Project Planning Under Temporal Uncertainty

Susanne Biundo Roland Holzer
Bernd Schattenberg
Department of Artificial Intelligence, University of Ulm, 89069 Ulm, Germany
phone: +49 731 50 24121 fax: +49 731 50 24199
mail: firstName.lastName@informatik.uni-ulm.de

Abstract. This paper presents an approach towards probabilistic planning with continuous time. It adopts stochastic concepts for continuous probabilities and integrates them into an HTN-based planning framework. Based on uncertain time durations associated with primitive tasks the time consumption probabilities of non-linear plans can be accumulated and thus an overall probability for a successful execution of complex plans can be computed. Furthermore, heuristics for the decomposition of abstract tasks can be derived that guide the search towards plans with a minimized average value/variance of their overall time consumption. An example from software project planning is used to demonstrate our approach.

1 Introduction and Motivation

The process of software development has to deal with a variety of unpredictable events, which make it particularly hard to calculate costs, to provide adequate buffer times, and finally to meet project deadlines. Keeping to the project's time schedule is obviously critical for every software company, and consequently there is a growing market for supporting tools helping to keep such projects manageable. This work is motivated by the observation that present-day tools mostly offer limited management support by basically highlighting already off-schedule project threads. AI planning techniques offer a notably more intelligent support in the project definition phase, especially if they are able to deal with uncertain information about actual implementation times and the like.

In this paper, we introduce an approach to handle uncertain time consumption of actions in planning. We adopt the concepts for continuous probabilities and their computations from stochastics and integrate them into a standard HTN-based planning framework. The resulting probabilistic planning approach allows for an adequate representation of (a continuous model of) uncertain time consumption. The duration probabilities of single actions and action sequences can be efficiently computed during planning. This enables the construction of plans that are guaranteed to meet certain probability thresholds w.r.t. given time limits. Furthermore, the approach can be generalized to handle parallel threads of execution and to accumulate alternative task decompositions. Not only does this enable a qualified decision if various alternative solutions are at hand, it even suggests a useful pruning of the search space. Furthermore, we show how heuristics for HTN-planning can be generated that lead to the synthesis of plans with a minimized average duration and/or variance of time consumption.

The rest of the paper is organized as follows. Section 2 introduces the foundations of HTN planning and the basic concepts of our probabilistic approach. In Section 3 the techniques are generalized to partially ordered plans, while Section 4 shows how the uncertain duration of primitive operations can be propagated along the task hierarchy into abstract tasks. An example taken from software project planning illustrates our approach in Section 5 and its implementation is sketched in Section 6. We conclude with a review of related work and some final remarks.

2 Basic Definitions

Our HTN-based planning formalism relies on the usual STRIPS representations of states and operators for the primitive action level. A *state* is a finite set of ground atoms. An *operator* instance $o = (\text{prec}(o), \text{add}(o), \text{del}(o))$ consists of three such sets: the *preconditions* and the *positive* and *negative effects*, respectively. It appears to be a ground instance of a respective operator schema. Such an operator instance o is *applicable* in a state s iff $\text{prec}(o) \subseteq s$. The result of applying operator $o = (\text{prec}(o), \text{add}(o), \text{del}(o))$ in state s is a state $\text{result}(s, o) = (s \cup \text{add}(o)) \setminus \text{del}(o)$. Operators are also called *primitive tasks*.

A *plan* $p = \langle o_0 \dots o_n \rangle$ is a sequence of operator instances such that for every state s_i, in which o_i is applicable, we have that o_{i+1} is applicable in $\text{result}(s_i, o_i)$, where $s_{i+1} = \text{result}(s_i, o_i)$ for $0 \leq i \leq n$. A plan p is then *applicable* in s_0, and the resulting state $\text{result}(\text{result}(\dots\text{result}(s_0, o_0), o_1) \dots, o_n)$ of p is denoted by $\text{result}(s_0, p)$.

Abstract actions are represented by *complex tasks* t. For each complex task t, there exists at least one *method* $m = (t, d)$, relating t and a *task network* d which implements t. Task networks are structures $d = (T, \prec, V)$, where T is the set of complex and/or primitive tasks into which t is decomposed, $\prec$ is a partial order on T, and V is a set of codesignation and non-codesignation constraints on the variables occuring in $T \cup \{t\}$. Plans are generated by *expanding* abstract tasks, i.e. by subsequently applying methods in the following way: given a task network $(T, \prec, V)$, an abstract task $t \in T$, and a method $m = (t, (T_t, \prec_t, V_t))$. Let furthermore $\prec_\delta$ be the subset of $\prec$ in which t occurs. The network resulting from applying m is $d' = (T', \prec', V')$ with

$$\begin{aligned}
T' &= (T \setminus t) \cup T_t \\
\prec' &= (\prec \setminus \prec_\delta) \cup \prec_t \cup \{(t_a, t_t) \mid (t_a, t) \in \prec, t_t \in T_t\} \cup \{(t_t, t_a) \mid (t, t_a) \in \prec, t_t \in T_t\} \\
V' &= V \cup V_t
\end{aligned}$$

A *planning problem* is a quadruple $(d, Init, \mathcal{T}, \mathcal{M})$, where d is a task network, $Init$ a set of ground atoms, the *initial state*, $\mathcal{T}$ a set of complex and primitive task schemas, and $\mathcal{M}$ a set of methods for the complex tasks in $\mathcal{T}$. Given a task network $d' = (T', \prec', V')$ which results from d by subsequently expanding all abstract tasks in d. Then d' is a *solution* to such a planning problem iff T' only contains primitive tasks from $\mathcal{T}$ and all operator sequences which can be built from the ground instances represented by T' and V' and which are in consistency with $\prec'$ represent a plan p which is applicable in $Init$.

In order to represent uncertain durations of operators, we make use of continuous random variables which are used in stochastics to model continuous events. A *random variable* $\mathcal{X} : \Omega \to \mathbb{R}$ is a measurable function that maps the event space Ω onto the real numbers $\mathbb{R}$.

The *distribution* of $\mathcal{X}$ is described by a *probability density* $D : \mathrm{I\!R} \rightarrow \mathrm{I\!R}$, denoted by $\mathcal{X}_D$ or $\mathcal{X} \sim D$. The *mean value* $\mu = E(\mathcal{X}_D)$ of a random variable with density D is defined as $\int_{-\infty}^{+\infty} xD(x)dx$, and the *variance* $\mathrm{Var}(\mathcal{X}_D)$ of a random variable is given as $E((\mathcal{X}_D)^2) - E(\mathcal{X}_D)^2$, also denoted by σ^2.

For this presentation we focus on random variables with the normal-distribution density, denoted by $\mathcal{X}_{\mathcal{N}(\mu,\sigma^2)}$. We will see below, that this choice does not imply any loss of generality. The probability that the value of a such random variable is lower than a given value a is computed as follows:

$$Pr[\mathcal{X}_{\mathcal{N}(\mu,\sigma^2)} < a] := \int\limits_{-\infty}^{a} \mathcal{N}(\mu,\sigma^2)dx = \int\limits_{-\infty}^{a} \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{\sigma^2}} dx$$

Note, that there is no antiderivative for the normal distribution, so we have to use a standard approximation to compute the probabilities (the error introduced by this approximation is however less than 10^{-8}).

Our operator description is extended by an annotation $rv(o)$, which is the random variable that describes the uncertain duration of the operator o. The mean value of such a random variable represents the average amount of time consumed by the operator, while the variance describes the uncertainty of its duration. The mean value of the density of a random variable $rv(o)$ is negative or zero. A mean value of zero means that the operator takes no time for execution. Variance is zero for absolutely certain operator durations. In addition, the description of a planning problem is extended by a random variable $\mathcal{I}$, that represents a limit to the maximum duration of the plan. $\mathcal{I}$ has a positive mean value and may have a variance of zero. Furthermore, a threshold is added to the problem description. A plan that has been generated in the HTN fashion described above is now considered valid only if the probability of exceeding the given duration limit is less than the user-defined threshold.

3 Computing the Uncertain Duration of Plans

The random variable that describes the duration of a plan is obviously the sum of all the random variables of the single operators. The density of a sum of random variables is computed by the *convolution* of the single densities. Given two random variables $\mathcal{X}^1_{D_1}$ and $\mathcal{X}^2_{D_2}$ where D_1 and D_2 are the density functions, the density of their sum is defined by:

$$D_{\mathcal{X}^1_{D_1}+\mathcal{X}^2_{D_2}}(t) \;:=\; \int\limits_{-\infty}^{+\infty} D_1(\tau)D_2(t-\tau)d\tau \;=\; \int\limits_{-\infty}^{+\infty} D_1(t-\tau)D_2(\tau)d\tau$$

Computing the convolution of densities is analytically hard in the general case and for some cases even impossible (depending on the kind of the random variables densities). Another difficulty is that simulation techniques like Monte Carlo Simulation are not applicable in this case, because the simulation converges to the value of the real integral only for finite intervals. To compute the probability for a plan to exceed a given duration, we need to compute the integral value over intervals like $(-\infty, \ldots, 0]$. Stochastics literature shows, that when dealing with continuous random variables, most of the stochastic processes can be approximated using the normal-distribution. This property can be utilized effectively because the distribution of the sum of two normal distributed random variables is also normal-distributed. Furthermore the convolution of normal-distributed random variables can be computed quite efficiently. However, there exists no formula for the antiderivative of the normal-density, but at least there exists a good approximation for it.

Given n normal-distributed random variables $\mathcal{X}^i$, the density of the sum of these variables can be computed by:

$$\mathcal{X}^i \sim \mathcal{N}(\mu_i, \sigma_i^2) : \sum_i \mathcal{X}^i \sim \mathcal{N}\left(\sum_i \mu_i, \sum_i \sigma_i^2\right)$$

The overall duration of a plan $p =< o_1, \ldots, o_n >$ is represented by a new single random variable $\mathcal{Y}$. Given operator durations $rv(o_i) = \mathcal{X}_{\mathcal{N}(\mu_i, \sigma_i^2)}$, the density of $\mathcal{Y}$ is calculated by convoluting the densities $\mathcal{N}(\mu_i, \sigma_i^2)$ of all operators. Using the formula above, we get the following density for $\mathcal{Y}$:

$$\mathcal{N}\left(\sum_{i=0}^{n} \mu_i, \sum_{i=0}^{n} \sigma_i^2\right)$$

In order to compute the density of the duration of a partially ordered plan or a task network, it is necessary to find a sequence of operators or tasks that leads from a “start” task to an “end” task and has a maximum duration. We call such a sequence the *critical path*. Calculating random variables $\mathcal{Z}^i$ for every path p_i in a partially ordered task network or plan can be done efficiently. The complex part is to figure out which path is the critical one, as there exists no comparison-metric on random variables. The critical path must be defined over the probability for each sequence to over-run the given maximum duration $\mathcal{I}$.

Our approach to solve this problem extends previous work on resource consumption for linear plans [1]: We extract every possible path p_i in the partially ordered task network and compute the random variable $\mathcal{Z}^i$ for the total duration of the respective sequence. The density of $\mathcal{Z}^i$ is calculated analogous to the duration of a totally ordered plan. After that, we add the initial time-bound $\mathcal{I}$ (i.e. the maximum duration allowed for the plan) to each $\mathcal{Z}^i$. The critical path can now be determined: it is the path p_i with maximum probability $Pr[\mathcal{I} + \mathcal{Z}^i \leq 0]$.

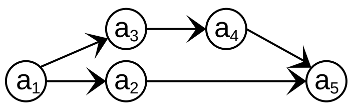

Figure 1: A partially ordered plan.

A short example illustrates the computation of the critical path. Given the operators $a_1, \ldots, a_5$ with $rv(a_3) = \mathcal{X}_{\mathcal{N}(-1,0.8)}$, $rv(a_4) = \mathcal{X}_{\mathcal{N}(-1.5,0.8)}$, and $rv(a_5) = \mathcal{X}_{\mathcal{N}(-2,0)}$ ordered like shown in Figure 1. The duration random variable $\mathcal{Y}^1$ for the first path a_1, a_3, a_4, a_5 has the density $\mathcal{N}(-8.5, 2.1)$ while that of $\mathcal{Y}^2$ for the second path a_1, a_2, a_5 has the density $\mathcal{N}(-9, 1.5)$. Let $\mathcal{I}_{\mathcal{N}(12,0)}$ be the random variable that describes the allowed maximum duration of the plan. We compute the probabilities for $\mathcal{Z}^1$ and $\mathcal{Z}^2$ by:

$$\begin{aligned} Pr[\mathcal{Z}^1_{\mathcal{N}(3.5,2.1)} < 0] &= Pr[\mathcal{I}_{\mathcal{N}(12,0)} + \mathcal{Y}^1_{\mathcal{N}(-8.5,2.1)} < 0] &\approx 0.00786 \\ Pr[\mathcal{Z}^2_{\mathcal{N}(3,1.5)} < 0] &= Pr[\mathcal{I}_{\mathcal{N}(12,0)} + \mathcal{Y}^2_{\mathcal{N}(-9,1.5)} < 0] &\approx 0.00715 \end{aligned}$$

The resulting probabilities show, that the path a_1, a_3, a_4, a_5 is the critical one, because it is more likely to exceed the total duration time.

4 Propagation

As described above, duration random variables are annotated to primitive tasks only. This means an HTN planner has to arrive at a primitive task network before it is able to reason about durations at all. However, it would perform much better if it were able to reason about the duration of abstract tasks and their possible refinements as well. One possibility to enable this would be to integrate the required information into the domain model, i.e. the domain modeler declares the abstract “duration” by hand. This alternative may be very time-consuming and may often lead to flawed models. We propose to propagate the abstract duration automatically, instead. Our approach guarantees that the propagated values are always under-estimations and can therefore be used to compute heuristic values for abstract plans. We will describe how the propagation works, and will show how this information is used to prune the search space during the planning process.

Note that, we can compute the duration of a given task network if all tasks in the network are assigned a random variable describing their duration. To compute an underestimation for an abstract task, we need to know the durations $\mathcal{X}_{\mathcal{N}(\mu_i,\sigma_i)}$ of all task networks the abstract task can be decomposed into. Given these durations, the underestimation for the abstract task is defined by the $\min$ function:

$$\min\left(\mathcal{X}_{\mathcal{N}(\mu_i,\sigma_i^2)}\right) := \mathcal{Y}_{\mathcal{N}\left(\min\limits_i(|\mu_i|),\min\limits_i(\sigma_i^2)\right)}$$

It can easily be shown that the random variable $\mathcal{Y}$ which describes the duration of the abstract task is always an underestimation of the real duration. This is because the density of $\mathcal{Y}$ has the minimal mean value and variance of all possible decompositions. This means, the real duration will be equal or greater than the duration described by $\mathcal{Y}$.

The propagation algorithm terminates if all tasks in the domain model are non-recursive, because the propagation can be done bottom-up starting with the task networks containing only primitive tasks. In the case of recursive tasks definitions, we cannot reason about all possible decompositions of the abstract recursive task. Nevertheless, to get an underestimation for recursive tasks we compute the minimum of the termination cases for the recursion. The result of this computation is a legal underestimation, because the termination case has to appear at least once in the recursion.

```
propagation:
1 process_primitive_tasks
2 mark_recursion_cycles
3 while not finished() do
4   process_task-nets
5   process_tasks
6 done
```

```
a.1 process_task-nets:
a.2 ∀ task-nets tn do
a.3  if not processed(tn)
a.4   if (∀t∈tn: processed(t))
a.5    underestimation(tn) =
       sum_critical_path(tn)
a.6 done
```

```
b.1 process_tasks:
b.2 ∀ tasks t do
b.3  if not processed(t)
b.4   if (∀methods m(t,tn_i):
           processed(tn_i))
b.5    underestimation(t) =
       min(underestimation(tn_i))
b.6 done
```

Figure 2: The three core procedures of the propagation algorithm.

Figure 2 shows the propagation algorithm. In line 1 the primitive tasks get their underestimations, which is the random variable of the corresponding operator. This procedure also sets `processed` true for every primitive task. The recursion cycles are marked in line 2, with a standard depth-first-search. The main loop (lines 3 to 6) of the algorithm terminates if finished becomes true. This is the case if all tasks and task networks are processed (i.e.

if all of them have their underestimation). The `process_task-nets` procedure called in line 4 is one of the main procedure of the algorithm. It is shown in Figure 2. It iterates over all task networks (a.2–a.6) and searches for not yet processed task networks (a.3). If such a task network is found, the procedure tries to compute an underestimation for it. To do so, all tasks in the network have to have a proper random variable for their duration (a.4). If all these conditions are met the underestimation for the task network is processed by the critical-path algorithm described in the section above.

The second core procedure is `process_task` (also shown in Figure 2). The main-loop (b.2–b.6) of this procedure is similar to that in `process_task-nets`. The loop iterates over all tasks in the domain model which are not already processed (b.3). To compute the underestimation for the current task, all task networks into which the task can be decomposed in have to be processed (b.4). The underestimation is then computed by the $\min$ function described above.

As it is not easy to see that the algorithm always terminates, we will give a sketch of a proof. Suppose we are given a legal domain model, i.e. one which among other properties includes at least one task network which contains only primitive tasks. After processing the primitive tasks, all such task networks get assigned an underestimation. I.e. that exists at least one abstract task which can be decomposed into a task network containing only primitives (recursive methods have been “eliminated” during pre-processing). In every run of the main loop there is therefore at least one abstract task for which the duration can be propagated. The complexity of this procedures is discussed in [1].

5 Project Planning for Software Development

Our example domain is taken from the daily work of a fictitious small software company which develops end-user tailored business software. In this domain we find much procedural knowledge capturing well proven “best practice”. However, these routines offer certain degrees of freedom which are hard to overview even for relatively simple applications.

Figure 3 shows what the planning domain for such projects can look like if we follow a (somewhat simplified) waterfall-based development approach: Each project is basically divided into 4 consecutive phases in which the requirements for the application are specified and refined, a software design is chosen and implemented, the resulting system is thoroughly tested, and finally installed on the customer’s hardware.

As one example for the variation in the procedure, we can identify 4 different ways to

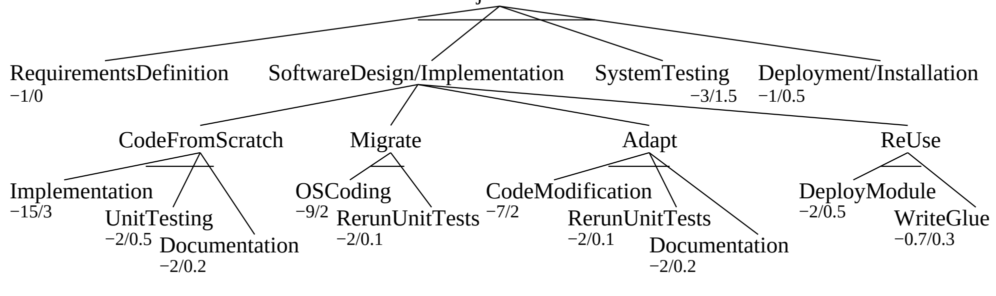

Figure 3: The decomposition hierarchy for the software project domain (numbers denote mean value and variance for the duration of primitive tasks).

perform the design and implementation task. The company can develop a completely new piece of software "from scratch", it can re-use an existing component unchanged, functionally extended or adapted, or it can re-implement software written for another operating system (migrate). All these alternatives imply different sub-tasks for documenting newly written code, testing, etc. For the presentation in this paper we omit many of the possible details, so the presented section of the decomposition hierarchy only shows the primitive tasks' duration annotations and no further preconditions or effects (e.g., only components can be installed which have been implemented for the respective operating system, components have to match requirement specifications, etc.). Concerning the precision of the distribution parameters, it has to be noted, that in general there exist relatively good estimations about how much time an action needs on average and typical fluctuations of it. Alternatively, the parameters can be reliably determined from the companies project history by standard statistical methods, e.g. the maximum-likelihood estimate.

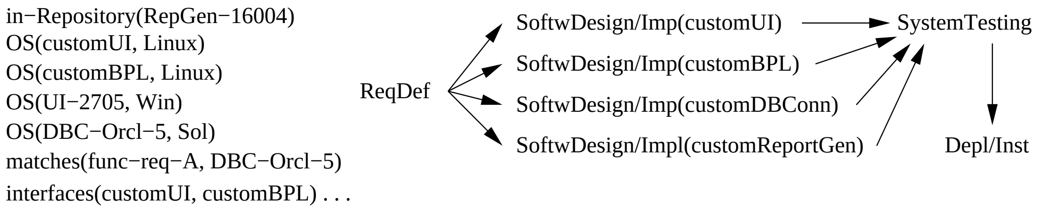
in−Repository(RepGen−16004)
OS(customUI, Linux)
OS(customBPL, Linux)
OS(UI−2705, Win)
OS(DBC−Orcl−5, Sol)
matches(func−req−A, DBC−Orcl−5)
interfaces(customUI, customBPL) . . .
ReqDef
SoftwDesign/Imp(customUI)
SoftwDesign/Imp(customBPL)
SoftwDesign/Imp(customDBConn)
SoftwDesign/Impl(customReportGen)
SystemTesting
Depl/Inst

Figure 4: The initial state and the initial task network in the software project domain.

Planning problems in this domain specify which software components are ordered by the customer: in our example it is a Linux application consisting of a user interface, a component implementing the respective business process logic, a database connection interface, and a report generator. The initial state specifies, among other things, the components present in the developers' repository, the operating system the component is specified for, for which functional requirements a component is suitable, and how the components interface each other (cf. Figure 4). Furthermore, the developers are given a total project deadline in 36 days plus a heavy contractual penalty fee if this deadline is missed by more than 4 more days. Given this situation, the project team is willing to accept an 85% probability of success for the first deadline, but they want to be more than 99% sure to meet the penalty deadline. We will start our analysis focusing on the first deadline.

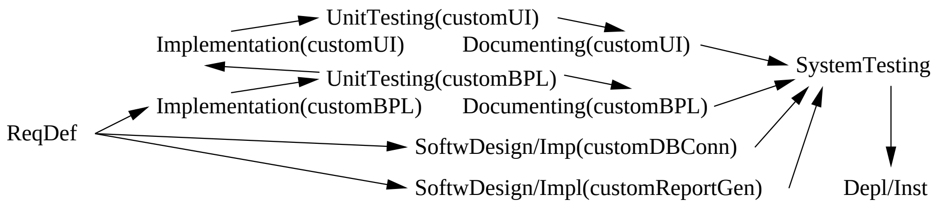

Figure 5: The first expansions in the initial task network.

After the first expansions, the resulting task network depicted in Figure 5 is under analysis. As no requirement matching modules for the user interface and business logic modules are in the repository, the system develops this (partial) plan for implementing both from scratch. The *interfaces* relation between the components enforces the business logic module to be implemented and tested before the implementation of the user interface can be done. Consequently, the critical path contains the two implementation tasks, resulting in a duration distribution for the network of $\mathcal{N}(-5, 9.2)$. This means, that the probability of meeting the first deadline is only $Pr[\mathcal{X}_{\mathcal{N}(-5,9.2)} \geq 0] = 4.96\%$

At this stage the plan is already unacceptable for the team. Therefore, the search continues with another expansion possibility, namely adapting an existing user interface instead. Figure 6 shows the task network after some more expansion steps: a suitable database interface exists for a related operating system, and an existing report generator could be re-used in the application.

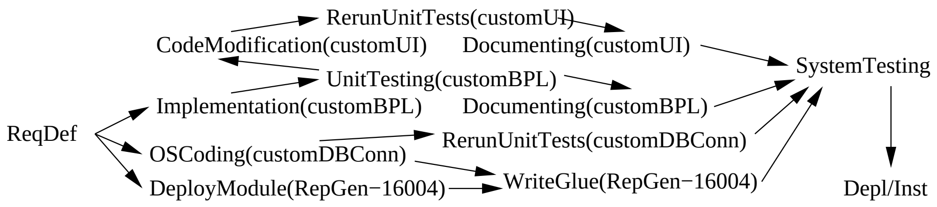

Figure 6: The final expansions in the initial task network.

The critical path includes the implementation and the adaptation with a distribution of $\mathcal{N}(7, 7.8)$, so we can assume that the team finishes its task within the given time of 36 days with a probability of $Pr[\mathcal{X}_{\mathcal{N}(7,7.8)} \geq 0] = 85.86\%$ Repeating the computation for the extended deadline of 40 days, the project leader can also be 99.39% sure that with the plan found, the company will not be sued for the delay. We see, that in contrast to an interval algebra based solution, the company's risk becomes quantifiable.

6 Implementation

The mechanisms for handling uncertain durations of operations are integrated in a multi-agent planning framework, originally developed for hybrid planning [2]. The basic architecture is depicted in Figure 7. Briefly, *detector* agents encapsulate checking routines which announce *flaws* in the shared blackboard data-structure, i.e. the current task network. Flaws consist in the presence of abstract tasks and un-satisfied preconditions of operators, among others. *Modifier* agents receive flaws and calculate solution proposals (e.g. task expansions, adding ordering constraints, etc.) which in turn are sent to the *strategy* agent. The strategy organizes the search by choosing such a task network modification at each step and executes it. After that the detectors are invoked again. Backtracking is initiated if no modification is issued for

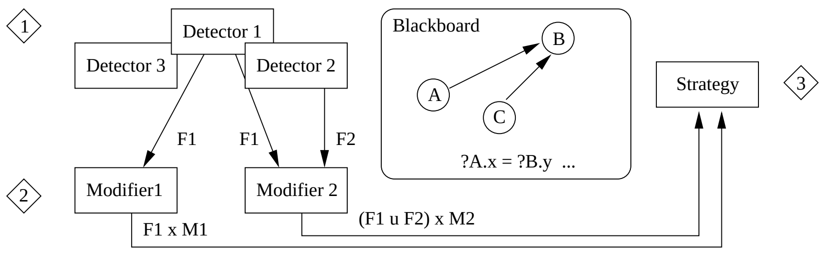

Figure 7: The components of the planning system architecture and their life-cycle.

at least one flaw. A deeper presentation of these mechanisms is beyond the scope of this paper, but it has to be noted that the representation of modification proposals allows the strategy to identify areas of interest in the plan (to guide its search), and that for our purposes it does not have to be adapted to make use of information delivered by additional agents. The framework has also been used to integrate scheduling capabilities in a similar way [11].

The integration of temporal uncertainty is done by utilizing the independence of the single agents. We just add a *duration-detector* which reasons about the uncertain time needed to execute the plan like it is described above. It roughly works as follows: After its invocation, the agent calculates the duration $\mathcal{Z}$ of the critical path for the (incomplete) plan on the blackboard. The presented propagation procedure allows for this on every level of abstraction. Note, that $\mathcal{Z}$ has a negative mean value, as it stands for usage of time. In the next step the agent adds the maximum allowable duration for the plan from the problem description to $\mathcal{Z}$ and determines the resulting random variable $\mathcal{Y}$. $\mathcal{Y}$ represents the time difference between the time limit and the expected duration of the plan. The probability of failure of the plan (w.r.t. the duration) is consequently $Pr[\mathcal{Y} < 0]$: the probability for exceeding the time limit. If this probability is higher than the given threshold, the duration-agent generates a *duration-exceeded-flaw*. As there is no modifier in the agent community to propose a modification step to fix this kind of flaw, the planner has to initiate backtracking.

By propagating duration under-estimations up to the abstract task level, the detector is able to compute a heuristic value for the duration on every level of abstraction during plan generation. To prune the search space, an abstract plan that already exceeds the maximum duration can be rejected at any stage, as the duration propagation always under-estimates.

7 Related Work

Related approaches in the literature mainly use intervals to represent temporal uncertainty.

β-robust scheduling for single machines is presented in [6], where the total flow time of all scheduled jobs is minimized. In this context, information is gathered about the execution time of single tasks and the duration of the abstract action is estimated by a maximum likelihood, the result of which is a random variable. A fast heuristic function for scheduling performance is compared with correct but slow computations, and it is shown how to select the schedule which promises the best performance.

A lot of work is done in the field of handling *discrete* probabilities in planning, of which we address that about epsilon-safe planning here [8]. It deals with the feature of uncertain sensing actions and introduces an approach to generate an ε-safe plan, which means to generate a plan that has only a probability of ε to fail in execution.

O-Plan, e.g., performs an optimistic and a pessimistic estimation of each resource profile [7]. If the optimistic profile gets below zero, i.e. if all consumption steps are performed as late as possible allocating the minimal quantity possible and all production steps are performed as early as possible producing as much as possible, and there is still a point in time in the plan where the capacity is exceeded, then this plan cannot be repaired and search has to backtrack. Furthermore, O-Plan can introduce constraints to evade potentially conflicting plans.

Simple Temporal Networks (STN) are constraint networks, in which nodes are time points and edges represent constraints like upper and lower bounds. [9] introduces the Temporal Constraint Satisfaction Problem (TCSP) in which a preference value is added to the temporal constraints. As solving the TCSP is np-hard, a restriction on convex intervals is presented, which can be solved in polynomial time. Another extension to STN is presented in [10], which focuses on the execution of STN w.r.t. events of uncertain timing. It describes how the STN has to be adapted if uncertain timing occurs during execution, and how the existing procedures for this problem can be improved in this way.

Alternatively, duration uncertainty can be modeled qualitatively by using fuzzy temporal nets. Approaches like [4] use them to obtain approximate solutions.

The presented idea on heuristic propagation of operator durations is loosely based on [5], in which intervals on abstract tasks are used as a heuristic function for resource consumption on the action layer. But uncertain resources consumption in operators is not discussed.

8 Conclusion

We have described an approach to handle uncertainty w.r.t. time consumption of operations in HTN planning. Durations are represented by continuous normal-distributed random variables. By adopting appropriate stochastic concepts the duration probabilities of critical execution paths can be accumulated. With that, overall probabilities for the successful execution of non-linear plans/task networks can be computed. We have shown how it can be embedded into an multiagent-based planning framework, where it allows for the derivation and use of heuristics to guide the decomposition-based planning process towards solutions that meet a certain probability threshold.

References

[1] Susanne Biundo, Roland Holzer, and Bernd Schattenberg. Dealing with continuous resources in AI planning. In *Proc. of the 4th Intern. Workshop on Planning and Scheduling for Space*, 2004. to appear.

[2] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In A. Cesta and D. Borrajo, editors, *Proc. of the 6th European Conf. on Planning (ECP-01)*, LNCS, pages 157–168. Springer Verlag, 2001.

[3] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Planning under continuous time and resource uncertainty: A challenge for AI. In A. Darwiche and N. Friedman, editors, *Proc. of the 18th Conf. in Uncertainty in AI*, pages 77–84. Morgan Kaufmann, 2002.

[4] L. Castillo and J. Fdez.-Olivares and A. Gonzalez. A flexible temporal planner In *Iberamia 2002, I Workshop on Planning, Scheduling and Temporal Reasoning*, pages 91–102, 2002.

[5] Bradley J. Clement, Anthony C. Barrett, Gregg R. Rabideau, and Edmund H. Durfee. Using abstraction in planning and scheduling. In A. Cesta and D. Borrajo, editors, *Proc. of the 6th European Conf. on Planning (ECP-01)*, LNCS, pages 145–156. Springer Verlag, 2001.

[6] Richard Daniels and Janice Carrillo. β-robust scheduling for single-machine systems with uncertain processing times. *IIE Transactions*, 29(11):977–985, 1997.

[7] Brian Drabble and Austin Tate. The use of optimistic and pessimistic resource profiles to inform search in an activity based planner. In K. Hammond, editor, *Proc. of the 2nd Intern. Conf. on AI Planning Systems (AIPS-94)*, pages 243–248. AAAI Press, 1994.

[8] Robert P. Goldman and Mark S. Boddy. Epsilon-safe planning. In R. López de Mántaras and D. Poole, editors, *Proc. of the 10th Annual Conf. on Uncertainty in AI*, pages 253–261. Morgan Kaufmann, 1994.

[9] Lina Khatib, Paul H. Morris, Robert A. Morris, and Francesca Rossi. Temporal constraint reasoning with preferences. In B. Nebel, editor, *Proc. of the 17th Intern. Joint Conf. on AI (IJCAI-01)*, pages 322–327, Morgan Kaufmann, 2001.

[10] Paul Morris and Nicola Muscettola. Execution of temporal plans with uncertainty. In *Proc. of the 17th National Conf. on AI (AAAI-2000)*. AAAI Press, July 2000.

[11] Bernd Schattenberg and Susanne Biundo. On the identification and use of hierarchical resources in planning and scheduling. In M. Ghallab, J. Hertzberg, and P. Traverso, editors, *Proc. of the 6th Intern. Conf. on AI Planning and Scheduling (AIPS'02)*, pages 263–272. AAAI Press, 2002.