbf{P}\left(X_{n-1} = x_{n-1} \middle| X_{n-2} = x_{n-2}\right)\ldots \\ &\ldots\mathbf{P}\left(X_1 = x_1 \middle| X_0 = x_0\right)\mathbf{P}\left(X_0 = x_0\right)\end{aligned} \tag{1}$$

For steady state analysis is used a stochastic system, which is in a state $j \in S$, of the state space S in every discrete time epoch. The consequences of stability for embedded DTMC are determined of n transient state probability vectors (2) and k one-step transient probability matrixes (3).

$$\mathbf{P}^{(n)} = \left[p_0^{(n)}, p_1^{(n)}, p_2^{(n)}, \ldots\right], \qquad p_i^{(n)} = \mathbf{P}\left(X_n = i\right) \tag{2}$$

$$X_n \rightarrow \quad \mathbf{A}^{(k)} = \left[a_{ij}^{(k)}\right], \quad a_{ij}^{(k)} = \mathbf{P}\left(X_{n+k} = j \middle| X_n = i\right) \tag{3}$$

The transition into consecutive states is determined of the probability, that the customer leaves state i for state j at time n, determined with (4).

$$\mathbf{P}\left(X_n = j \middle| X_n \neq i, X_m = i, m < n\right) = \frac{a_{ij}}{\sum_{m \neq i} a_{im}} \tag{4}$$

Every transition in the DTMC corresponds to an elementary event in the queuing model: an arrival or a service completion at one of the queues. These transition events are defined independently of the state, and there is only one transition event for a service completion at a given queue. This single transition event corresponds to a transition out of every state in which the particular queue is not empty. Not all transition events are enabled in every state (in the state where a particular queue is empty, the service completion event of that particular queue is not possible). In this way the elements of k one-step transition matrixes can be evaluated with (5).

$$\mathbf{A}^{(k)} = \begin{cases} a_{ij}^{(k)} = \dfrac{a_{ij}}{\sum_{m \neq i} a_{im}} \\ a_{ii}^{(k)} = 0 \end{cases} \tag{5}$$

Based on the well-known results available for the matrix elements, (5) have been applied for numerical computation of probability distribution of high order continuous and discrete Markov chains [8], [14]. Although because of high complexity and variety of constraint conditions in most of cases the transient analysis leads to a rude representation of the reality. In this study for performance evaluation the real configuration of telecommunication network can be treated as certain combination of two queues, connected in parallel or in tandem. In [12] are presented the

transformation rules according to which a complex configuration of queuing system is transforming into a combination of two-node queue networks with 2D DTMC models. In these models the states are arranged on a grid with as many dimensions as the number of queues and on each axes are represented the number of customers in one of the queues. The probability of interest is determining of the occupancy and entrance distribution, partial overlap, call blocking probabilities of steady state or feasible states of 2D Markov sets.

2.1 Two Parallel Queues Performance Estimation

Let consider the chain of two parallel queues with Markovian arrival and time service processes. The input parameters of the system for each state are the set of arrival rates (λ_1, λ_2) and departure rates (μ_1, μ_2). The model state is unique characterized by the couple (n_1, n_2), where n_1 is the number of customers in the first queue and n_2 – number of customers in the second queue.

The queues n_1 and n_2 are with full access and only transitions between neighboring states are allowed as presented on Fig. 1.

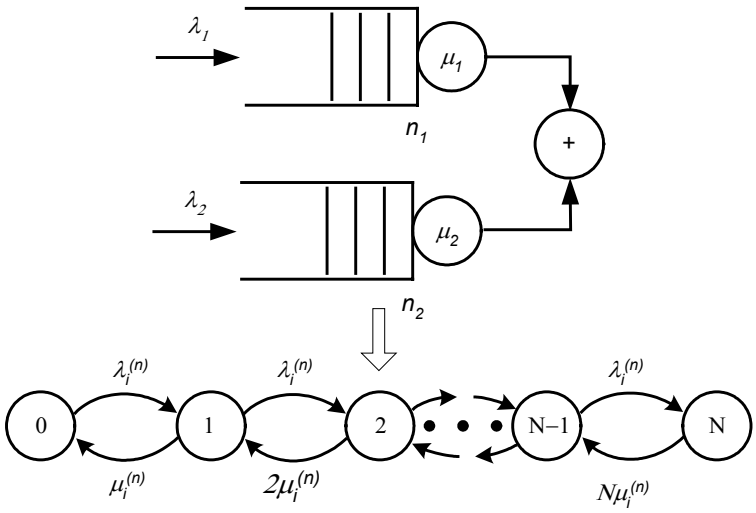


Fig. 1. Two parallel queues M/M/N/N performance

The two-node queues can be presented as two (n=1,2) separate M/M/N/N queues, which arrival rates λ_j , and departure rates μ_j , j=1,2,…, N, and N is the number of states of DTMC model. In this way the queues n_1 and n_2 determine the auxiliary coordinates of two-dimensional grid set for visualization of the state attribute space of the discrete model. There is four transition events: arrivals rate at the first and second queues λ_1 and λ_2 and full access holding service times $i\mu_1$ and $j\mu_2$ for both queues. The transition from the state i to state j of embedded discrete Markov chain is described with the graph (6).

$$a_{ij} = \begin{cases} \lambda_1 & i = i+1, \quad j = j, \quad 0 \le i, j \le N \\ \lambda_2 & i = i, \quad j = j+1 \\ i.\mu_1 & i = i-1, \quad j = j \\ j.\mu_2 & i = i, \quad j = j-1 \end{cases} \tag{6}$$

As an illustration, is presented a model of two-dimensional set of two parallel queues. Let the embedded discrete Markov chain has, for example, 16 feasible steady states, as is shown on Fig. 2, upper. The model nodes are described as a combination of steady states of the two queues. The one-step transitions between these steady states are realized with orthogonal vectors, where for example, for state (1,2) n_1 =1, n_2 =2, λ_1 is orthogonal to λ_2 and $2\mu_1$ is orthogonal to $2\mu_2$. The one-step transition in both queues is independent of each other. At the same time this DTMC can be presented via two-dimensional Vector Quantization (VQ) with separated inputs. If the both inputs are the time series $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, as is presented at (7), than we can receive 2D model of Vector Quantization with axial coordinates, which are number of customers n_1 and n_2.

$$\mathbf{x}^{(1)} = \left\{X_1^{(1)}, X_2^{(1)}, ..., X_N^{(1)}\right\} \qquad \mathbf{x}^{(2)} = \left\{X_1^{(2)}, X_2^{(2)}, ..., X_N^{(2)}\right\} \tag{7}$$

The clustering problem is reduced to a set of hit values, which determine belonging to one of M target classes $S_1,\ldots,S_M$, (as is shown on Fig. 2, down, M = 4 ×4 = 16). Than each input cluster can be presented only with the class to which it belongs. The vector quantization can be adapted in such a manner, that for each steady state, the Markov chain corresponds to strictly determined class (for 16 steady states we have 16 rectangular cluster classes). The components of the input time series most often are continuous values and we can keep the label of corresponding target class instead of the input vector. Indeed, while class labels are not used to constrain the structure of the model, freedom from this constraint coupled with careful initialization of the models using any prior information available about the data, can yield very quick and effective models. These models have the additional feature that the centers are arranged in a low dimensional rectangular grid, such that nearby points in the topological structure map are to nearby points in the attribute space [3]. The probability distribution of input clusters of time series is determined as conditional probability of random values, which output are associated with the distribution of probability (8).

$$\mathbf{F}\left(x_1^{(1)}, ..., x_N^{(1)}, x_1^{(2)}, ...x_N^{(2)}\right) = \mathbf{P}\left\{X_1^{(1)} \le x_1^{(1)}, ..., X_N^{(1)} \le x_N^{(1)}, X_1^{(2)} \le x_1^{(2)}, ..., X_N^{(2)} \le x_N^{(2)}\right\} \tag{8}$$

The basic problem is to determine weight centers of each class, which in practice correspond to feasible steady states of DTMC. Furthermore, using a neural structure with Kohonen learning rules gives possibility for optimal adjusting of classes' boundaries in such a manner to receive the most possible precise solution for probability density function.

As is described in [11], Learning Vector Quantization (LVQ) with adjusting of target classes boundaries, can suggest a solution for which the weights centers of target classes (the circles on Fig. 2, down) approximates the Markov chain feasible states. The boundaries of classes are orthogonal to the lines connecting the weight vectors of neighbor classes.

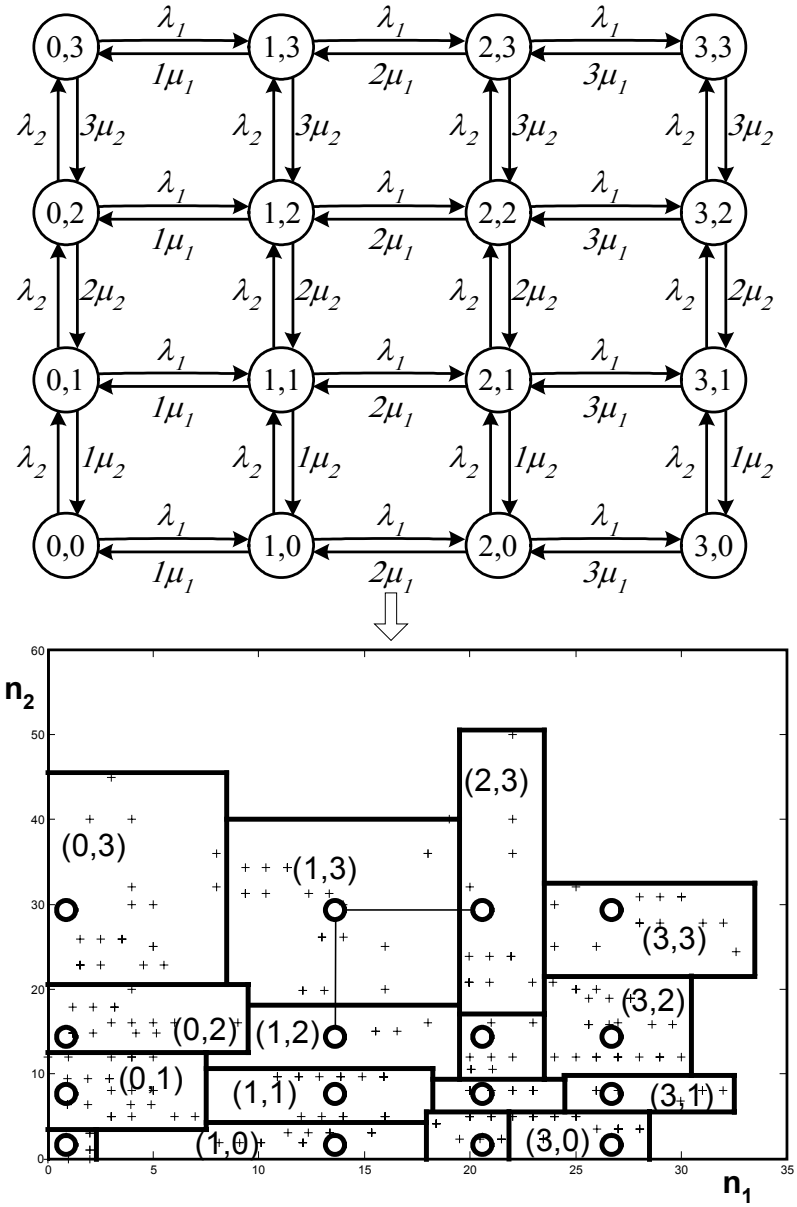


Fig. 2. Solution of the 2D stable set of parallel M/M/N/N queues with finite interarrival and service times

2.2 Markov Chain Models for Tandem Queues

Consider the tandem Jackson queues n_1 and n_2 with arrival rate λ and holding time services of first and second queues $\mu_1$and μ_2, as is presented on Fig. 3.

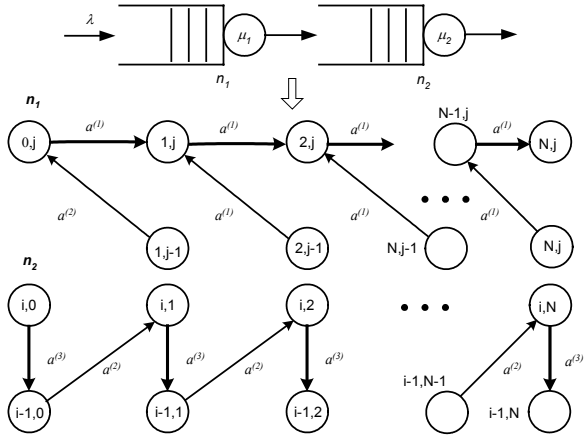


Fig. 3. Tandem queues performance with DTMC

The customer arrival processes and time series phases are random distributed Markovian variables. The complete state of the Markov model is given with the pair (n_1, n_2). The set of feasible steady states are defined with the help of possible movement between neighbor nodes according three transient matrixes, which describe the horizontal, up and down transitions from state (i,j) to corresponding state (i+1,j); to state (i-1,j+1) and to state (i,j-1), presented at [12]. The horizontal transition increases the content of queue n_1 with arrival rate λ, described with the graph (9).

$$a_{ij}^{(1)} = \begin{cases} \dfrac{\lambda}{\lambda+\mu_1+\mu_2} & i = i+1, \quad j = j, \quad i \neq 0, \quad j \neq 0 \\ \dfrac{\lambda}{\lambda+\mu_1} & i = i+1, \quad j = j, \quad i \neq 0, \quad j = 0, \\ \dfrac{\lambda}{\lambda+\mu_2} & i = i+1, \quad j = j, \quad i = 0, \quad j \neq 0, \end{cases} \tag{9}$$

The up transition increases the content of queue n_2 and at the same time decreases the content of queue n_1 with departure rate μ_1 and is described with the graph (10).

$$a_{ij}^{(2)} = \begin{cases} \dfrac{\mu_1}{\lambda+\mu_1+\mu_2} & i = i-1, \quad j = j+1, \quad i \neq 0, \quad j \neq 0 \\ \dfrac{\mu_1}{\lambda+\mu_1} & i = i-1, \quad j = j+1, \quad i \neq 0, \quad j = 0, \\ 0 & i = i-1, \quad j = j+1, \quad i = 0, \quad j \neq 0, \end{cases} \tag{10}$$

The graph $\mathbf{a}^{(3)}$ presents down transitions, which makes the content of the queue n_2 empty with departure rate μ_2 and its elements are presented at (11).

$$a_{ij}^{(3)} = \begin{cases} \dfrac{\mu_2}{\lambda + \mu_1 + \mu_2} & i = i, \quad j = j-1, \quad i \neq 0, \quad j \neq 0 \\ 0 & i = i, \quad j = j-1, \quad i \neq 0, \quad j = 0, \\ \dfrac{\mu_2}{\lambda + \mu_2} & i = i, \quad j = j-1, \quad i = 0, \quad j \neq 0, \end{cases} \tag{11}$$

As is seen from graphs (9) – (11), for the three one-step transient matrixes the values in matrixes kernel differs from axial values.

The three one-step transitions matrixes are not orthogonal to each other, which make difficult the presentation of embedded two-dimensional DTMC as a two-dimensional neural network with separated inputs. The one-step matrix $\mathbf{a}^{(2)}$ influences on horizontal queuing structure n_1 and on vertical queuing structure n_2 (see Fig. 3, down). This influence is restricted via treating of n_1 and n_2 as time series with random discrete distribution.

In this research is suggesting an approach in which we utilize distribution functions of the two tandem queues without the necessity of using continuous numerical procedures about the number of phases of arrival processes and the general holding time. In this approach, is spread the scope of the input distribution in such a manner that we approximate single G/G/1, M/G/1, GI/M/1 and M/M/1 queues with the help of combinations of approximations of standard distribution functions.

The probability of interest is the probability distribution of the total population of the two queues, which reaches a given level L within a busy cycle. The system starts immediately after the first arrival of a busy cycle in state (1,0). It is searched the probability distribution for $n_1 + n_2 \leq 2M$.

It is possible to solve of two separated tasks. In first case the first queue n_1 is finite and the solution is searched in constraints $n_1 \leq M$ and $n_2 \leq M$. The embedded two-dimensional DTMC in this case is squared grid with 16 steady states, shown on Fig. 4 with small empty white circles. The customers constraints are reached at all states $(3,j)$ and $(i,3)$. The solution of this task is analogous as presented at 2.1 example, and can be made with the help of LVQ neural network.

The second task is solving for the case of infinite first queue and constraints $n_1+n_2 \leq 2M$. Then all end steady states $(i,3^{(i)})$ and $(3,j^{(i)})$ lies on the line $n_1 + n_2 = 2M$. The transformation of these states is shown on Fig. 4 with big gray circles. Analogous are transformed steady states $(i,2^{(i)})$ and $(2,j^{(i)})$, which as well lies on a line. In this way are transformed all steady states with exceptions of the states (i,j) for all $i=j$, which satisfy the conditions of both tasks. The analysis shows that embedded DTMC for infinite first queue is not appropriate to be solved with the help of rectangular or squared cluster zones. Analogical solutions lead to big difficulties with searching steady states probability density distribution and increases number of errors [9].

In this research will be shown one solution of the second task with implementation of self-organizing map learned to classify input time series $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ according to how they are grouped in the attribute space. This is deferent from competitive layer of LVQ, where neighbouring neurons are learned to recognize neighbouring clusters.

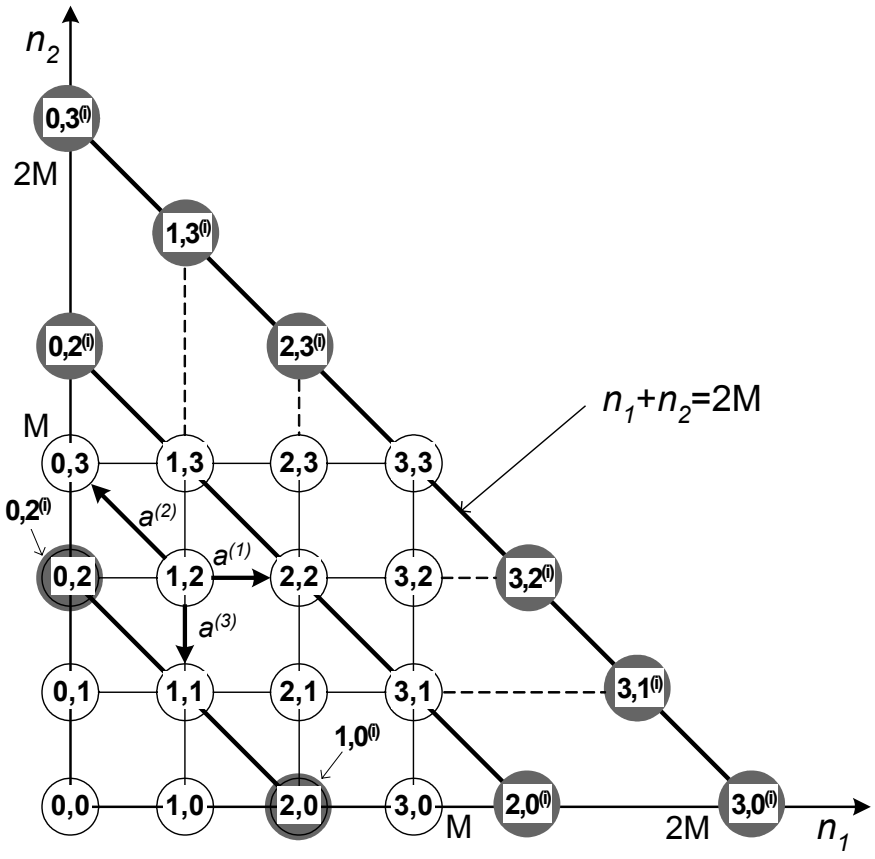


Fig. 4. The embedded 2D DTMC with finite and infinite first queue

Thus, self-organizing map learns the topology of weight vectors, which it trains on for receiving of distribution into cluster zones, corresponding to the steady states of the embedded DTMC.

3 Probability Density Function with Neural Networks

The Kohonen networks solve classification problems in which the probability density function is unknown in advance. With vector quantization are estimated non-parametric classification distribution procedures, which can be used without the assumption, that the form of the underlying densities are known [5], [10]. Kohonen's network algorithm provides a tessellation of the input space into patches with corresponding code vectors. It has an additional feature that the centers are arranged in a low dimensional structure (usually a string, or a square grid), such that nearby points in the topological structure (the string or grid) map to nearby points in the attribute space. The Kohonen learning rule is used when the winning node represents

the same class as a new training pattern, while a difference in class between the winning node and a training pattern causes the node to move away from training pattern by the same distance. In training, the winning node of the network, which is nearest node in the input space to a given training pattern, moves towards that training pattern, while dragging with its neighboring nodes in the network topology. This leads to a smooth distribution of the network topology in a non-linear subspace of the training data. Recall that M, n, N denotes the number of classes, of input samples and attributes, respectively. Classes will be denoted by $S_1,\ldots,S_M$ and attribute values for input samples (p=1,2,…,n) will be denoted by the N-dimensional vector $\mathbf{x}^{(p)} = \left\{X_1^{(p)},\ldots,X_N^{(p)}\right\}$. The classification procedure is linked with the estimation of the local probability density at each target class and contains the kernel density method.

3.1 Density Estimation

For density estimation is proposed a nonparametric approach, where we have to estimate the densities $F_k(\mathbf{x})$, k=1,2,…,M. To introduce the kernel density, is made assumption that we have to estimate the N – dimensional density function $F(x)$ of an unknown distribution. This process is performed for each of the M densities $F_k(x)$, k=1,2,…,M. Let $\hat{F}(\mathbf{x})$ is the estimate of $F(\mathbf{x})$ as an average function of $\mathbf{x}$ and the input samples $\mathbf{x}^{(p)}$. In general can be used (12),

$$\hat{F}(\mathbf{x}) = \frac{1}{n}\sum_{p=1}^{n} K(\mathbf{x}, \mathbf{x}^{(p)}, \lambda_n) \qquad (12)$$

where $K(\mathbf{x}, \mathbf{x}^{(p)},\lambda_n)$ are kernel functions and λ_n is the length of the edge of the n-dimensional hypercube. If λ_n is very large, than the kernel $K(\mathbf{x}, \mathbf{x}^{(p)},\lambda_n)$ changes very slowly with $\mathbf{x}$ and as a result the estimation of $F(\mathbf{x})$ is very smooth. For solving of the classification problem is made a choice for density estimation via specification of the kernel and the value of the smoothing parameter.

The kernel function can be restricted to the kernels with N independent coordinates. It is presented according to (13),

$$K(\mathbf{x}, \mathbf{x}^{(p)}, \lambda) = \prod_{j=1}^{N} K_j(\mathbf{x}_j, \mathbf{x}_j^{(p)}, \lambda) \qquad (13)$$

where K_j indicate the kernel function component of the j^{th} attribute and λ is average smoothed length of the hypercube and is not dependent on j. It is clear that kernels could have a more complex form and that the smoothing parameter could be coordinate dependent. The kernels depend on the type of variables. If are used two-dimensional coordinates, the kernel function is presented by (14) and boundaries of target classes are edges of rectangular or square.

$$K_j(\mathbf{x}_j, \mathbf{x}_j^{(p)}, \lambda) = \frac{1}{1+\lambda}\lambda^{\left(\mathbf{x}_j - \mathbf{x}_j^{(p)}\right)^2} \qquad (14)$$

3.2 Two-dimensional Learning Vector Quantization

The main goal of a learning neural model for vector quantification is to determine the probability density function for each steady state of two-dimensional embedded DTMC set.

The probability distribution of the input attribute space is determined with two neural layers – first, competitive and second, linear layer. The competitive layer is based on a set of input/target pairs (15),

$$\{\mathbf{x}_1, \mathbf{C}_1\}, \{\mathbf{x}_2, \mathbf{C}_2\}, \ldots, \{\mathbf{x}_j, \mathbf{C}_j\}, \ldots, \{\mathbf{x}_N, \mathbf{C}_N\} \tag{15}$$

with the help of which is trained the neural network. Here $\mathbf{x}_j$ are two N-dimensional input vectors, and the M- dimensional vector $\mathbf{C}_j$ describes the condition of target classes, presented at (16).

$$\begin{aligned} \mathbf{x}_j &= \{X_j^{(1)}, X_j^{(2)}\}, \qquad j = 1,...,N \\ \mathbf{C}_j &= \{S_1, S_2, ...S_k, ...S_M\}, \quad k = 1,...M \end{aligned} \tag{16}$$

The hidden neurons from first layer compete via initializing of the weight matrix $\mathbf{W}_{kj}$ and are determining the winner. This is the neuron, which has minimal Euclid distance d_k to the input vectors $\mathbf{x}_j$. Then the corresponding target class receives value 1 and the rest target classes receive 0, as is presented by (17).

$$\begin{aligned} &S_k = 1, \quad for \quad d_k^{\min}, \qquad d_k = \sum_{j=1}^{N} \left(X_j - W_{kj}\right)^2 \\ &S_k = 0, \quad otherwise \end{aligned} \tag{17}$$

The neuron-winner has feedback negative links to the rest of neurons and strong positive link to himself, which is used for learning in linear layer. During the training in the next epoch q are changing the coefficients of all neurons according to Kohonen learning rule [6], which is summarized at (18).

$$W_{kj}(q) = W_{kj}(q-1) \pm \xi\left(X_j(q) - W_{kj}(q-1)\right), \qquad 0 < \xi \le 1 \tag{18}$$

The coefficient ξ depends on the number of training epochs q and can be adjusted in advance in interval [0,1], where standard is determined equal to 0,1. The sign before the training coefficient ξ is positive for the neuron-winner, and negative for the neighbor neurons. As a result during the process of the training is changed the area of neighbor neurons for the neuron-winner, e.g. decreases the Euclid distances.

As an illustrative numerical example on Fig. 5 is shown the training of two-dimensional embedded Markov chain with LVQ. A model of partial overlap single link transmission is under study, where are used two parallel single queues for estimation of link occupancy distribution. On the input of the queue n_1 is given vector $\mathbf{x}^{(1)}$, which contain 600 discrete random variables in interval (0,70) which are distributed according to exponential density distribution. On the input of the queue n_2 is given vector $\mathbf{x}^{(2)}$, which contain 600 discrete random variables in interval [0,50) which are distributed according to Poisson density distribution. Both queues are

modelling a small single link with capacity 24 Bandwidth Units (BU). Let the peak bandwidth requirements for the traffic flow of n_1 are b_1 =5 BU and for n_2 are b_2 =7 BU. The traffic flows are characterized as well with minimal accepted bandwidth, which are determined as $b_1^{\min}$ = 2,8 BU and $b_2^{\min}$ = 4,2 BU. At the same time the link cut-off parameters are limited to N_1 = 4 and N_2 = 2. This partial overlap allocated strategy gives possibility to use an embedded discrete Markov model with 15 feasible steady states (5 for n_1 and 3 for n_2).

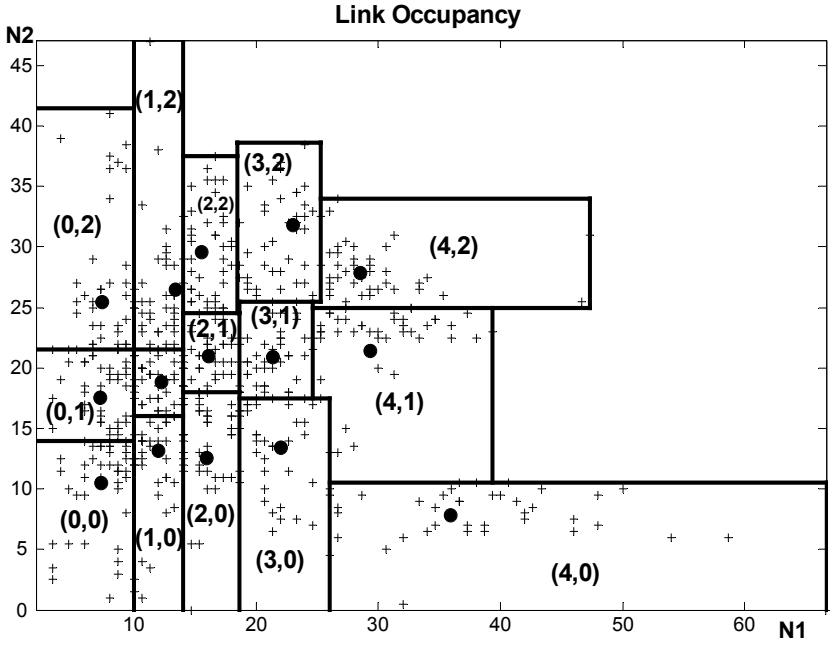


Fig. 5. An example for 2D Learning Vector Quantization

The clustering procedure consists of two stages – VQ with equal number of target values in each class and optimization of the weights of cluster classes via learning. As a result from VQ the input space is divided into 15 cluster classes, which total number of target values in each class equal to 40. The boundaries, which are shown with thick lines on Fig. 5, are determined according to (14) and limited the rectangular faces of cluster classes.

The process of bipartition is realizing in such a manner, that at first partition under consideration is one class and the rest of values are in another class. The weight vectors for both classes are determining via training of neural network. These "rest values" are dividing into two classes and again are determining weight vectors via training, which procedure is repeating until finishing number of classes.

The results from training of the weight neurons for each of 15 target classes with 1000 epochs and ξ = 0,1 are shown with black points on Fig.5. The role played by the discrete random values of input vectors and trained epochs is clear. Their very large number leads to improving the model estimation and model's weight characteristics.

The occupancy density probability distribution for each of target classes, received with LVQ network is shown on Fig. 6.

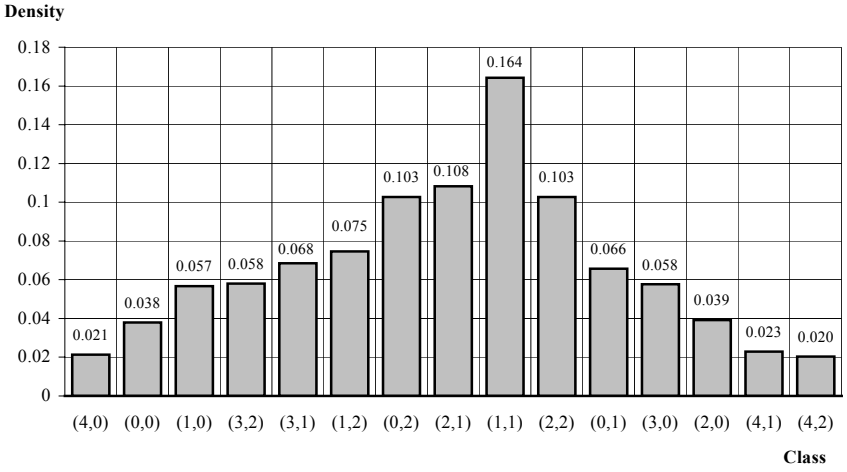


Fig. 6. Probability density of model's feasible states

The classes are ordered according their density, that's why they aren't consecutive. It is clear, that at classes (4,0) and (4,2) there are minimal density and at classes (1,1) the density is maximal. This example illustrated possibilities of non-supervised neural learning for steady state analysis and estimation of two-dimensional discrete time Markov Chain models in queuing systems.

4 Self-organizing Map for Two-node Tandem Queue

As an input of a two-dimensional self-organizing map is used two *N*-dimensional input vectors $\mathbf{x}_j$, determined according to (16). In 2D output space is expected a map, corresponding to the *M*- dimensional array of output neurons $\mathbf{C}_i$, which can be one or two-dimensional. The connection between inputs and outputs is realized with the weight matrix $\mathbf{W}_{ij}$. At competitive learning for winner is selected the output neuron i^*, which weight vector is closer to the current input according to (19).

$$\left|W_{ij}^{*} - X_j\right| \le \left|W_{ij} - X_j\right|, \qquad \forall j \in \left[1,...N\right] \tag{19}$$

The Kohonen learning rule is differ from vector quantization rule and is determine by (20).

$$\Delta W_{ij} = \xi \wedge \left(i, i^*\right)\left(X_j - W_{ij}\right), \qquad 1 \le i \le M, \qquad 1 \le j \le N \tag{20}$$

The neighborhood function $\wedge(i, i^*)$ is equal to 1 for $i = i^*$, and decreases with increasing of distance between neurons i and i^*in input array. The neurons closer to the winner i^*, changes their weights more quick than remote neurons, for which the neighborhood function is very small. The topological information contents in the fact, that closer neurons are changing almost in the same way and in this manner corresponds to neighbor input samples. The learning rule (20) attracts the winner's weight vector to the point X_j. The self-organizing map is supposed to be an elastic set in input space, which wont to be moved maximal closer to the input values. The set has topology of input array and it points have as coordinates weight vectors.

For optimizing of clustering procedure a minimum-squared-error algorithm is used. Suppose that there are given a dataset **x** of points in some Banach space, partition the data into k clusters such that some empirical loss function to minimize is (21).

$$D(x)=\frac{1}{n}\sum_{j=1}^{k}\sum_{i}^{n_j}\left(\left\|x_{ij}-s_j\right\|\right)^2; x_{ij}\in C_j, \qquad x_{ij}=x_i I_{\{x_i\in C_j\}}, \qquad \sum_{j=1}^{k} n_j = n \tag{21}$$

Here dataset points belong to a d-dimensional Euclidean region ($d \geq 2$), C_j denotes the j-th cluster, n_j denotes the number of point x_i in C_j and I is the indicator function of $\{x_i \in C_j\}$. The mean vectors and the criterion function are updated after each pattern move. These approaches guarantee local but not global optimization. Different initial partitions and sequences of the training patterns can lead to different solutions.

One example for implementation of two-dimensional self-organizing map for determination of weighs vectors, which corresponds to steady states of DTMC, is presented in this chapter. A tandem Jackson queue is under study, with infinite first queue n_1. The condition for proper work of the network which has a buffer with capacity M=100 cells, e.g. $n_1+n_2 \leq M$. The embedded DTMC model of this network has 16 steady states, which are determined of arrival rate λ and holding time services μ_1 and μ_2 of the first and second queues respectively. Let λ and μ_1 and μ_2 are continuous in time and buffer content in n_1 and n_2 has Gamma and exponential distribution respectively.

The problem for this queuing network is that the probability density distribution is impossible to be determined with two-dimensional VQ, and that's why is impossible to implement LVQ neural network. One possible solution of this problem is shown SOM on Fig. 7. On the input of the queue n_1 and queue n_2 are given 600 discrete random variables in interval [0,100].

Received one-layered neural network defines 16 output neurons in such a manner that four of them are on axial coordinate n_1 and the rest four are placed on axial coordinate n_2. The starting input neurons are depicted with squared black points. Their topology differs significant after 10 000 epochs of training and it is shown with circle black points, which are connected with lines. As is seen, instead of updating only the winner, the map updates the weights of the winner and its neighbors. The result is that neighbouring neurons tend to have similar weigh vectors and to be responsive to the topology of input data.

The receiving of neural map corresponding to the two-dimensional discrete Markov model is a good basis for further classification of steady states. With the help of (21) easy can be determined target values from surrounding area of each weigh

vector. As a result, boundaries of target classes can be approximated via geometric figures and can be calculated probability density distribution in target classes.

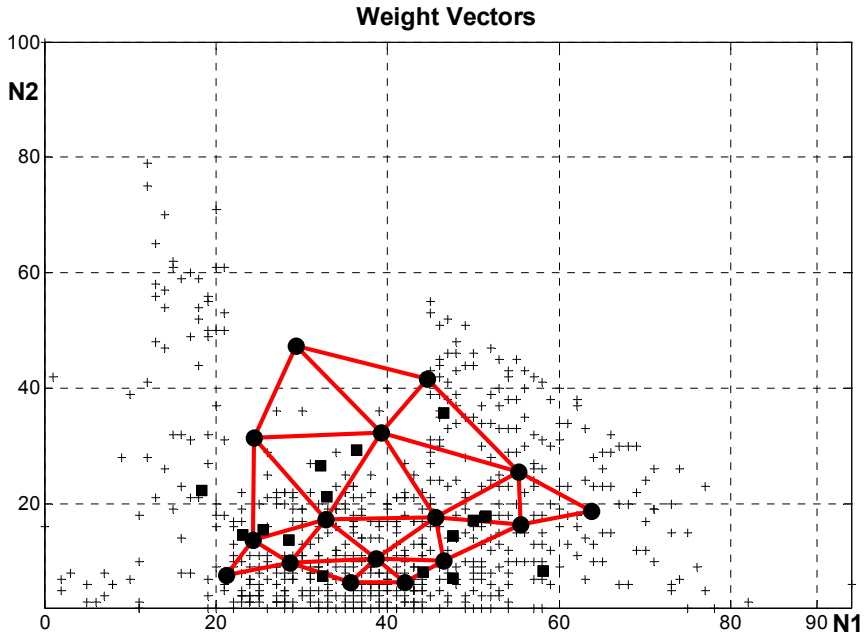


Fig. 7. Self-organizing map of embedded 2D DTMC with infinite first queue

In any case, such solution is much more correct than using of LVQ with rectangular boundaries of target classes.

5 Conclusions

In this study are shown simulation techniques for implementation of Kohonen networks in two-node queuing systems. The two-dimensional models of discrete time Markov chains are presented for parallel and tandem queues, which have implementation in packet-switching communication networks.

An approach for LVQ of discrete models is suggested with the help of Kohonen neural network. This network via proper determined learning and training parameters gives possibility for qualitative estimation of probability density distribution for Markov chains in their steady state nodes at variety of random discrete distribution of arrival processes and holding time services.

The presented approach can be useful for investigation of the entrance distributions, link occupancy distribution, homogeneous continuous arrival processes, remaining time services phases, overflow probability, consecutive cell loss performance, packet delivery services analysis and other performance parameters of queuing systems.

The developed simulation procedure for SOM implementation at two-dimensional queuing models is suitable for determining of feasible steady states topology for models with unknown in advance clustering. An example for creating of steady state model topology at tandem queues with infinite first queue is shown. Simulation and numerical results for implementation of SOM and LVQ networks in digital communication nets with Poisson, Gamma and exponential distributions of arrival and service processes are shown.

References

1. Borkar, V. S.: On the Lock-in Probability of Stochastic Approximation. Combinatorics, Probability and Computing, 11 (2002) 11–20
2. Borst, S. C., Mitra, D.: Virtual Partitioning for Robust Resource Sharing: Computational Techniques for Heterogeneous Traffic. IEEE Journal on Selected Areas in Comunications, vol. 16, 5 (1998) 668–678
3. Bourland, H., Wellekens, C. J.: Links between Markov Models and Multilayer Perceptrons. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12 (1990) 1–12
4. Fazekas, P., Imre, S., Telek, M.: Modeling and analysis of broadband cellular networks with Multimedia Connections. Telecommunication Systems, vol. 19, 3-4 (2002) 263–288
5. Hertz, J., Krogh, A., Palmer, R.: Introduction to the Theory of Neural Computation. Addison-Wesley, New York (1991)
6. Kohonen, T.: Self-Organizing Maps, Second Edition, Springer Verlag, Berlin (1997)
7. Kollman, C., Baggerly, D., Cox, D., Picard, R.: Adaptive Importance Sampling on Discrete Markov Chains. Annals of Applied Probability, 9 (1999) 391–412
8. Kroese, D. P., Nicola, V. F. Efficient Simulation of Overflow Probabilities in Queues with Breakdowns. Performance Evaluation, 36 - 37 (1999) 471–484
9. Michie, D., Sammut, C.: Machine Learning from Real-time Input-output Behaviour. Proceedings of the International Conference Desing to Manifacture in Modern Industry, Digit. Libr. 1 (1997) 108–121
10. McLachlan, G.J.: Discriminant Analysis and Statistical Pattern Recognition. John Wiley, New York (1992)
11. Radev, D., Lockshina, I., Radeva, S.: Neural Modelling of Link Occupancy Distribution for Broadband Telecommunication Transmission. 35^{-th} Annual Meeting of the Decision Sciences Institute, Boston, USA, (2004), under press
12. Radev, D., Radeva, S.: Artificial intelligence modelling of stochastic processes in digital communication networks. Journal of Electrical Engineering, vol. 54, 9-10 (2003), 255-259
13. Rubinstein, R. Y., Melamed, B.: Modern Simulation and Modelling. John Wiley Series in Probability and Statistics, New York (1998)
14. Shwartz, A., Weiss, A.: Large Deviations for Performance Analysis. Chapman and Hall, New York (1995)

A Grid-based Application of Machine Learning to Model Generation

Volker Sorge[1,*], Simon Colton[2,**], Andreas Meier[3,**], and Roy McCasland[4]
[1] School of Computer Science, University of Birmingham, UK,
V.Sorge@cs.bham.ac.uk, http://www.cs.bham.ac.uk/~vxs
[2] Department of Computing, Imperial College London, UK,
sgc@doc.ic.ac.uk, http://www.doc.ic.ac.uk/~sgc
[3] DFKI GmbH, Saarbrücken, Germany,
ameier@dfki.de, http://www.ags.uni-sb.de/~ameier
[4] School of Informatics, University of Edinburgh, UK
rmccasla@dai.ed.ac.uk, http://www.dai.ed.ac.uk/~rmccasla

Abstract. The classification of mathematical structures is a driving force in pure mathematics. A first step in producing algebraic classification theorems is to determine for which sizes certain algebras exist. Computational approaches to solving such existence problems using constraint satisfaction and model generation approaches have had much success. We look here at the question of distributing the model generation process using Grid technology. We present a novel distribution approach which involves using the HR machine learning program to intelligently suggest specialisations of the problem which are given to separate processors. Using the MACE, FINDER and SEM model generators, we demonstrate how this approach provides greater efficiency over a single-process approach for a series of quasigroup existence problems. We compare several approaches for the production and choice of specialisations, including the generation of proved classification theorems for algebraic structures of small sizes. We discuss how this approach could be used for more general problems.

1 Introduction

The classification of finite simple groups in 1980 was one of the major intellectual achievements of the twentieth century. Often, the first step towards such algebraic classification theorems is to determine how many structures exist up to isomorphism for each size. In particular, it is instructive to determine for which sizes algebras exist at all. Such existence problems have also attracted much attention in the research fields of automated reasoning and constraint solving. For instance, solving open quasigroup existence problems has become a challenge for friendly competition [18, 20], and such problems are used for benchmarking AI systems.[1] Existence problems are often solved by the construction of an algebra of the given size. If asked to perform the same task, a mathematician would probably not attempt to construct a multiplication table as a constraint solver or model generator would. Instead, they might determine the families which include a model of the given size, and attempt to write down the correct member of a particular family. For instance, writing down a multiplication table for a group of

* The author's work was supported by a Marie-Curie Grant from the European Union.
** Author's work supported by EU IHP grant Calculemus HPRN-CT-2000-00102.
[1] For example, see prob003 of the CSPLIB library (http://www.4c.ucc.ie/~tw/csplib).

size 17 is trivial, because only one group exists of this size and this group is cyclic, so its multiplication table consists of the seventeen possible cycles of the elements. Constructing a size 17 multiplication table from the axioms alone, however, is not trivial.

In the case of cyclic groups, a constructive theorem exists which describes how to build the models. The existence of such a theorem is not true in the general case. However, we can still propose a middle ground between the computational (brute force) approach and the more mathematical approach suggested above. This involves automatically specialising the model generation problem in the hope that one of the specialised – more constrained – sub-problems will be easier to solve. Such an approach may work if the existence problem can be solved positively, i.e., by actually finding a solution. The specialisations are generated via an un-supervised machine learning approach which uses smaller models that satisfy the axioms of the algebra to intelligently suggest properties to specialise the problem. For instance, if the problem was to produce a group of size 12, a model generator would be used to find groups of size 1 to 11, which become the input to an un-supervised machine learning system. The system would recognise that many of the models given to it are *Abelian*, i.e., $\forall x_{\blacksquare} \forall y_{\blacksquare}\ (x \circ y = y \circ x)$. Hence this suggests specialising the problem of finding a group of size 12 into the problem of finding an Abelian group of size 12, a much easier task.

We compare two machine learning approaches for automatically specialising model generation problems, including a sophisticated method that involves constructing and automatically proving (using automated theorem proving) classification trees for deciding the isomorphism family of algebras of given sizes. Both specialisation procedures are capable of producing large numbers of suitable sub-problems. This provides an opportunity for further efficiency gains by distributing the sub-problems over multiple processors. To gain the maximum efficiency possible, we have employed Grid technology to organise the distribution of processes. As a third specialisation method we also employed a simple method for distributing the problem, namely by instantiation variables. All specialisation methods are integrated into a uniform framework.

To demonstrate the effectiveness of our distribution approach, we performed a series of experiments using a standard problem set: QG-quasigroups. Using the MACE [15], FINDER [17] and SEM [21] model generators, we compared the three specialisation methods against a single-processor approach which uses the axioms alone. We show that in most cases, in particular, for larger sizes, the classification method for specialising into sub-problems produces substantial gains in efficiency over the other approaches, when distributed using Grid technology.

The paper is organised as follows. In section 2, we provide necessary background information, including a description of the problem domain (QG-quasigroups) and descriptions of the constituent AI programs in our system. In section 3, we provide an overview of how our system operates, including details of the Grid technology employed. In section 4, we provide details of the three methods employed to specialise the model generation problems. Following this, in section 5, we describe the experiments we performed in order to test the efficiency of the system and the results we obtained. Finally, in sections 6 and 7 we place this work in context and conclude by looking at the potential of this approach for more general problems.

2 Background

2.1 Problem Domain

Finite algebras describe a set of elements and a set of operators which each define a function taking a subset of elements to a boolean or a subset of elements, e.g., the multiplication function, which takes a pair of elements to a third element. The *axioms* of the algebra dictate properties of the operators which identify the algebra. Quasigroups – also known as Latin Squares – are algebras over a single multiplication operator, $\circ$, which is subject to a single axiom: $\forall a. \forall b. \ (\exists x. x \circ a = b) \wedge (\exists y. a \circ y = b)$. This corresponds to the property that every element appears in every row and every column of the multiplication table. As an example consider the following five quasigroups of order 3, which constitute representants of the five different isomorphism classes:

A_2	a	b	c
a	b	a	c
b	a	c	b
c	c	b	a

A_4	a	b	c
a	a	c	b
b	c	b	a
c	b	a	c

A_6	a	b	c
a	a	b	c
b	b	c	a
c	c	a	b

A_7	a	b	c
a	a	b	c
b	c	a	b
c	b	c	a

A_8	a	b	c
a	a	c	b
b	b	a	c
c	c	b	a

The discovery of quasigroups with certain properties is interesting outside of pure mathematics, because the underlying structure of quasigroups is similar to that found in many real-world applications, such as scheduling and timetabling, experimental design, error correcting codes, and wavelength routing in fibre-optic networks [12, 13]. Constraint satisfaction and model generating methods have been applied to decide quasigroup existence problems with much success (see section 6). Moreover, such problems have become benchmark domains for model generators and constraint solvers. Of particular interest are the so-called 'QG'-quasigroup families of problems [18] which algebras satisfying the quasigroup axiom and one of these additional axioms:

[QG1] $\forall a. \forall b. \forall c. \ (a \circ b) \circ b = (a \circ c) \circ c \rightarrow b = c$
[QG2] $\forall a. \forall b. \forall c. \ (b \circ a) \circ b = (c \circ a) \circ c \rightarrow b = c$
[QG3] $\forall a. \forall b. \ ((a \circ b) \circ (b \circ a) = a)$
[QG4] $\forall a. \forall b. \ ((b \circ a) \circ (a \circ b) = a)$
[QG5] $\forall a. \forall b. \ (((b \circ a) \circ b) \circ b = a)$
[QG6] $\forall a. \forall b. \ ((a \circ b) \circ b = a \circ (a \circ b))$
[QG7] $\forall a. \forall b. \ ((b \circ a) \circ b = a \circ (b \circ a))$

We are interested here in the following problem setting: given a specification of a (finite) algebra in the form of a set of axioms A, construct an algebra of given finite size n which satisfies these properties? Naturally, solving such problems also solves the question of whether an algebra of size n satisfying A exists.

2.2 Artificial Intelligence Systems Employed

The following Artificial Intelligence system were used in our experiments:
• The MACE [15] model generator first constructs all instances of the given axioms over the given finite domain and then employs a decision procedure that searches for models, i.e., satisfiable instances of the formulas.

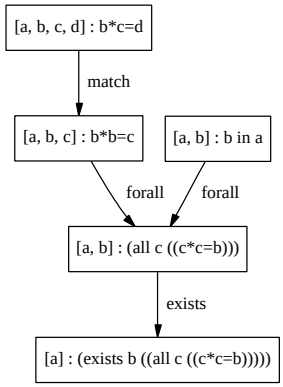


Fig. 1. Example construction by the HR System

- The SEM [21] model generator also first constructs all instances of the given axioms over the given finite domain. Then, it performs a heuristic search over the possible interpretations on the instances to find a model.
- The FINDER [17] model generator performs an exhaustive search for interpretations of the given axioms, using the axioms to direct its backtracking. Thereby, first a candidate model is generated and tried against all the axioms together. If all axioms are true, then the candidate model is accepted. If one of the axioms is false, the candidate model is adjusted to deal with the detected badness, resulting in a new candidate model.
- HR [1] is a machine learning program which performs automated theory formation by building new concepts from old ones using a set of production rules, and using measures of interestingness to drive the search [2]. It uses the examples of the concepts to empirically form conjectures relating their definitions and employs third party theorem proving and model generation software to prove/disprove the conjectures. In particular, for the application discussed in this paper HR used the following four production rules:

`Compose:` this composes functions using conjugation.
`Match:` this equates variables in predicate definitions.
`Exists:` this introduces existential quantification.
`Forall:` this introduces universal quantification.

As an example construction, consider the concept of there being a single element on the diagonal of the multiplication table of an algebra. This concept is constructed by HR using the match, forall and exists production rules, as depicted in Fig. 1. In this scenario, two concepts are supplied by the user, namely the concept of an element of the algebra and the multiplication of two elements to give a third. Using the match production rule with the multiplication concept, it invents the notion of multiplying an element by itself. By using this in the forall production rule, it invents the concept of elements which all other elements multiply by themselves to give. Then, using the exists production rule, HR invents the notion of algebras where there is such an element.

HR has a variety of different search strategies for employing the production rules. For the application described here, we employed a 'tiered' search for concepts, whereby the `Match`, `Exists` and `Forall` production rules were used greedily, with the compose rule used only when no further steps with the other rules were available. Al-

ternatively, HR can use heuristic searches whereby it employs a variety of measures of interestingness to assess the worth of a concept, then chooses the best concepts to build new concepts from [2]. For more details of how HR forms concepts, see [3]. In addition to forming concepts, HR makes conjectures about the concepts and uses third party software to prove/disprove these conjectures, details of which are described in [1], but are beyond the scope of this paper.

3 System Overview

In model generation, while solutions to small problem instances, e.g., QG5 quasigroups of size 5, are easily found, the search spaces explode combinatorially, and their sizes prohibit the systems from finding models for larger problems unless the problems are sufficiently constrained. Therefore, dividing the search space into more tractable parts is crucial for problems involving structures of a larger size. A division of the search space can be achieved by adding auxiliary properties to the problem in order to specialise the original axioms thus reducing the search space. In order not to change the problem itself it is necessary to divide the search into two parts: one with the additional property and another one with its negation. This ensures that the union of both search spaces will still contain all possible solutions.

When searching for a single model which satisfies the axioms, the hope is that it can be found faster in one of the two subspaces, than in the search space of the original problem. Thus, if two processors are available, parallelising the search into two separate model generation processes should produce a result faster than for the original problem. Furthermore, if we can employ a set of such specialising properties, a subsequent division into several parallel model generation processes should result in an overall speed up. The input to our system is a quadruple $\langle A, N, S, M \rangle$, where A is the set of axioms describing the algebra we are interested in, N is the order (size) of the algebra we want to construct a model for, S specifies the type of specialisation method to be employed and M is the model generator to use (SEM, MACE or FINDER).

Given this input, the system first generates a set of specialisation sub-problems P which consist of using model generator M to find models of size N which satisfy axioms A and an additional set of properties. The number and nature of the sub-problems depends on the specialisation process employed, as discussed in section 4. The specialisations used to attack the problem of, say, generating size 10 QG5-quasigroups can also be used for the problem of generating size 12 QG7-quasigroups. Hence, the specialisation procedure usually amounts to looking up the set of pre-determined specialisations to use, which have been generated automatically. Once the sub-problems P have been generated, for each P_i in P, model generator M is used to try to find a solution to P_i.

As discussed above, the benefit of the heuristic to constrain the model generation can be greatly improved if we distribute the search. Our approach lends itself well to distribution over a set of processors: each specialisation sub-problem can be run on a different processor. In order to achieve maximum distribution, we employ grid technology, which ensures that the processes are evenly distributed over a cluster of processors, as a single processor is assigned exclusively to each process. It also keeps the communication time to a minimum and free from interfering network traffic by

using the cluster specific interconnection facilities. The distribution itself is relatively straightforward: for each specialised axiom set a separate model generation process is created and distributed on the grid. When a process terminates, the result is checked, and if it has resulted in a solution, all other model generation processes are terminated immediately, since we have successfully solved the existence problem. If it does not contain a model, the other processes continue to run. Thus, if applied to a problem for which no model exists, all processes have to run until their search space is exhausted.

As discussed in section 5, we compare this approach with a single process approach that uses the original axiom set, hence covers the entire search space. This is run at the same time as the sub-problems, and is given the same status on the Grid as the specialisation processes. This has an additional benefit: for problems which have no solution, we have found that this process often finishes quicker than the set of specialisation sub-problems. Obviously, if the entire search space is traversed and no model is found, then no specialisation problem will be successful, and hence every process can be halted.

4 Generating Specialisation Concepts

An important process in our system is the way in which it decides how to specialise the original problem into a set of sub-problems for distribution. It is possible that adding properties to a set of axioms can increase the search space. In particular, if properties involve existentially quantified variables, the search for models of Skolem terms can enlarge the search space rather than reduce it. Even when the specialisation reduces the search space, we need to take into consideration the *constrainedness knife-edge* [9]: adding too few constraints can leave the problems unsolvable in a reasonable time, yet adding too many constraints may over-constrain the search space. Bearing these considerations in mind, we adopted an experimental approach where three methods for generating specialisations are compared and contrasted:

- One possibility is to split the search space into subspaces of roughly equal sizes, and we look at a simple way to do this in section 4.1.
- A second possibility is to more intelligently generate properties using machine learning procedures to generate and choose the properties. We discuss how the HR system can be used for this purpose in section 4.2.
- A final possibility is to attempt to place sub-problems onto the knife-edge, whereby the problem is constrained enough to be solvable in a reasonable time, yet not over constrained. We look at a method which uses classification trees to attempt to do this in section 4.3.

4.1 Instantiating Variables

As a simple way of specialising the model generation problems, we instantiated variables as follows: given a problem to generate an algebra of size n satisfying axioms A, the first specialisation has axioms A with the property that $0*0 = 0$, the second specialisation has axioms A with the property $0 * 0 = 1$ and so on, with the nth specialisation being axioms A with the property $0*0 = n-1$. In addition, further specialisations with

properties $0 * 1 = x$ (with $x \in \{0, \ldots, n-1\}$) are possible. This acts as a control in our experiments – it may be that the distribution alone increases the efficiency, and not the specialisation procedures we have implemented, which would mean that this method would be competitive with the specialisation procedures.

4.2 Using Applicability

One approach to dealing with the constrainedness knife-edge is to be cautious, and choose properties which are not likely to over-constrain the search space. As discussed above, HR can generate numerous concepts which specialise the notion of algebra, and it can choose from these using various measures of interestingness [2]. To do this, the system takes the axioms of the algebra under investigation and uses the model generator specified to generate models of size 1, 2, 3, etc. up until the model generator fails to find a model in a given time. These are used as the objects of interest in a theory formation session with HR, and HR's search is adjusted to favour the production of specialisation concepts. In addition to working with smaller algebras satisfying exactly the same axioms as the problem we are looking at, we can employ more general or more specific axiom sets for property generation.

Once the theory has been produced, the *applicability* measure is used to extract the best specialisations with respect to that measure. The applicability of a concept is measured as the proportion of models which satisfy the concept. The idea is that, when being cautious, it is better to choose specialisations with high applicability, as, given the evidence of smaller sizes, it is unlikely that these will over-constrain the search space. In practice, applicability is used via a threshold: given a theory formed by HR which contains specialisation properties $\{P_1, \ldots, P_m\}$, the system chooses a subset of these concepts as possible constraining properties by considering those concepts that hold for at least a certain, specified number of the given structures. Given a threshold $0 < t \leq 1$, only those properties are considered that actually hold for at least $t \cdot n$ structures.

For instance, if $t = 0.5$ all properties are chosen that hold for at least half the models. The system then takes the set of chosen properties $\{Q_1, \ldots, Q_l\} \subseteq \{P_1, \ldots, P_m\}$ and generates sets of axioms that will be employed to generate models. Suppose we are looking for structures that satisfy a particular set of axioms $\mathcal{H}$, the system will then generate the sets of axioms of the form $\mathcal{H} \cup \{Q_1\}$, $\mathcal{H} \cup \{Q_2\}$, $\ldots$, $\mathcal{H} \cup \{Q_l\}$, and $\mathcal{H} \cup \{\neg Q_1 \vee \neg Q_2 \vee \ldots \vee \neg Q_l\}$. These $l + 1$ axiom sets are then submitted to the model generators. Any structure found will naturally also be a model of the original axiom set $\mathcal{H}$. As an example, when considering existence problems of QG5 quasi-groups, to generate properties with HR, the system generated a set of QG5 quasigroups of size 3 to 9. HR then came up with 68 properties of which 19 were chosen as they hold for more than half of the quasigroups. These resulted in 20 sub-problems being given to the model generators.

4.3 Using a Classification Algorithm

Being cautious with the addition of specialising properties may mean that the sub-problems are still under-constrained and take too long to solve. One way to reduce the search space without over-constraining is to reduce symmetry: two solutions which

may be syntactically different, may be considered by the user as really the same, and adding constraints which rules out one of these solutions without ruling out the other is said to break the symmetry. In algebraic domains, the main cause of symmetry is *isomorphism*. Two algebras are considered isomorphic if a permutation of the elements of one means that it can be re-written to look exactly like the other one. Hence, when generating algebras, we could consider a problem on the constrainedness knife-edge if it was guaranteed to find only solutions in a particular isomorphism class. Some computationally tractable techniques exist for determining isomorphism types – e.g., for groups [16] – but in general, it is as difficult to do this as it is to exhaust the space of models for a given size. However, we can approximate the properties required for isomorphic specialisation of the problem size of interest, by looking at properties which can be used to classify smaller sized algebras up to isomorphism.

In [4], we present a bootstrapping algorithm for generating theorems which enable the classification of algebras up to isomorphism. As a simple example, the algorithm is given the axioms of group theory and told to find a classification theorem for groups of size 6. It returns the following (paraphrased) result: "all groups of size 6 can be classified up to isomorphism as either Abelian or non-Abelian". The implementation of this algorithm uses HR, MACE, SEM, FINDER, the automated theorem prover Spass, and the Gap computer algebra system [8] to generate such results. Note that Spass is used to show that the theorems provide valid classifications by proving that each concept in the theorem is a *classifying property*, i.e., true for all members of exactly *one* isomorphism class.

The general idea of the algorithm is to construct a decision tree by successively generating non-isomorphic algebraic structures and associated discriminants until all isomorphism classes have been found. A discriminant property P for two non-isomorphic structures A_1 and A_2 is a property that is invariant under isomorphism such that P does hold for A_1 but not for A_2 or vice versa. That is, the existence of P for A_1 and A_2 is a proof that A_1 and A_2 are not isomorphic. The algorithm takes the size n and the axioms of the algebraic structures to be considered as input. It returns a decision tree for isomorphism classes of the algebraic structure that serves as classification for the specified algebras up to isomorphism for a given cardinality.

For example, Fig. 2 gives the decision tree constructed for the classification of quasigroups of order 3. The edges of the tree are labelled with properties. Each node in the tree is associated with the conjunction of the properties that are the labels of the edges of the path leading from the root node to the node. For instance, node 4 is associated with the property $P_1 \wedge P_2$. The tree shows 5 isomorphism classes for quasigroups of size 3 corresponding to the leaf nodes 2, 4, 6, 7, and 8, which are called the isomorphy nodes of the tree. For isomorphy nodes, the property associated with the node uniquely characterises an isomorphy class of quasigroups of size 3. That is, all quasigroups of size 3 satisfying the property associated with an isomorphy node are isomorphic. The representants of the isomorphism classes are the quasigroups given in section 2.1 where the enumeration of the quasigroups corresponds to the labels of the respective isomorphy nodes, e.g., A_2 is the representant of the isomorphism class given by node 2.

Initially, the decision tree consists only of the root node. The single nodes of the tree are expanded by first generating an algebraic structure satisfying the properties specified by the node. For example, for the root node 1 in Fig. 2, an arbitrary quasigroup

of order 3 is constructed; for node 3, however, a quasigroup that additionally satisfies the property $\exists b_{\blacksquare} b \circ b = b$ is required. If no representant can be generated, we must prove that there exists no algebraic system of cardinality n satisfying both the original axioms and the additional properties generated up to this point.

When a representant is generated we have two cases to consider: there exists a non-isomorphic structure exhibiting the same properties considered so far or the property represented by the node constrains the structures to a single isomorphism class. In the former case, we have to branch further by generating a structure non-isomorphic to the original one. The two structures are then passed to HR to compute two — generally the smallest — discriminants. In the example, the quasigroup A_4 given in section 2.1 was constructed as a non-isomorphic counterpart to A_2. Given those two quasigroups, HR came up with discriminants P_1 and $\neg P_1$ as stated in Fig. 2. Depending on the nature of the discriminants, either two or four child nodes are constructed. The case of two child nodes can be observed in the expansion of nodes 1 and 3 in Fig. 2, whereas the expansion of node 5 leads to four children.

This construction mechanism guarantees that the resulting tree is a decision tree for algebras, i.e., each single branching as well as the whole tree are covering (an algebra satisfies the properties associated with *at least* one child node/leaf node) and exclusive (an algebra satisfies the properties associated with *at most* one child node/leaf node). However, by this construction mechanism the algorithm can also introduce nodes for which no algebra of the considered size and axioms exists. Such nodes are called *dead-ends*. For instance, since there is no quasigroup of size 3 satisfying the properties P_1, $\neg P_2$, and $\neg P_3 \wedge \neg P_4$ node 9, in Fig. 2 is a dead-end node.

The tree in Fig. 2 is a classifying decision tree for quasigroups of size 3. That it is a decision tree, however, holds for arbitrary algebras, for instance, also for quasigroups of size 4, 5, 6, ... However, for other algebras the tree is not longer classifying, i.e., the leaf nodes do not necessarily specify isomorphy classes and dead-ends for other algebras.

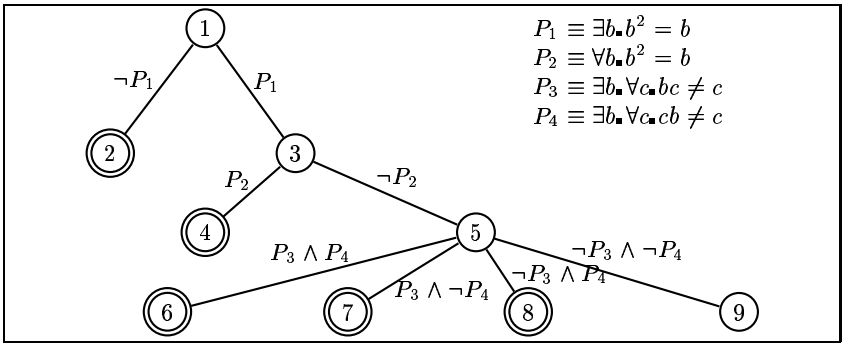


Fig. 2. Decision tree for the classification problem of order 3 quasigroups.

The classification trees produced by this algorithm are employed by the system discussed in this paper as follows. Let $\{P_1, \ldots, P_n\}$ be the properties associated with the leaf-nodes of a classifying decision tree and let $\mathcal{H}$ be a set of axioms, then our

system combines these to n specialisations of $\mathcal{H}$ by adding each property to the set of original axioms. These sets of axioms are then submitted to the model generators. For instance, consider the classifying decision tree for quasigroups of order 3 depicted in Fig. 2. The specialisation properties are the properties associated with the leaf nodes. We therefore get six different axiom sets as specialisations: $\mathcal{H} \cup \{\neg P_1\}$, $\mathcal{H} \cup \{P_1 \wedge P_2\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge P_3 \wedge P_4\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge P_3 \wedge \neg P_4\}$, $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge P_4\}$, and $\mathcal{H} \cup \{P_1 \wedge \neg P_2 \wedge \neg P_3 \wedge \neg P_4\}$.

Note that this heuristic can be approximated by the applicability approach described in section 4.2. If the system started off with a covering set of non-isomorphic algebras for each size, and if the threshold was set so that properties which applied to only a single algebra were chosen, then each property must be a classifying property (although, as Spass is not employed, this is not proved). However, in practice, there is no guarantee that the model generator has produced exemplars of each isomorphism class, and when the number of initial algebras becomes more than around 10 or so, HR usually fails to produce classifying properties for all isomorphism classes. In addition to potentially placing the sub-problems closer to the constrainedness knife-edge, another potential advantage of the classification tree approach is that, since the specialisation properties result from a decision tree, they divide the search space of possible algebras disjointly and hence there is little repetition of work. Also, it is more likely that the specialisation properties split the search space more evenly. Moreover, the properties cover the search space, so no solutions are lost.

5 Experiments and Results

Our aim here is to compare and contrast the three specialisation methods described in the previous section. In particular, we want to determine whether any gains in efficiency can be made when applying model generators FINDER, SEM and MACE to the quasigroup existence problems introduced in section 2.1. We have used each specialisation technique for all of the QG problems, but, due to space considerations, we present here only the results for QG1, QG4 and QG5 (for more detailed results please see ftp://ftp.cs.bham.ac.uk/pub/authors/V.Sorge/quasigroups/results.dvi). We compared the seven following approaches to solving the model generation problems:

- [Orig] This used a single processor with the original (non-specialised) axiom set. This is essentially a benchmark our parallel approach with respect to the performance of the original systems.
- [Inst1 + Inst2] refers to the two methods described in section 4.1. For Inst1, we fixed a single pair of elements $(0, 0)$ and pre-instantiated all possible results. For a problem involving quasigroups of size n the heuristic resulted in n specialised axiom sets. For Inst2, we additionally fixed a second pair $(0, 1)$, which resulted in $n \cdot (n - 1)$ specialisations.
- [Appl] refers to the method given in section 4.2. The 19 concepts of applicability 0.5 or higher resulted in 20 specialised axiom sets.
- [Class3, Class4, Class5] employ results of the classification algorithm described in section 4.3. In detail, Class3 uses the classifying properties from the decision tree for

System	Problem	Orig	Inst1	Inst2	Appl	Class3	Class4	Class5
FINDER	QG5.6*	0.04	0.03	0.08	0.05	0.05	0.06	0.15
	QG5.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	QG5.8	27.06	0.04	0.01	27.27	0.01	0.02	0.02
	QG5.9	0.04	0.02	0.02	0.01	0.04	0.04	0.01
	QG5.10*	3569.51	35300.13	7905.71	t/o	3570.17	t/o	t/o
	QG5.11	t/o	62.36	9.01	t/o	21.84	16.12	15.25
	QG5.12	316.42	317.22	57.26	332.12	321.72	3.57	494.4
	QG5.13	11342.30	12881.42	1682.91	8217.31	4014.80	3533.37	118.74
SEM	QG5.6*	0.00	0.00	0.01	0.01	0.00	13.71	142.55
	QG5.7	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	QG5.8	0.06	0.11	0.01	0.02	0.00	0.00	0.00
	QG5.9	0.00	0.00	0.00	0.00	0.02	0.01	0.00
	QG5.10*	1172.52	t/o	42749.04	1927.82	15193.60	t/o	t/o
	QG5.11	t/o	t/o	19.19	2518.37	38.69	40.28	170.89
	QG5.12	t/o	84.72	t/o	t/o	3.94	0.01	0.10
	QG5.13	t/o	t/o	t/o	t/o	t/o	t/o	89.83
MACE	QG5.6*	0.03	0.05	0.12	0.06	0.01	8.38	5.31
	QG5.7	0.57	0.58	0.60	0.01	0.01	0.01	0.01
	QG5.8	46.80	47.40	48.75	0.58	0.57	0.58	0.46
	QG5.9	48.52	48.63	47.95	0.00	0.00	0.01	0.01
	QG5.10*	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	QG5.11	t/o	t/o	t/o	t/o	t/o	t/o	t/o
	QG5.12	t/o	t/o	t/o	t/o	t/o	105.04	t/o
	QG5.13	t/o	t/o	t/o	t/o	t/o	t/o	t/o
FINDER	QG4.6*	1.58	0.33	0.04	1.58	1.22	1.20	2.07
	QG4.7*	84.75	38.77	8.81	98.26	89.87	90.59	123.47
	QG4.8	0.05	0.02	0.03	0.05	0.06	0.06	0.61
	QG4.9	0.38	0.14	0.01	0.31	0.37	0.36	0.66
SEM	QG4.6*	0.00	0.01	0.02	0.12	0.01	45.29	29.00
	QG4.7*	0.06	0.63	0.97	0.21	0.16	885.64	488.59
	QG4.8	0.09	0.02	0.00	0.01	0.01	0.01	2.26
	QG4.9	2.51	0.22	0.01	0.03	0.01	0.01	0.05
MACE	QG4.6*	0.07	0.07	0.13	0.10	0.02	37.82	54.66
	QG4.7*	2.64	2.87	5.68	4.22	0.31	76.78	4123.43
	QG4.8	61.49	61.45	60.86	0.88	0.87	0.92	2.27
	QG4.9	27023.21	26798.08	27040.67	2343.61	329.87	339.73	134.94
FINDER	QG1.6*	1660.59	782.67	117.88	1885.42	4875.55	5139.38	t/o
	QG1.7	0.34	0.26	0.24	0.05	0.09	0.01	0.02
	QG1.8	12776.18	100.18	4778.38	2.50	8704.15	4.54	74.13
	QG1.9	6438.08	74275.36	144.66	8929.90	6518.62	7.82	82.71
SEM	QG1.6*	0.69	6.58	1.36	2.16	1.62	6313.57	t/o
	QG1.7	0.29	0.01	0.00	0.01	0.01	0.00	0.00
	QG1.8	1566.77	0.34	0.37	0.04	1.25	0.01	0.21
	QG1.9	0.00	0.00	0.00	0.00	12.49	0.00	0.00

Fig. 3. Experimental results for the QG problem domain. Times are given in seconds, t/o indicates the model generator exceeded the 1 day timeout. Asterisks indicate that a given problem does not have a solution.

quasigroups of order 3, which comprises 6 properties and therefore the same number of specialisations. Class 4 uses the 41 properties of quasigroups of order 4 produced for the decision tree. For quasigroups of order 5, the decision tree yields more than 1500 properties, which we considered an impractical size. We therefore used the data from an intermediate decision tree for order 5 quasigroups, which yielded 468 properties, i.e., the tree still had all the properties of a decision tree, but the leaf nodes were not necessarily isomorphism classes for quasigroups of size 5.

The results from running these methods for QG1, QG4 and QG5 quasigroups of various sizes are given in table 5. Each row contains the time (in seconds) needed to solve a problem. The first column indicates the model generator for which the results were obtained; the second column specifies the particular problem, where QG$m.n$ is the problem to show the existence of an order n quasigroup for which QGm holds. Asterisks indicate that a given problem does not have a solution. For example QG5.6* is the problem to find a QG5 quasigroup of order 6 and the asterisk indicates that there is no such quasigroup. T/o indicates the model generator exceeded a 1 day timeout. The times are recorded as the time when the first of the parallel model generation pro-

cesses succeeded in finding a model. In the cases where no model exists, all processes have to terminate in order to guarantee that the search was exhaustive. Thus the time corresponds to the run time of the last process that terminated.

The experiments where conducted on a 113 node 2GHz Pentium 4 cluster with 2GB of RAM each with regular Ethernet interconnect. As a grid engine, we employed GridEngine v5.3 running on a quadruple 2.4GHz Xeon with 4GB of memory. For some of the experiments – notably for Class5 and the Inst2 heuristic for quasigroups of order > 10 – the number of specialisations exceeded the capacity of the cluster. We therefore ran these in several blocks, and the results should be understood as a simulation of a cluster of the necessary size. The results for the problems for which no model exists (QG5.6, QG5.11, QG4.6, QG4.7, QG1.6) demonstrate that in this case our specification approach fails. This is not surprising, as to conclude that no model exists, the search must terminate on all specialised axiom sets. Thus, if only one of the added properties has a negative effect on the search space, it affects the overall result.

The situation is exactly the opposite if there exists a model. Then, one specialised sub-problem which provides a model is sufficient to solve the overall problem. The results of our experiments demonstrate that employing the instantiation heuristic already gives a speed up in the cases where a model exists. Employing the classifiers can often further increase the speed up. In particular, for the more complex problems, the speed up is sometimes not only significant but also the model generators can find solutions for problems on which they timeout without the specialisations.

The results for the problems for which a model exists do not point out a clear winner, i.e., no method consistently outperformed all the other methods. However, altogether for each problem for which a model exists and which is non-trivial (i.e., the original problem is solved in less than 1 second) at least one specialisation heuristic gives a considerable speed up.

Another pattern also appears to emerge: for the small sizes, there seems to be little point in using a specialisation method. For the intermediate sizes, simply distributing the process using the Inst2 method appears to be quicker in many instances. For the larger sizes, however, the classification methods appear to do better. Also apparent is that the classification methods usually outperform the applicability method. It is very interesting to compare the three decision trees methods for the generation of specification sub-problems. The Class4 decision tree outperforms the Class3 decision tree on almost all problems for which a model exists (and for the exceptions of this rule the Class3 tree performs only slightly better). However, for the Class5 tree we cannot observe a clear improvement as compared with the Class4 tree: except for QG5.13 it does not perform significantly better but on the other problems it performs worse. This may indicate that our approach is limited: splitting the problem into more sub-problems will not necessarily result in faster solutions. We intend to evaluate this hypothesis with more experimentation.

6 Related Work

The study of quasigroups, and more generally, automated discovery in finite algebras, stimulated much research in Automated Reasoning. For instance, [7, 14, 18, 19] report

on the use of model generation techniques and propositional provers to tackle quasigroup existence problems. In particular, open problems in quasigroups became a challenge for friendly competition. J. Zhang was the first to use a general reasoning program to solve an open quasigroup existence problem. Later, Fujita, Slaney, Stickel, McCune, and H. Zhang used their systems to solve several open cases and reported very competitive results. More recently, completing partial quasigroups has been proposed as a structured benchmark domain for the study of constraint satisfaction methods [11].

The difficulty with using automated reasoning techniques and constraint satisfaction techniques is that the underlying problems are intractable (undecidable, NP-complete). No matter how good the algorithms are, there is a constant need for more computing power. This motivated the distribution of problems and the application of concurrently running applications in the context of quasigroup existence problems. [19] presents Psato a parallel prover for propositional satisfiability for networks of workstations. A key property of the approach is that the concurrent processes explore disjoint portions of the search space such that parallelism is employed without introducing redundant search. [11] and [10] report on the possible concurrent application of randomised algorithms, which are among the best current algorithms for solving computationally hard problems. The time required by these algorithms to solve a particular problem instance can vary drastically (indeed the algorithms studied in [11, 10] exhibit so-called heavy-tailed runtime distributions). Hence, running several instances of a randomised algorithm can boost the performance.

In contrast to Psato, the randomised algorithms all search the same search space but because of the inherent randomisation they traverse it differently. Our approach is similar to the Psato approach with respect to the disjoint partition of the algebra space that results from the usage of decision trees. However, since the generation of classifying decision trees is not deterministic (see [4] for details) we could also randomly create different decision trees to use in our approach and employ concurrency on these decision trees, which would then be similar to different runs of randomised algorithms.

Automatically re-formulating problem statements in order for solvers to more efficiently find solutions has been addressed in many areas of Artificial Intelligence, most notably constraint solving. In particular, the process of deriving additional information from the problem statement so that *implied constraints* can be added without loss of generality has been researched recently [6]. Also, in [5], Colton and Miguel use the HR program to generate additional constraints for quasigroup existence problems, and this approach provided up to 10 times speed up for certain problems. In contrast to the work described here, however, this approach was semi-automated, ad-hoc and concentrated more on using implied constraints, rather than the case splits we use here. It also focused on improving a single constraint solver, rather than distributing the workload over a grid, as discussed here.

7 Conclusions and Further Work

We have presented a novel approach to distributing a model generation process for solving algebraic existence problems. Based on a general framework, we investigated several methods to split a model generation problem into multiple more specific sub-

problems, which can be processed in parallel. The specialisations range from simple instantiation to more intelligent choosing of specialisation properties. Our experiments demonstrate that, by using a grid-based concurrent distribution, our approach provides greater efficiency over a single-process approach. Moreover, in many cases, the more elaborate specification approaches outperformed the simple specification approaches. We plan to perform more experiments in order to determine more precisely when a particular approach will outperform another.

We have used this approach for the QG5 problem of size 18, which is still an open problem. Unfortunately, none of the specialised sub-problems has returned a solution, which gives some evidence towards the hypothesis that no solution exists. We have also looked for cases where a search using specialising property P and a search using specialising property $\neg P$ have both ended unsuccessfully, thus proving that there are no solutions, but none have been forthcoming. One avenue for further work may be to generalise this approach: suppose that the search for sub-problems with specialisations $P_1, \ldots, P_n$ have all finished without solutions. Then, if a solution does exist, it must have the property $\neg P1 \wedge \ldots \wedge \neg Pn$. Hence, we could take successively larger subsets of $\{\neg P1, \ldots, \neg P_n\}$ and attempt to prove with an automated theorem prover that all the properties in the subset holding at once is inconsistent with the axioms, hence showing that no solution can exist.

Our experiments with the three specialisation methods described in the paper did not provide us with a clear winner, i.e., a method that outperforms all other methods on all problems. Furthermore, without additional experimentation, it is difficult to draw generalisations about how best to specialise model generation problems into sub-problems. However, the specialisation methods showed a significant speed up over a single-process approach, and, in particular, for larger sizes, the classification methods appear to be better than the other specialisation methods. It is also difficult to choose a clear winner for the the three different classification trees approaches. In the worst case, a specialisation method turned out to take even longer than the original problem. These observations raise the question whether it is possible to predict the performance of a particular method on a particular problem. If this is possible, then a suitable method can be chosen. Our current experiments do not provide sufficient data to give an account on this question. However, we hope that further experiments will provide enough data that we can try to figure out (learn) problem and specification parameters for an automated selection of the specification method.

Although we experimented here only with quasigroups, our approach is general enough for model generation of any finite algebra with particular properties. Moreover, the idea to specialise problems goes beyond algebraic domains and beyond the use of model generators. That is, we can envisage our approach being used for general problems in model generation and constraint solving. While our main motivation remains adding to the mathematical literature on finite algebras, we hope to demonstrate that the notion of specialising problems into sub-problems, then distributing these sub-problems on a Grid architecture has much potential for Artificial Intelligence problem solving.

References

1. S. Colton. *Automated Theory Formation in Pure Mathematics*. Springer Verlag, 2002.

2. S Colton, A Bundy, and T Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.
3. S. Colton, A. Bundy, and T. Walsh. Automatic identification of mathematical concepts. In *Proc. of the 17th International Conference on Machine Learning (ICML2000)*, pages 183–190. Morgan Kaufmann, USA, 2001.
4. S. Colton, A. Meier, V. Sorge, and R. McCasland. Automatic generation of classification theorems for finite algebras. In D. Basin and M. Rusinowitch, editors, *Proc. of the International Joint Conference on Automated Reasoning (IJCAR–2004)*, LNAI. Springer Verlag, 2004. Forthcoming.
5. S. Colton and I. Miguel. Constraint generation via automated theory formation. In *Proc. of CP-01*, 2001.
6. A.M. Frisch, I. Miguel, and T. Walsh. Extensions to proof planning for generating implied constraints. In *Proc. of Calculemus-01*, pages 130–141, 2001.
7. M. Fujita, J. Slaney, and F. Bennett. Automatic Generation of Some Results in Finite Algebra. In R. Bajcsy, editor, *Proc. of the 13th International Joint Conference on Artificial Intelligence (ICJAI)*, pages 52–57. Morgan Kaufmann, USA, 1993.
8. Gap. *GAP Reference Manual*. The GAP Group, School of Mathematical and Computational Sciences, University of St. Andrews, 2000.
9. P Gent, E MacIntyre, P Prosser, and T Walsh. The constrainedness of search. In *Proceedings of AAAI-96*. American Association for Artificial Intelligence, 1996.
10. Carla P. Gomes and Bart Selman. Algorithm portfolios. *Journal of Artificial Intelligence*, 126:43–62, 2001.
11. Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.
12. S.R. Kumar, A. Russel, and R. Sundaram. Approximating latin square extensions. *Algorithmica*, 24:128–138, 1999.
13. C. Laywine and G. Mullen, editors. *Discrete Mathematics using Latin Squares*. Wiley, 1998.
14. W.W. McCune. A davis-putnam program and its application to finite first-order model search: quasigroup existence problems. Technical report, Argonne National Laboratory, Division of MSC, 1994.
15. W.W. McCune. *Mace4 Reference Manual and Guide*. Argonne National Laboratory, Division of MSC, 2003. ANL/MCS-TM-264.
16. G L Miller. On the $n^{log_2 n}$ isomorphism technique. Technical Report TR17, The University of Rochester, 1976.
17. J. Slaney. *FINDER, Notes and Guide*. Center for Information Science Research, Australian National University, 1995.
18. J. Slaney, M. Fujita, and M. Stickel. Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. *Computers and Mathematics with Applications*, 29:115–132, 1995.
19. H. Zhang, M. Bonacina, and H. Hsiang. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems. *Journal of Symbolic Computations*, 21:543–560, 1996.
20. H Zhang and J Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symposium*, 1994.
21. J. Zhang and H. Zhang. *SEM User's Guide*. Department of Computer Science, University of Iowa, 2001.

Approaches to Case-Base Similarity for Retrieval

Savvas J. Nikolaidis[1]

[1]Department of Informatics, Aristotle University of Thessaloniki
54124 Thessaloniki, Greece.
e-mail: snikol@csd.auth.gr

Abstract: Similarity measurement, the most important factor to the performance of a Case-based reasoning system, has been generally considered in a rather restricted view. Most case-based reasoners tend to look at similarity as the measurement of distances of objects. However, this restricted aspect constrains the scope of Case-based reasoners to areas where the cases can be described as measurable vectors. Alternative approaches to Case-based similarity for retrieval can extend the scope of Case-based reasoners to fields where it was not possible with the classical approach. In this paper we study some alternative approaches and conduct experiments to study under which circumstances these alternative methods have performance advantages against the classic approach.

Keywords: Case-Based Reasoning, Case Retrieval, Similarity Measurement.

1 Introduction

Case-based reasoning is in essence analogical reasoning where the system tries to predict the outcome of a given situation according to the outcome of the most similar case, or cases, from a knowledge base of solved cases. Case Based Reasoning is a technique used in situations where we want to reduce the burden of knowledge acquisition, avoid repeating mistakes made in the past, work in domains where a well understood model doesn't exist, learn from past experiences, reason with incomplete or imprecise data, provide means of explanation and reflect human reasoning [14]. There are four key issues in the case-based reasoning process: (a) identifying key features, (b) retrieving similar cases in the case base, (c) measuring case similarity to select the best match, and (d) modifying the existing solution to fit the new problem. The most important part of a case-based reasoning system is the retrieval stage, where the system must find, in a sometimes-huge case base, the best matching case or cases from which to produce the prediction for the outcome of a given situation. The case based reasoning system must be able to identify the suitable cases for retrieval. The efficiency of this stage is a critical factor for the overall system performance. If the system is not able to properly identify a suitable case this case may not be retrieved, although it might be useful. Improving retrieval is an open problem in case-based reasoning research and case-based reasoning system development [11]. Case adapta-

tion may also be needed. Case attributes can be qualitative, quantitative, descriptive or other.
The retrieval stage in Case-Based Reasoning systems requires the use of some kind of similarity measurement for the best case to match. Search for similarity, is a problem which occurs in diverse applications, such as stock market prediction [17], [20], plagiarism detection [19], forest fire prediction [18], and protein and DNA sequencing [16]. A number of similarity measuring techniques have been used in different systems. The selection of the similarity measurement is very important, because, if the one selected is not the appropriate, the system will produce erroneous results. The selection depends on being able to identify relevant attributes and make use of them. There is no similarity measurement that can fit in all situations. The Question of defining similarity is one of the most subtle and critical issues raised by case-based reasoning [12]. Very serious consideration must be given to the nature of the data, which dictate the selection of the suitable similarity measurement.

2 New Approaches for Case Identification and Retrieval

In this paper we present different approaches for representing similarity and identifying cases for the retrieval stage in case based reasoning systems. First we study the Euclidean distance. Euclidean distance is a similarity measure frequently used in the literature. The Euclidean distance is employed in the Nearest Neighbor algorithm and the k- Nearest Neighbor algorithm. With the NN algorithm we try to find from our Knowledge Base the closest match for a given situation and with the k-NN algorithm we are searching for the k closest matches. Then from the results we derive our evaluation. Two different measures of similarity, namely the unweighted Euclidean distance and the weighted Euclidean distance are in use. Each one of them is detailed below. The Euclidean distance can only be used with quantitative attributes. All formulas presented assume that x and y represent size attributes. If our attributes are of a different kind we have to devise a mapping to the set of real numbers so that we can use this method. The number of attributes employed will determine the number of dimensions used. It is common in CBR for the attributes, i.e., features vectors to be weighted to reflect the relative importance of each feature.
Depending upon the problem we are called to confront and the nature of the data involved on top of the classic distance-based approach we have a useful array of alternative techniques that can be used. In the case of textual similarity, as for example when we are trying to compare documents or web pages the “edit distance” is an interesting approach. This similarity function is discussed in detail in [15]. Edit Distance *ed(s1, s2)* between two strings s1 and s2 is the minimum number of character edit operations (delete, insert, and substitute) required to transform s1 into s2, normalized by the maximum of the lengths of s1 and s2.
Cases that are not simple in form can be described as graphs. Pin graphs are finite collections of particular graphs which have certain distinguished properties. There are special nodes which may be replaced by complete individual graphs. For more details the reader is referred to [1] and [9]. Pin graphs can be used for case representation. These structures can naturally describe certain technical objects and there are algo-

rithms known to be more efficient on hierarchically structured graphs than on flat ones. In these structures structural similarity is employed as the similarity function.
A much promising alternative approach to similarity measurement is the incorporation of fuzzy logic to case based reasoning systems. Fuzzy logic methods and techniques have been used in CBR since the early nineties [3], [4], [6] and they have been extensively used during more resent years. [5], [8], [10]. Main et al [13] explain how fuzzy logic applies to CBR. A case-based reasoner has to find a case or a set of cases similar to the target problem. Most CBR systems are based on similarity relations between the target and the cases. These relations are vague by nature. Fuzzy logic deals with vague relationships. We also combine fuzzy logic with CBR because fuzzy logic is helpful for acquiring knowledge and it provides methods for applying knowledge to real-world data. Fuzzy logic simplifies elicitation of knowledge from domain experts, such as knowledge of how similarity between two cases depends on the difference between their individual, collective, and temporal attributes. Fuzzy logic emulates human reasoning about similarity of real-world cases, which are fuzzy, that is, continuous and not discrete. [8]. A fuzzy approach is proposed and implemented below.
More complex applications may require the combination of different concepts and techniques. Formal concepts as well as sophisticated heuristics may be employed to deal with more complicated situations.
In the CBR cycle, during the analysis of the similarity among cases, the Initially Match Process during the Retrieve Step [7], fuzzy classification methods can be used in order to improve the performance and the efficiency of the CBR system. The cases can be “preprocessed” and assigned to one or more of a number of fuzzy categories. The retrieval of previously solved problems similar to the current one is a two-step process. The initial matching process can be described as a procedure to isolate interesting, or as we say, potentially applicable cases. For example we could have four preprocessed categories of a person’s age: “child”, “young”, “middle aged” and “old” and search only the appropriate one(s) when we encounter a new case. During this step the system reduces the set of cases to be compared to the current case in the second step of the similarity computation. Using the Similarity Function in the second stage we evaluate selected cases to determine the definitely applicable ones. The algorithm can be described:

1. Locate and retrieve potentially applicable cases from the case-base.
2. Evaluate selected cases to determine the applicable ones.
3. Transfer knowledge from the old case(s) to the current one.

The fuzzification process is a feature selection algorithm and creates fuzzy categories that are used for indexing. The fuzzyfier dictates the way an attribute is turned into a classifiable one in a fuzzy set. At the end of the process the case base has been converted into an indexed case base according to the fuzzy categorization.
The fuzzification process works through the case base according the following steps:

1. Eliminate one attribute and test (try to find irrelevant attributes).

2. Eliminate all but one attributes and test (find the relative significance of different attributes).
3. Assign weights to the attributes.
4. Identify significant or irrelevant ranges in attribute values. Enhance the former and discard the later.
5. Normalize to the fuzzy set belonging range [0, 1].
6. Repeat the above steps until there is no significant result improvement (the performance “tops up”).

The fuzzification process is shown in *Figure 1*.

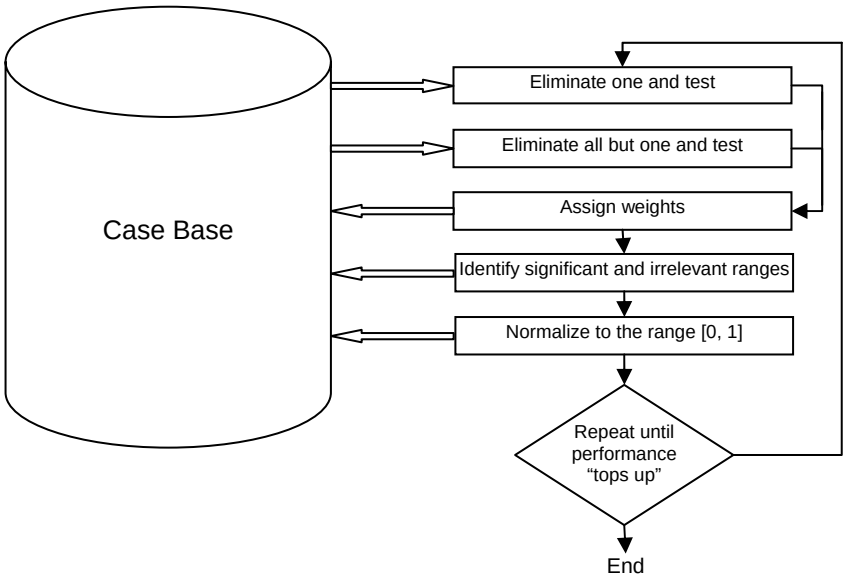


Fig. 1. *The fuzzification algorithm*

For similarity function we use the similarity function used in case cased reasoning systems based on the
By using the above procedure we preserve a lot of the information that would be lost if we were using crisp sets, useless information that could produce erroneous predictions is discarded and finally our prediction results become more easily justifiable.

3 Experiments

We have executed a series of experiments. Initially we used two unrelated sets of data describing the real estate market situation. The common aim of all the experiments is to estimate the real value of the property from the data in each data set.

Table 1 Mean values of correct and wrong evaluations of the value of real estate property using randomly selected 50 member testing sets, according to different methods for the first data set consisting mainly of nominal attributes.

	"Hits"	"Misses"	Percentage
Unweighted Euclidean	14,3	35,7	28,6%
Weighted Euclidean	27,6	22,4	55,2%
k-NN	32,3	17,7	64,6%
F-CIR	37,2	12,8	74,4%

The first data set consists mainly of quantitative data. Attributes in these set include: property surface area in m^3, the flour on which the property is located, distance from the city center and from major highways in km, age of the building etc. The data has been tested with the unweighted Euclidean distance method, the weighted Euclidean distance method *(NN)* and *k-NN* with different values of *k*, and our Fuzzy method. From the dataset of more than 500 cases 50 were randomly selected for evaluation and the rest for training. The experiment was repeated 400 times for statistical purposes. The results are shown in *Table 1*.
All Algorithms except the unweighted Euclidean Nearest Neighbor performed well. The slightly better performance of F-CIR compared with the weighted Euclidean is derived from the identification of significant and irrelevant attribute ranges during the fuzzification process.

Table 2 Mean values of correct and wrong evaluations of the value of real estate property using randomly selected 50 member testing sets, according to different methods for the second data set consisting mainly of nominal attributes.

	"Hits"	"Misses"	Percentage
Unweighted Eucledean	13,5	36,5	27%
Weighted Eucledean	16,8	33,2	33,6%
k-NN	18,3	31,7	36,6%
F-CIR	33,1	16,9	66,2%

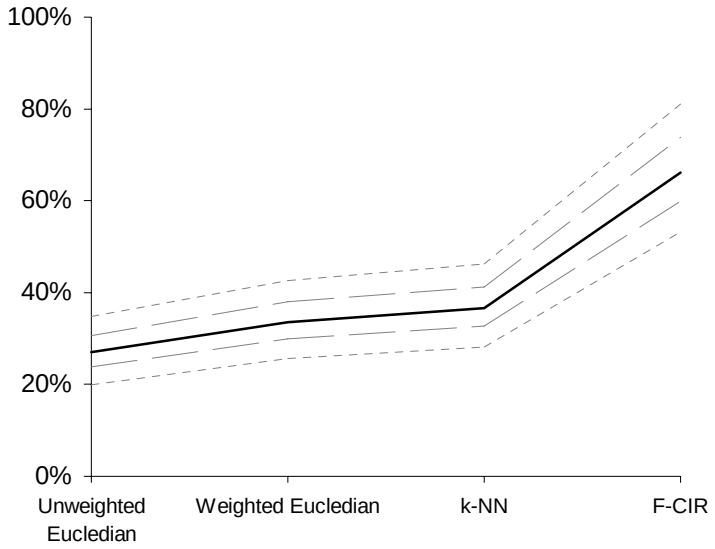


Fig. 2 *Different methods performance comparison for the evaluation of the value of real estate property using data describing the real estate market situation which consists mainly of nominal attributes.*

The data set used in the second set of experiments, consists mainly of nominal data. Attributes in these set include: city area where the property is located (city center, east side, name of suburb), description of the apartment (studio, 2-bedroom etc), accessibility to the employment centers, traffic conditions in the area, apartment orientation, level of pollution in the area etc. The experiments were conducted with the same philosophy as the ones for the first dataset (400 repetitions). The results are shown in *Table 2* and *Figure 2.* F-CIR worked satisfactory.

4 Conclusions

We have examined and tested different approaches to the similarity measuring problem for the case-based reasoning systems retrieval stage. We have seen that the classic distance based approach may work satisfactory in a number of situations but it can be inadequate to describe efficiently the similarity between objects in a case base. Under these circumstances the use of a non-standard similarity function is required. Depending on the nature of the problem and the data involved alternatives include a number of methods. The selection of the appropriate one is critical for the performance of the system. As the problems faced by case based reasoning systems get increasingly complex the need for alternative approaches, or even combinations of them, is certain to increase. The fuzzy case based reasoning system we presented here was tested and it performed satisfactory.

References

1. Helmut Alt and Johannes Blomer. Resemblance and symmetries of geometric patterns. In Burkhard Monien and Thomas Ottmann, editors, Data Structures and Effcient Algorithms, volume 594 of Lecture Notes in Computer Science, pages 1-24. Springer-Verlag 1992.
2. Bandini S., and Manzoni, 2001, Proceedings of the 2001 ACM symposium on Applied computing, pp.462-466
3. Bento, C. and Costa E., 1993: A similarity metric for retrieval of cases imperfectly described and explained, in: Proceedings First European Workshop on Case-Based Reasoning, Richter, Wess, Althoff and Mauer (eds.), Vol. 1, 1993, pp 8-13.
4. Bonissone, P.P., and Ayub, S., 1992: Similarity measures for case based reasoning systems, in Proceedings of the IPMU-92 Conference, 1992, pp. 483-487.
5. S. Chaudhuri, K. Ganjam, V. Ganti, R, Motwani, 2003: Robust and Efficient Fuzzy Match for Online Data Cleaning, Procceedings of SIGMOD 2003.
6. Dubois and Prade, 1992: Gradual inference rules in approximate reasoning, Information Sciences, 61, 103-122
7. Kolodner. Case-Based Reasoning. Morgan Kaufmann, San Mateo (CA), 1993.
8. Hansen, B. K. (2000) Weather prediction using similarity between temporal cases and fuzzy sets, Master of Computer Science thesis, Dalhousie University – Daltech.
9. Franz Hofting, Thomas Lengauer, and Egon Wanke. Processing of hierarchally defined graphs and graph families. In Burkhard Monien and Thomas Ottmann, editors, Data Structures and Efficient Algorithms, volume 594 of Lecture Notes in Computer Science, pages 44-69. Springer-Verlag, 1992.
10. Jeng, B. C., and Liang, T.-P. (1995) Fuzzy indexing and retrieval in case-based systems, Expert Systems With Applications, Vol. 8., No. 1, 1995. Elsevier Science Ltd., 135–142.
11. Leake, D. B. (1996) CBR in context. The present and future; in Leake, D. B. (editor) (1996) Case-Based Reasoning: Experiences, Lessons & Future Directions, American Association for Artificial Intelligence, Menlo Park California, USA, 3–30.
12. Luger, G. F., and Stubblefield, W. A. (1998) Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Addison Wesley Longman, Reading, Massachusetts, USA, pg. 238.
13. Main. J., Dillon, T. S., and Khosla. R. (1996) Use of fuzzy feature vectors and neural vectors for case retrieval in case based systems, NAFIPS 1996 Biennial Conference of the North American Fuzzy Information Processing Society, IEEE, New York, NY, 438–443.
14. Main, J.; Dillon, T. S.; and Shiu, S. C. K. 2000. A tutorial on case-based reasoning; in Pal, S. K.; Dillon, T. S.; and Yeung, D. S. eds. 2000. Soft Computing in Case Based Reasoning. London, UK, Springer.
15. G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing methods for approximate string matching. IEEE Data Engineering Bulletin, 24(4):19--27, 2001.

16.Pearson, W. R., and Lipman, D. J. (1988) Improved tools for biological sequence comparison, Proceedings of the National Academy of Sciences, Vol. 85, 2444–2448, April, 1988, Biochemistry.
17.Rafiei, D. (1999) Fourier-Transform Based Techniques in Efficient Retrieval of Similar Time Sequences, Ph.D. thesis, Department of Computer Science, University of Toronto, Ontario, Canada.
18.Rougegrez, S. (1993) Similarity evaluation between observed behaviours and the prediction of processes. In Wess, S., Althoff, K. D. and Richter, M. (eds.), Topics in Case-Based Reasoning, Proceedings First European Workshop on Case-Based Reasoning, 1993, Springer-Verlag, Berlin, 155–166.
19.Shivakumar, N., and Garcia-Molina, H. (1995) SCAM: A copy detection mechanism for digital documents, Digital Libraries ‘95, The Second Annual Conference on the Theory and Practice of Digital Libraries, 155–163.
20.Xia, B. B. (1997) Similarity Search in Time Series Data Sets, Master of Science thesis, Department of Computer Science, Simon Fraser University, BC, Canada.