Hybrid Multi-agent Planning

Mohamed Elkawkagy and Susanne Biundo

Dept. of Artificial Intelligence
Ulm University, D-89069 Ulm, Germany
<firstname>.<lastname>@uni-ulm.de

Abstract. Although several approaches have been constructed for multi-agent planning, solving large planning problems is still quite difficult. In this paper, we present a new approach that integrates landmark preprocessing technique in the context of hierarchical planning with multi-agent planning. Our approach uses `Dependent` and `Independent` clustering techniques to break up the planning problem into smaller clusters. These clusters are solved individually according to landmark information, then the obtained individual plans are merged according to the notion of fragments to generate a final solution plan. In hierarchical planning, landmarks are those tasks that occur in the decomposition refinements on every plan development path. Hierarchical landmark technique shows how a preprocessing step that extracts landmarks from a hierarchical planning domain and problem description can be used to prune the search space that is to be explored before actual search is performed. The methodologies in this paper have been implemented successfully, and we will present some experimental results that give evidence for the considerable performance increase gained through our system.

1 Introduction

Multi-agent planning (MAP) has been used to solve large planning problems. It works by splitting the given planning problem into subproblems. Each subproblem is solved individually to produce a solution so-called *subplan*. Then, these subplans have to be combined to construct the solution plan [14]. Furthermore, MAP has been used to interpret the plan coordination process of a set of independent agents. There are three different approaches which discuss the plan coordination process. The first one focuses on the coordination between completed plans such as the work of Tonino et al. [12] that achieves a less costly merging plan by exploiting positive interactions and resolving the conflicts between the generated subplans. In the second one, the processes of coordination and planning are interleaved i.e. the conflicts between subplans are resolved before each agent generates its subplan [13]. The third one is divided into two categories: implicit coordination that propagates general rules to manage agent behavior [15], and explicit coordination that allows for the exchange of information between agents before planning is started and provides additional constraints to the original planning problem to ensure that the generated solution plan is feasible [10].
Many researchers have used hierarchical structure in the MAP approach to improve planning efficiency. NOAH [3] is the first system which was built to interleave the hierarchical planning and merging process by exchanging shared resources. It was developed by focusing on the efficient communication among planner agents [4]. Afterwards,

a new method has been developed to detect the conflicts not only at the primitive levels, but also at abstract levels [2]. Recently, Hisashi [8] presented an HTN planning agent system working in dynamic environments. The set of agents in his approach are arranged in stratified form as parent and children agents which work together to achieve the goal. As opposed to these approaches, integrating MAP with hierarchical preprocessing techniques which prune the search space of a hierarchical planner has not been considered so far. Recently, we have introduced the hierarchical landmark technique for the purpose of domain model reduction [5]. In hierarchical planning, *landmarks* are mandatory abstract or primitive tasks that have to be performed by any solution plan.
In this paper, we present a novel hybrid approach that combines the landmark preprocessing technique in the context of hierarchical planning with multi-agent planning in order to enhance planning efficiency. Our architecture consists of a set of agents. The ***pre-processing agent*** analyzes a given planning problem by applying a landmark algorithm in hierarchical planning. It does so by systematically inspecting the methods that are eligible to decompose the relevant abstract tasks. Beginning with the (landmark) tasks of the initial plan, the procedure follows the way down the decomposition hierarchy until no further abstract tasks qualify as landmarks. The ***master agent*** can be divided into two parts: the first part handles a split process and the second handles a merging process. Finally, the ***slave agents*** are a set of identical agent planners that are executed concurrently, they do not cooperate among each others and each one of them uses HTN-style planning to generate its own individual plan.
Before introducing our approach in section 3, we will review the underlying planning framework in section 2. Section 4 presents the merging technique that combines individual plans to generate a solution plan. Section 5 describes the experimental setting and the evaluation results. The paper ends with some concluding remarks in section 6.

2 Formal Framework

HTN planning relies on the concepts of tasks and methods [7]. Primitive tasks correspond to classical planning operators, while abstract tasks represent complex activities. For each abstract task, a number of methods are available each of which provides a task network, *i.e., a plan that specifies a pre-defined (abstract) solution of the task*. Planning problems are (initial) task networks. They are solved by incrementally decomposing the abstract tasks until the network contains only primitive ones in executable order.
Our planning framework relies on a *hybrid* formalization [1] which combines HTN planning with partial-order causal-link (POCL) planning. For the purpose of this paper, only the HTN shares of the framework are considered, however. A task schema $t(\overline{\tau}) = \langle \mathrm{prec}(t(\overline{\tau})), \mathrm{add}(t(\overline{\tau})), \mathrm{del}(t(\overline{\tau}))\rangle$ specifies the preconditions and effects of a task via conjunctions of literals over the task parameters $\bar{\tau} = \tau_1 \ldots \tau_n$. States are sets of literals. Applicability of tasks and the state transformations caused by their execution are defined as usual. A *plan* $P = \langle S, \prec, V, CL\rangle$ consists of a set S of *plan steps* – (partially) instantiated task schemata that carry a unique label to differentiate between multiple occurrences of the same schema –, a set $\prec$ of *ordering constraints* that impose a partial order on S, a set V of *variable constraints*, and a set CL of *causal links*. V consists of (in)equations that associate variables with other variables or constants; it also reflects

the (partial) instantiation of the plan steps in P. We denote by $Ground(S, V)$ the set of ground tasks obtained by equating all parameters of all tasks in P with constants, in a way compatible with V. The causal links are adopted from POCL planning: a causal link $s_i \rightarrow_\varphi s_j$ indicates that φ is implied by the precondition of plan step s_j and at the same time is a consequence of the effects of plan step s_i. Hence, the precondition φ is said to be *supported* this way. Methods $m = \langle t(\bar{\tau}), P \rangle$ relate an abstract task $t(\bar{\tau})$ to a plan P, which is called an *implementation* of $t(\bar{\tau})$. In general, multiple methods are provided for each abstract task. Please also note that no application conditions are associated with the methods, as opposed to other representatives of HTN-style planning.
An HTN planning problem $\Pi = \langle D, s_{init}, P_{init} \rangle$ is composed of a domain model $D = \langle T, M \rangle$, where T and M denote sets of task schemata and decomposition methods, an initial state s_{init}, and an initial plan P_{init}. Note, that in our *hybrid* planning framework, one can specify a goal state. However, since we restrict ourselves in this paper to pure HTN planning, the goal state is omitted. A plan $P = \langle S, \prec, V, CL \rangle$ is a solution to Π if and only if: (1) P is a refinement of P_{init} i.e., a successor of the initial plan in the induced search space (see Def. 1 below), (2) each precondition of a plan step in P is supported by a causal link in CL and no such link is threatened, i.e., for each causal link $s_i \rightarrow_\varphi s_j$ the ordering constraints in $\prec$ ensure that no plan step s_k with an effect that implies $\neg\varphi$ can be placed between plan steps s_i and s_j, (3) the ordering and variable constraints are consistent, i.e., $\prec$ does not induce cycles on S and the (in-) equations in V are not contradictory, and (4) all plan steps in S are primitive ground tasks. Please note that we encode the initial state via the effects of an artificial primitive task, as it is usually done in POCL planning. In doing so, the second criterion guarantees that the solution is executable in the initial state.
In order to refine the initial plan into a solution, there are various *refinement steps* (or *plan modifications*) available; in HTN planning, these are: (1) the decomposition of abstract tasks using methods, (2) the insertion of causal links to support open preconditions of plan steps, (3) the insertion of ordering constraints, and (4) the insertion of variable constraints. Given an HTN planning problem Π, its initial plan and the available plan modifications we can define the induced search space as follows.

Definition 1 (Induced Search Space) *The directed graph $\mathcal{P}_\Pi = \langle \mathcal{V}_\Pi, \mathcal{E}_\Pi \rangle$ with vertices $\mathcal{V}_\Pi$ and edges $\mathcal{E}_\Pi$ is called the induced search space of the planning problem Π iff* (1) *$P_{init} \in \mathcal{V}_\Pi$, (2) if there is a plan modification refining $P \in \mathcal{V}_\Pi$ into a plan P', then $P' \in \mathcal{V}_\Pi$ and $(P, P') \in \mathcal{E}_\Pi$, and (3) $\mathcal{P}_\Pi$ is minimal such that (1) and (2) hold. For $\mathcal{P}_\Pi$, we write $P \in \mathcal{P}_\Pi$ instead of $P \in \mathcal{V}_\Pi$. In general, $\mathcal{P}_\Pi$ is neither acyclic nor finite.*

In order to search for solutions, the induced search space is explored in a heuristically guided manner by our refinement planning algorithm (Alg. 1).
The fringe $\langle P_1 \dots P_n \rangle$ is a sequence containing all unexplored plans that are direct successors of visited non-solution plans in $\mathcal{P}_\Pi$. It is ordered in a way such that a plan P_i is estimated to lead more quickly to a solution than plan P_j for $j > i$. The current plan is always the first plan of the fringe. The planning algorithm iterates on the fringe as long as no solution is found and there are still plans to refine (line 1). Hence, the flaw detection function f^{FlawDet} in line 2 calculates all flaws of the current plan. A flaw is a set of plan components that are involved in the violation of a solution criterion. The presence of an abstract task raises a flaw that consists of that task, a causal threat consists of a

causal link and the threatening plan step, for example. If no flaws can be found, the plan is a solution and returned (line 3). In line 4, the modification generating function f^{ModGen} calculates all plan modifications that address the flaws of the current plan. Afterwards, the modification ordering function f^{ModOrd} orders these modifications according to a given strategy. The fringe is finally updated in two steps. First, the plans resulting from applying the modifications are computed (line 5) and put at the beginning of the fringe (line 6). Second, the plan ordering function f^{PlanOrd} orders the updated fringe. This step can also be used to discard plans, i.e., to delete plans permanently from the fringe. This is useful for plans that contain unresolvable flaws such as an inconsistent ordering of tasks. If the fringe becomes empty, no solution exists and `fail` is returned.

Algorithm 1: Refinement Planning Algorithm

input : The sequence $\texttt{Fringe} = \langle P_{init} \rangle$.
output: A solution or `fail`.

1 **while** $\texttt{Fringe} = \langle P_1 \dots P_n \rangle \neq \varepsilon$ **do**
2 $\quad F \leftarrow f^{\text{FlawDet}}(P_1)$
3 $\quad$ **if** $F = \emptyset$ **then return** P_1
4 $\quad \langle \mathtt{m}_1 \dots \mathtt{m}'_n \rangle \leftarrow f^{\text{ModOrd}}(\bigcup_{\mathtt{f} \in F} f^{\text{ModGen}}(\mathtt{f}))$
5 $\quad \texttt{succ} \leftarrow \langle apply(\mathtt{m}_1, P_1) \dots apply(\mathtt{m}'_n, P_1) \rangle$
6 $\quad \texttt{Fringe} \leftarrow f^{\text{PlanOrd}}(\texttt{succ} \circ \langle P_2 \dots P_n \rangle)$
7 **return** `fail`

3 Hybrid Multi-agent Planning

Figure 1 depicts the components of our architecture. It consists of the pre-processing agent and the planning agents. The planning agents encapsulate the master agent and slave agents as well as a shared constraints set in their context. The shared constraints set (SC) is a shared memory that includes a set of constraints. Note, that all agents have complete knowledge about the initial state of the planning problem. In the following subsections, we will illustrate the key features of our agents.

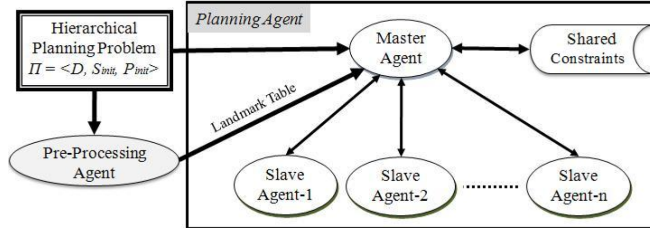

Fig. 1: Hybrid Multi-agent Planning Architecture

3.1 Pre-processing Agent

The pre-processing agent uses a preprocessing technique in order to perform some pruning of the search space before the actual search is performed in order to reduce the planning effort. Recently, we introduced a landmark technique which restricts the domain and problem description of an HTN to a smaller subset, since some parts of the domain

description might be irrelevant for the given problem at hand [5, 6]. Therefore, the pre-processing agent relies on hierarchical landmarks – ground tasks that occur in the plan sequences leading from an initial plan to its solution. They are defined as follows.

Definition 2 (Solution Sequences) *Let $\langle \mathcal{V}_\Pi, \mathcal{E}_\Pi \rangle$ be the induced search space of planning problem Π. Then, $\mathcal{S}ol\mathcal{S}eq_\Pi(P) := \{\langle P_1 \dots P_n \rangle | P_1 = P, (P_i, P_{i+1}) \in \mathcal{E}_\Pi$ for all $1 \leq i < n$, and $P_n \in \mathcal{S}ol_\Pi, n \geq 1\}$.*

Definition 3 (Landmark) *A ground task $t(\overline{\tau})$ is called a* landmark *of planning problem Π, if and only if for each $\langle P_1 \dots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P_{\text{init}})$ there is an $1 \leq i \leq n$, such that $t(\overline{\tau}) \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, CL_i \rangle$ and $P_n = \langle S_n, \prec_n, V_n, CL_n \rangle$.*

While a landmark occurs in every plan sequence that is rooted in the initial plan and leads towards a solution, a *local* landmark occurs merely in each such sequence rooted in a plan containing a specific abstract ground task $t(\overline{\tau})$.

Definition 4 (Local Landmark of an Abstract Task) *For an abstract ground task $t(\overline{\tau})$ let $\mathcal{P}_\Pi(t(\overline{\tau})) := \{P \in \mathcal{P}_\Pi | P = \langle S, \prec, V, CL \rangle$ and $t(\overline{\tau}) \in Ground(S, V)\}$.*
A ground task $t'(\overline{\tau}')$ is a local landmark *of $t(\overline{\tau})$, if and only if for all $P \in \mathcal{P}_\Pi(t(\overline{\tau}))$ and each $\langle P_1 \dots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P)$ there is an $1 \leq i \leq n$, such that $t'(\overline{\tau}') \in Ground(S_i, V_n)$ for $P_i = \langle S_i, \prec_i, V_i, CL_i \rangle$ and $P_n = \langle S_n, \prec_n, V_n, CL_n \rangle$.*

Since there are only finitely many task schemata and we assume only finitely many constants, there is only a finite number of (local) landmarks. Given a planning problem Π, the relevant landmark information can be extracted in a pre-processing step. The outcomes of the extraction procedure that is introduced by Elkawkagy et al. [5] is stored in a so-called *landmark table*. Its definition relies on a *Task Decomposition Graph* (TDG). A TDG is a representation of all possible ways to decompose the initial plan P_{init} of Π using methods in M. More formally, a TDG of a planning problem Π is a bipartite graph $\langle V_T, V_M, E \rangle$, where V_T is a set of ground tasks (called *task vertices*), V_M is a set of methods (called *method vertices*), and E is the set of edges connecting task vertices with method vertices according to M. Note, that a TDG is finite as there are only finitely many ground tasks and a finite number of methods. The *landmark table* is a data structure that represents the TDG plus some additional information about local landmarks.

Definition 5 (Landmark Table) *Let $\langle V_T, V_M, E \rangle$ be a* TDG *of the planning problem Π. The* landmark table *of Π is the set $LT = \{\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle | t(\overline{\tau}) \in V_T\}$, where $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$ are defined as follows:*
$M(t(\overline{\tau})) = \{t'(\overline{\tau}') \in V_T \mid t'(\overline{\tau}') \in Ground(S, V)$ *for all* $\langle t(\overline{\tau}), \langle S, \prec, V, CL \rangle \rangle \in V_M\}$
$O(t(\overline{\tau})) = \{Ground(S, V) \setminus M(t(\overline{\tau})) \mid \langle t(\overline{\tau}), \langle S, \prec, V, CL \rangle \rangle \in V_M\}$

Each landmark table entry partitions the tasks introduced by decompositions into two sets. Mandatory tasks $M(t(\overline{\tau}))$ are those ground tasks that are contained in all plans introduced by some method which decomposes $t(\overline{\tau})$; hence, they are local landmarks of $t(\overline{\tau})$. The optional task set $O(t(\overline{\tau}))$ contains for each method decomposing $t(\overline{\tau})$ the set of ground tasks which are not in the mandatory set.
Once the landmark table has been constructed, the pre-processing agent terminates itself after sending the landmark table to the master agent. Please note that the information

about landmarks can be exploited in two ways. The first way is to reduce the domain model by ignoring infeasible method decompositions or, more precisely, to transform a universal domain model into one that includes problem-specific pruning information [5]. The second application of the landmark table is to serve as a reference for the planning strategy to deduce heuristic guidance from the knowledge about which tasks have to be decomposed on refinement paths that lead towards a solution [6]. Each slave agent will use the former way in order to construct its individual solution plan.

3.2 Planning Agents

Our planning scenario includes a set of planning agents. They integrate to generate a final solution for the given planning problem. The set of planning agents is divided into two different types: a master agent and a set of identical agents so-called *slave agents*.

Master Agent: It takes a hierarchical planning problem Π and the computed landmark table LT as input and generates the final solution plan. To this end, the master agent performs two processes: a split process and a merging process.
In the split process, the master agent decomposes a planning problem Π into a set of clusters according to two different techniques: `Dependent` (`Dep`) and `Independent` (`Ind`). The `Dep` technique relies on the idea of constructing a set of dependent clusters, while the `Ind` technique creates a set of independent clusters. A comparison of the respective results will be done in the experimental section (cf. Sec. 5).
In each iteration of the `Dep` algorithm (Alg. 2), in the current plan, the tasks that are not preceded by other tasks are separated in one cluster. While the set of constraints between tasks in different clusters is inserted in the shared constraints set SC.
The `Dep` algorithm takes the planning problem Π, a landmark table LT *(i.e., the LT is computed by the pre-processing agent)*, and a set $\Gamma = \langle \Pi_\gamma,\ SC \rangle$ as input and computes the final set Γ. The set Γ consists of the shared constraints set SC and the set of clusters $\Pi_\gamma = \bigcup_{i=0}^{n} \{\Pi_{\gamma_i}\}$, where n is the number of clusters. Each cluster Π_{γ_i} will be considered as a sub-problem. In order to identify different clusters, the `Dep` algorithm runs recursively through all tasks in the initial plan P_{init} until all tasks have been traversed. Each cluster $\Pi_\gamma = \langle D, s_{init}, P_{\gamma_{init}}, LT_\gamma \rangle$ includes a domain model D, an initial state s_{init}, a partial plan $P_{\gamma_{init}}$ which represents an initial plan for the cluster Π_γ, and an LT_γ that represents relative landmark information of the tasks in the partial plan $P_{\gamma_{init}}$.
Now, we will have a look at how the set Γ is constructed by our `Dep` algorithm. First, the set Γ is initialized (line 1). Afterwards, in the current plan P_{init}, the tasks te_i that are pre-request-free are collected (line 3). In lines 4 to 6, a new cluster Π_{γ_i} is created, and the tasks in te_i are added to the task network of the plan $P_{\gamma_{init}}$. Then, the tasks te_i are removed from the task network TE of the plan P_{init}. For the current tasks te_i at hand, lines 7 to 12 illustrate iterative loops to extend the created cluster Π_{γ_i} by adding different constraints and updating SC by inserting new constraints. For each task t in the set of tasks te_i, the preprocessing information in the LT which is relevant to the current task t is added to the field LT_{γ_i} in the cluster Π_{γ_i}. This information in LT_{γ_i} will help the planner (slave) agent to reduce the planning effort necessary to find the individual plan for the cluster Π_{γ_i}(line 8). In lines 9 and 10, the plan $P_{\gamma_{init}}$ in the cluster Π_{γ_i} is extended by adding all variable V, ordering $\prec$ and CL constraints that point to a relation

between the current task t and the other tasks in the set te_i. After that, these constraints are removed from the current plan P_{init} in Π. Afterwards, the shared constraints set SC and the current partial plan P_{init} are updated by inserting and removing respectively all constraints that relate task t to other tasks outside the set of tasks te_i (lines 11 and 12). Finally, the current set Γ is updated by adding the current new cluster Π_{γ_i} to the set of cluster Π_γ as well as updating the shared constraints set SC (line 13). Note that SC will play a great role in the merging process. The `Dep` algorithm is called recursively with the modified plan and the updated Γ to inspect a new cluster (line 14).
On the other hand, in each iteration of the `Ind` algorithm, in the current plan, the tasks that are dependent are collected in one cluster and consequently SC get empty. Therefore, in order to split the planning problem into a set of clusters according to the `Ind` algorithm, we will replace line 3 in Algorithm 2 by the following line ***"Select tasks*** te_i ***that are dependent"***. Due to this replacement, the set of clusters which are produced are explicitly independent, and the shared constraints set SC will be empty.
Once the set of clusters has been generated either by the `Dep` or `Ind` algorithm, the master agent initiates a number of slave agents based on the number of clusters and distributes these clusters among slave agents in order to construct individual solution plans for them. Afterwards, the master agent suspends its activity until all slave agents respond by returning individual solution plans for all clusters, and then wakes up again to perform the merging process in order to generate a general solution plan (cf. sec. 4).

Algorithm 2: *Dependent (Dep) Algorithm*

Input : $\Pi = \langle D, s_{init}, P_{init}\rangle$: Planning problem,
LT: LandmarkTable, $\Gamma = \langle \Pi_\gamma, SC\rangle$
Output: Γ

1 $\Gamma = \langle \Pi_\gamma, SC\rangle \longleftarrow null$
2 **if** *($TE == \emptyset$)* **then return** Γ
3 Select tasks te_i that are pre-request-free from P_{init}.
4 Create a new cluster $\Pi_{\gamma_i} = \langle D, s_{init}, P_{\gamma_{init}}, LT_{\gamma_i}\rangle$.
5 Add these tasks te_i to the partial plan $P_{\gamma_{init}}$ in cluster Π_{γ_i}.
6 Let $TE \longleftarrow (TE - te_i)$
7 **foreach** *(task $t \in te_i$)* **do**
8 Attach the relevant information of the task t in the LT to the LT_{γ_i} in the cluster Π_{γ_i}
9 Add all constraints $(\prec_t, V_t, CL_t)$ that relate task t with the other tasks in te_i to $P_{\gamma_{init}}$
10 Let $V \longleftarrow (V - V_t)$; $\prec \longleftarrow (\prec - \prec_t)$; $CL \longleftarrow (CL - CL_t)$
11 Insert all constraints $(\prec_{\bar{t}}, V_{\bar{t}}, CL_{\bar{t}})$ that relate task t with another task $\bar{t} \notin te_i$ into SC.
12 Let $\prec \longleftarrow (\prec - \prec_{\bar{t}})$; $V \longleftarrow (V - V_{\bar{t}})$; $CL \longleftarrow (CL - CL_{\bar{t}})$
13 $\Gamma = \langle \Pi_\gamma, SC\rangle \longleftarrow \langle (\Pi_\gamma \cup \Pi_{\gamma_i}), SC\rangle$
14 **return Dependent**(Π, LT, Γ)

Slave Agents: It is a set of identical agents working concurrently in order to solve the set of clusters which are passed by the master agent. To this end, each slave agent uses our refinement planning algorithm (cf. Alg. 1) to generate its own individual plan. Our refinement algorithm takes the initial plan $P_{\gamma_{init}}$ of the assigned cluster as an input and refines it stepwise until the individual solution plan is found.
Since our approach is based on a *declarative* model of task abstraction, the exploitation of knowledge about hierarchical landmarks can be done *transparently* during the gener-

ation of the task expansion modifications: First, the respective modification generation function $f^{ModGen}_{AbsTask}$ is deployed with a reference to the landmark table LT_{γ_i} of the cluster problem Π_{γ_i}. During planning, each time an abstract task flaw indicates an abstract plan step $t(\bar{\tau})$ the function $f^{ModGen}_{AbsTask}$ does not need to consider all methods provided in the domain model for the abstract task $t(\bar{\tau})$. Instead, it operates on a reduced set of applicable methods according to the respective options $O(t(\bar{\tau}))$ in the LT_{γ}.
It is important to see that the overall plan refinement procedure is not affected by this domain model reduction, neither in terms of functionality *(flaw and modification modules do not interfere)* nor in terms of search control *(strategies are defined independently, and completeness of search is preserved).*
Note that if the refinement planning algorithm returns `fail` , the slave agent works as a master agent and performs all functions of the master agent. Finally, each slave agent terminates itself after sending the generated individual solution p_{γ} to the master agent.

4 Merging Methodology

Our merging methodology depends on the notion of *Fragments*. The merging technique proceeds in two processes. Firstly, the set of individual plans $P_{\Gamma} = \{p_{\gamma_1}, p_{\gamma_2}, \cdots, p_{\gamma_n}\}$ *(i.e., which are produced by slave agents)* is divided into a set of *Fragments*. These *Fragments* are constructed according to the ordering constraints in the SC. Each fragment $F = \langle P_{\gamma}, O_{\gamma}\rangle$ consists of a set of individual plans P_{γ} ($P_{\gamma} \subseteq P_{\Gamma}$), and a set of ordering constraints O_{γ} that impose a partial order on P_{γ}. For example, suppose the ordering constraints in SC are $\{s_1 \prec s_2, s_2 \prec s_3, s_4 \prec s_5\}$. Let a set of individual plans $P_{\Gamma} = \{p_{\gamma_1}, \cdots, p_{\gamma_7}\}$ be respectively a solution plan for abstract plan steps $s_1, \cdots, s_7$. Then according to the ordering information in SC, these plans in P_{Γ} constitute three different fragments $F_1 = \langle\{p_{\gamma_1}, p_{\gamma_2}, p_{\gamma_3}\}, \{p_{\gamma_1} \prec p_{\gamma_2}, p_{\gamma_2} \prec p_{\gamma_3}\}\rangle$, $F_2 = \langle\{p_{\gamma_4}, p_{\gamma_5}\}, \{p_{\gamma_4} \prec p_{\gamma_5}\}\rangle$, and $F_0 = \langle\{p_{\gamma_6}, p_{\gamma_7}\}, \{\emptyset\}\rangle$. Note that there are two types of fragments. *Related-Fragment* includes those individual plans that are dependent such as F_1 and F_2, and *Zero-Fragment* which includes those individual plans that do not have explicit dependency such as F_0.
Secondly, the plans in each fragment are merged to produce a plan so-called *Merge-Fragment-Plan (MFPlan)*, and then all *MFPlans* are combined in order to generate a general final solution plan. To this end, we need to identify the implicit dependency between individual plans especially in *Zero-Fragment*. This dependency is identified by matching preconditions and effects of the tasks in individual plans. This means, certain postcondition of plan p_{γ_i} tasks required as preconditions for plan p_{γ_j} tasks.
There are two reasons for determining the plan dependency: canceling the negative interactions *(one task deletes the effect or precondition of another task)*, and benefit from the positive interactions *(two different tasks need the same precondition and at least one of them does not remove it, or one task generates precondition of another task).*
Intuitively, the set of independent plans can be executed concurrently by integrating them into one large plan. Otherwise, if the implicit dependency between individual plans in *Zero-Fragment* is detected, then the master agent updates it by adding ordering constraints between these plans. Despite the order dependency between plans, some tasks in these plans can be performed concurrently. Note that tasks cannot take place

concurrently if the pre- or post-conditions of the tasks in the successor plan are inconsistent with the postconditions of the tasks in the predecessor plan [9]. Therefore, in order to determine the concurrent tasks, we will establish a comparison between pre- and post-conditions of tasks in the successor plan with the postconditions of the tasks in the predecessor plan. The comparison process will be started by checking pre- and post-conditions of the first task in the successor plan p_{γ_j} with postconditions of different tasks in the predecessor plan p_{γ_i}. If a pre- or post-condition of the first task in p_{γ_j} is violated, then this task will execute sequentially with plan p_{γ_i} and the procedure of case-1 will be performed. Otherwise, the first task will be executed concurrently with the plan p_{γ_i} and the procedure of case-2 will be performed. The comparison process in case-2 will go further in the successor plan p_{γ_j} in order to repeat the comparison with the next task. At this point, we neither need case-2 nor steps number 4 and 5 in case-1.
Case-1: (1) Create ordering constraint $\langle last\ task\ in\ p_{\gamma_i},\ current\ task\ in\ p_{\gamma_j}\rangle$, (2) Remove ordering constraint $\langle last\ task\ in\ p_{\gamma_i},\ goal()\ task\ in\ p_{\gamma_i}\rangle$, (3) Remove goal() task from p_{γ_i}, (4) Remove ordering constraint $\langle initial()\ task\ in\ p_{\gamma_j},\ current\ task\ in\ p_{\gamma_j}\rangle$, (5) Remove initial() task from p_{γ_j}, (6) Stop comparison process.
Case-2: (1) Remove ordering constraint $\langle initial()\ task\ in\ p_{\gamma_j},\ current\ task\ in\ p_{\gamma_j}\rangle$, (2) Remove initial() task in p_{γ_j}, (3) Create ordering constraint $\langle initial()\ task\ in\ p_{\gamma_i},\ current\ task\ in\ p_{\gamma_j}\rangle$, (4) Continue comparison with the next task in the successor plan p_{γ_j}.
On the other hand, in the merging process, we need to detect possible plan steps that will be merged and to update their constraints. The pair of plan steps in different plans is merged, if their postconditions are matched and there is no other task that is ordered after them that could violate these postconditions. More formally,

Definition 6 (Merging Plan Steps $merge(s_i, s_j)$**)** $\forall\ plan\ steps\ s_i \in P_{\gamma_i}\ and\ s_j \in P_{\gamma_j}$, $merge(s_i, s_j)\ iff\ ((post(s_i) = post(s_j)) \wedge (\neg\exists s_k \in P_{\gamma_j}\ violate\ post(s_j)\ s.t.\ (s_j \prec s_k)))$

Once a pair of plan steps has been merged, the related constraints of the removed plan step should be modified. This means that the replaced plan step will inherit all constraints of the merged plan steps as well as adding new constraints.
For all merged plan steps $s_j \in P_{\gamma_j}$ and $s_i \in P_{\gamma_i}$: (1) The plan step s_j is replaced by plan step s_i, (2) $\forall\ s_k, s_l \in P_{\gamma_j}$ and $\exists\ \langle s_k, s_j\rangle,\ \langle s_j, s_l\rangle \in \prec$ of plan P_{γ_j} add new order constraint $\langle s_k, s_l\rangle$ to plan P_{γ_j}, (3) Remove $\langle s_k, s_j\rangle,\ \langle s_j, s_l\rangle\ from\ plan\ P_{\gamma_j}$. These rules ensure that the whole ordering constraints of the merged plan step are preserved.
On the other hand, the causal link constraints that include the merged plan step in its components should be updated. For all merged plan steps $s_j \in P_{\gamma_j}$ and $s_i \in P_{\gamma_i}$: $\forall\ s_k, s_l \in P_{\gamma_j}$: (1) $\forall$ causal link $s_j \xrightarrow{\Phi} s_l \in CL$ of plan P_{γ_j} add new causal link $s_i \xrightarrow{\Phi} s_l$ to CL of plan P_{γ_j}, (2) remove causal link $s_j \xrightarrow{\Phi} s_l\ from\ CL\ of\ plan\ P_{\gamma_j}$, (3) remove causal link $s_k \xrightarrow{\Phi} s_j\ from\ CL\ of\ plan\ P_{\gamma_j}$.

5 Experimental Results

In order to quantify the practical performance gained by our approach, we conducted a series of experiments with our planning framework. The experiments were run on a machine with a 3 GHz CPU and 256 MB Heap memory for the Java VM. Note that

this machine has only one single processor unit. We ran our experiments on two well-established planning domains. The *Satellite* domain is a benchmark for non-hierarchical planning. It is inspired by the problem of managing scientific stellar observations by earth-orbiting instrument platforms. Our encoding of this domain regards the original primitive operators as implementations of abstract observation tasks, which results in a domain model with 3 abstract and 5 primitive tasks, related by 8 methods. The *UM-Translog* is a hierarchical planning domain that supports transportation and logistics. We adopted its type and decomposition structure to our representation which yielded a deep expansion hierarchy in 51 methods for decomposing 21 abstract tasks into 48 different primitive ones. We have chosen these domain models because of the problem characteristics they induce. On the other hand, *Satellite* problems typically become difficult when modeling a repetition of observations, which means that a small number of methods is used multiple times in different contexts of a plan. *UM-Translog* problems, on the other hand, typically differ in terms of the decomposition structure, because specific transportation goods are treated differently, e.g., toxic liquids in trains require completely different methods than transporting regular packages in trucks. We consequently defined our experiments on qualitatively different problems by specifying various transportation means and goods. The number of tasks in the initial plan of these planning problems ranges from one to six tasks.

Table 1: Results for the *UM-Translog* domain.

Problem	PANDA	PLM	HMAP					
			Dependent			Independent		
	Planning Time	Planning Time	Planning Time	Merging Time	Total	Planning Time	Merging Time	Total
Translog-P1	180	104	115	0	115	113	0	113
Translog-P2	155	99	103	0	103	105	0	105
Translog-P3	1450	153	159	0	159	157	0	157
Translog-P4	772	621	630	0	630	625	0	625
Translog-P5	1184	639	358	179	537	512	0	512
Translog-P6	-	3437	476	956	1432	1794	964	2758
Translog-P7	-	-	1413	2397	3810	703	1967	2670
Translog-P8	-	-	4562	6094	10656	1587	6731	8318
Translog-P9	-	-	454	148	602	450	0	450
Translog-P10	1284	583	451	941	1392	627	878	1505
Translog-P11	-	3930	2954	2769	5723	750	2343	3093
Translog-P12	-	-	3335	5981	9316	1218	3622	4840
Translog-P13	-	-	4370	6327	10697	1463	7250	8713
Translog-P14	-	-	770	223	993	673	351	1024
Translog-P15	-	-	1705	1440	3145	2109	1345	3454
Translog-P16	-	-	547	418	965	4785	578	5363
Translog-P17	-	-	3366	5921	9287	1328	4364	5692
Translog-P18	3268	1287	1079	0	1079	698	392	1090
Translog-P19	-	4184	3417	1002	4419	489	835	1324
Translog-P20	-	-	3692	2015	5707	1123	1910	3033
Translog-P21	-	-	4007	3842	7849	1379	4808	6187
Translog-P22	-	-	4705	5841	10546	1777	6370	8147
Translog-P23	5238	1211	832	0	832	383	376	759
Translog-P24	-	10006	3227	1045	4272	537	833	1370
Translog-P25	-	-	3445	2614	6059	686	1939	2625
Translog-P26	-	-	3874	5637	9511	1040	5481	6521
Translog-P27	-	-	4739	6627	11366	1521	5904	7425
Translog-P28	-	2623	1047	0	1047	2045	753	2798
Translog-P29	-	-	6008	697	6705	5069	3471	8540
Translog-P30	-	-	3237	940	4177	540	1014	1554

Tables 1 and 2 show the runtime behavior of our system in terms of the planning and merging time *(in seconds)* consumption for the problems in the *UM-Translog* and

Satellite domains, respectively. The planning time includes the time of breaking up the planning problem, the time used to solve sub-problems and the preprocessing time. Dashes indicate that the plan generation process did not find a solution within the allowed maximum number of $10,000$ plans and $18,000$ seconds and has therefore been canceled. The column PANDA refers to the reference system behavior [11], the PLM to the version that performs a preprocessing phase and HMAP to the version that performs our hybrid MAP. The column HMAP considers clustering the planning problem by two different clustering techniques `Dep` and `Ind`. Our experiments in the *UM-Translog* and *satellite* domains show poor performance (cf. Tables 1 and 2) in PANDA and PLM versions, as it is difficult to solve planning problems which have a large number of abstract tasks in the initial plan. The experiments show that, dividing the planning problem into smaller clusters either by `Dep` or `Ind` technique are easier to solve than the original problem. Consequently, we are able to solve the problems for which the competing systems could not find a solution within the given resource bounds. For example, for the *UM-Translog* problems that have a single abstract task in the initial plan *(Translog-P1 to P4)*, the average performance improvement of HMAP is about 59% in comparison with PANDA planner. In those problems, the PLM improves the results by 2% w. r. t. HMAP. Therefore, it is not a big disadvantage of using HMAP instead of PLM for problems which have a single task in the initial plan. Not surprisingly, the performance improvements will increase dramatically with the number of tasks in the initial plan. Our experiments proved that when there is a causal interaction between tasks in the plan, the `Ind` decomposition technique is more efficient than the `Dep` decomposition technique such as in the *UM-Translog* domain, where the `Ind` technique achieves an average improvement of 22% w. r. t. the `Dep` technique as documented in table 1.
Although, *satellite* domain does not benefit significantly from the landmark preprocessing technique due to the shallow decomposition hierarchy, it achieves good performance with decomposition techniques *(either `Dep` or `Ind` technique)* as depicted in table 2.

Table 2: Results for the *Satellite* domain.

Problem	PANDA	PLM	HMAP					
			`Dependent`			`Independent`		
	Planning Time	Planning Time	Planning Time	Merging Time	Total	Planning Time	Merging Time	Total
Satellite-P1	62	60	65	0	65	69	0	69
Satellite-P2	788	708	14	3	17	272	5	277
Satellite-P3	2035	2027	29	7	36	327	26	353
Satellite-P4	-	-	42	10	52	342	26	369
Satellite-P5	-	-	582	26	608	512	26	539
Satellite-P6	-	-	483	19	502	557	26	582
Satellite-P7	-	-	473	27	501	593	34	627
Satellite-P8	-	-	28	7	35	386	23	409
Satellite-P9	1699	1474	247	0	247	15	0	15
Satellite-P10	3053	3062	356	6	362	26	4	31
Satellite-P11	-	-	364	12	376	30	6	36
Satellite-P12	-	-	529	9	538	37	7	44
Satellite-P13	-	-	820	35	855	52	11	63
Satellite-P14	-	-	643	50	693	70	23	93

6 Conclusion

We have presented a new hybrid approach that integrates the hierarchical landmark preprocessing technique with MAP. Our approach enables us to break up the plan-

ning problem into a set of clusters using two different techniques; `Dependent` and `Independent`. It guarantees that: (1) the set of agents work independently, (2) the individually constructed plans are merged successfully in order to generate a global plan without additional refinement in any individual plan, and (3) the problems are solved in shorter time. We have performed a number of experiments on our representation framework on exemplary problem specifications for two hierarchical domains in which the HMAP approach competed with a planner "with and without" preprocessing. These results give evidence for the practical relevance of our approach.

ACKNOWLEDGEMENTS

This work is done within the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG).

References

1. Biundo, S., Schattenberg, B.: From abstract crisis to concrete relief (a preliminary report on combining state abstraction and HTN planning). In: Proc. of ECP. pp. 157–168 (2001)
2. Bradley, J., Edmund, H.: Theory for coordinating concurrent hierarchical planning agents using summary information. In: Proc. of AAAI. pp. 495–502 (1999)
3. Corkill, D.: Hierarchical planning in a distributed environment. In: Proc. of IJCAI. pp. 168–175 (1979)
4. desJardins, M., Wolverton, M.: Coordinating a distributed planning system. Journal of AI Magazine, 20(4). p. 4553 (1999)
5. Elkawkagy, M., Schattenberg, B., Biundo, S.: Landmarks in hierarchical planning. In: Proc. of ECAI. pp. 229–234 (2010)
6. Elkawkagy, M., Bercher, P., Schattenberg, B., Biundo, S.: Exploiting landmarks for hybrid planning. In: 25th PuK Workshop Planen, Scheduling und Konfigurieren, Entwerfen (2010)
7. Erol, K., Hendler, J., Nau, D.: UMCP: A sound and complete procedure for hierarchical task-network planning. Proc. of AIPS pp. 249–254 (1994)
8. Hisashi, H.: Stratified multi-agent HTN planning in dynamic environments. In: Proce. of KES-AMSTA. pp. 189–198 (2007)
9. Jeffrey, S., Edmund, D.: An efficient algorithm for multiagent plan coordination. In: Proc. of the AAMAS. pp. 828–835 (2005)
10. Mors, A.W., Valk, J.M., Witteveen, C.: Task coordination and decomposition in multi-actor planning systems. In: Proc. of the Workshop on Software-Agents in Information Systems and Industrial Applications (SAISIA). pp. 83–94 (2006)
11. Schattenberg, B.: Hybrid planning and scheduling. PhD thesis, The University of Ulm, Institute of Artificial Intelligence (2009)
12. Tonino, J., Bos, A., de Weerdt, M., Witteveen, C.: Plan coordination by revision in collective agent-based systems. Journal of Artificial Intelligence 142, 2 pp. 121–145 (2002)
13. Weerdt, M., Witteveen, C.: A resource logic for multi-agent plan merging. In: Proc. of the 20th Workshop of the UK planning and Scheduling. pp. 244–256 (2003)
14. Wilkins, D., Mayers, K.: A multi-agent planning architecture. In: Proc. of AIPS-98. pp. 154–162 (1998)
15. Yang, Q., Nau, D.S., Hendler, J.: Merging separately generated plans with restricted interactions. Journal of Computational Intelligence, 8(4): p. 648 676 (1992)