ulm university universität
uulm

Hierarchical Landmarks

A Means to Reduce Search Effort in Hybrid Planning

Dissertation zur Erlangung des Doktorgrades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Fakultät für Ingenieurwissenschaften und Informatik
der Universität Ulm

vorgelegt von

Mohamed Mohamed Nabil Mohamed Shams Eldeen Elkawkagy
aus Menoufyia, Ägypten
Institut für Künstliche Intelligenz
Fakultät für Ingenieurwissenschaften und Informatik
Universität Ulm

Institutsleitung
Prof. Dr. Susanne Biundo-Stephan

Ulm, Deutschland
July 2011

Amtierender Dekan der Fakultät für Ingenieurwissenschaften und Informatik:
Prof. Dr.-Ing. Klaus Dietmayer

Vorsitzender des Promotionsausschusses:
Prof. Dr. Uwe Schöning

Mitglieder des Promotionsausschusses:
Prof. Dr. Heiko Neumann
Prof. Dr. Helmuth A. Partsch

Die Gutachter der Dissertation:
Prof. Dr. Susanne Biundo-Stephan
Prof. Dr. Günther Palm

Tag der Promotion: 13 July 2011

ulm university universität
uulm

Hierarchical Landmarks

A Means to Reduce Search Effort in Hybrid Planning

A PhD thesis submitted to the Faculty of
Engineering and Computer Science, Ulm University
in fulfillment of the requirements for the degree of
Doktor rerum naturarum (Dr. rer. nat.)

by

Mohamed Mohamed Nabil Mohamed Shams Eldeen Elkawkagy
from Menoufyia, Egypt
Institute of Artificial Intelligence
Faculty of Engineering and Computer Science
Ulm University

Advisor

Prof. Dr. Susanne Biundo-Stephan

Ulm, Germany
July 2011

Dean of the Faculty of Engineering and Computer Science:
Prof. Dr.-Ing. Klaus Dietmayer

Chairman of the doctoral committee:
Prof. Dr. Uwe Schöning

Members of the doctoral committee:
Prof. Dr. Heiko Neumann
Prof. Dr. Helmuth A. Partsch

Reviewers of the dissertation:
Prof. Dr. Susanne Biundo-Stephan
Prof. Dr. Günther Palm

Day of Conferral of Doctorate: 13 July 2011

ABSTRACT

Artificial Intelligence planning is a key problem solving technology currently being used in a variety of applications including military campaigns, robot navigation, airplane scheduling, and human computer interaction. The generation of plans - courses of actions to achieve desired goals or perform specific tasks - is a costly process, however. Developing methods to systematically reduce the search effort and increase (the) performance of planning systems is thus a central concern.

In recent years, a number of approaches have been proposed to run a preliminary analysis of the artificial intelligence planning domain. They aim to extract and exploit knowledge from the domain model and problem description in order to reduce the planning effort. In general, pre-processing approaches can be done either "off-line" - analyzing the domain model before having access to a problem- or "on-line" - analyzing the domain model and planning problem description together.
We have developed a novel pre-processing technique to extract knowledge from a hierarchically structured planning domain and a current planning problem description which is used to significantly improve planning performance. This pre-processing technique enables pruning all branches which can be proven to never lead to a solution by identifying tasks that are not achievable from a certain initial situation.
The efficiency of planning systems depends on the kind of planning search strategy which the planner uses. We developed novel domain independent strategies relying on the knowledge that is generated by pre-processing in order to guide the hierarchical planning processes more effectively towards a solution of a given planning problem.
The complexity in real-world applications has led artificial intelligence planning researchers to develop algorithms and systems that more closely match realistic planning environments, in which planning activity is often distributed, and plan generation can happen concurrently with plan execution. Finally, our pre-processing technique in the context of hierarchical planning approach is integrated with a multi-agent based planning approach to decompose the original planning problem into a set of sub-problems each of which can then be solved separately. Our integration approach presents two different techniques to split the planning problem into a set of sub-problems: `Dependent` which constructs a set of dependent sub-problems and `Independent` which produces a set of independent sub-problems.

Our empirical evaluation shows that the pre-processing technique improves performance because the dead ends can be detected much earlier than without pruning and that our search strategies outperform many other possible strategies. In addition, the integration between the pre-processing technique and a multi-agent based planning approach dramatically reduce computation effort. In general, our empirical evaluation proves that our approach improves the efficiency of planning systems on the tested domains.

ACKNOWLEDGMENTS

Foremost, I would like to express my deep and sincere gratitude to my advisor Prof. Dr. Susanne Biundo-Stephan, *the director of the Institute of Artificial Intelligence, University of Ulm*, accepting me as a doctoral student at her institute, for the continuous support of my Ph.D study and research, for her patience, motivation, enthusiasm, and immense knowledge. Her guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study. I have enjoyed working with her.

Besides my supervisor, I would like to thank the rest of my defense committee for their guidance. My sincere thanks also go to Dr. Bernd Schattenberg for his valuable advice and friendly help. His extensive discussions around my work and he always had an answer when had a question.

During this work I have collaborated with my colleagues Dr. Julien Bidot and Pascal Bercher for whom I have great regard, and I wish to extend my warmest thanks to all those who have helped me with my work in the Institute of Artificial Intelligence.

Last but not least, I owe my loving thanks to my wife Heba Elbeh, my son Yousuf and my daughter Salma. They have lost a lot due to my research abroad. Without their encouragement and understanding it would have been impossible for me to finish this work. My special gratitude is due to my parents, brothers, my sister and their families for their loving support. They let me own a happy family in Egypt.

Ulm, Germany, 2011

Mohamed Elkawkagy

DEDICATION

To my parents and my wife for their support and encouragement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Outline

Efficiency of planning systems strongly depends on the size of the search space. We have various classes of AI planners: *state space planners* and *plan space planners*. The search space in *state space planners* consists of a set of nodes and a set of arcs. Each node represents a state of the world, each arc represents a state transition *(action)*, and a plan is obtained by searching in the space of states for action sequences that induce a state transformation that leads to the goal state. The search space in *plan space planners* includes a set of nodes, each node is a partial plan *(i.e. a partial plan consists of a set of partially-instantiated actions and a set of constraints)*.

In recent years, the exploitation of knowledge gained by pre-processing planning domain and/or problem description has proven to be an effective means of reducing planning effort. A lot of pre-processing procedures, like effect relaxation [1], abstractions [2], and landmarks [3], have been proposed for classical state-based planners *(i.e. state space planners)*, where they serve to compute strong search heuristics.

As opposed to this, developing methods to systematically reduce the search effort and increase the

performance of hierarchical planning *(i.e. plan space planners)* by pre-processing the underlying hierarchical structured planning domain and a current problem description in order to increase the performance of a hierarchical planner has not been considered so far.

Furthermore, the process of improving the efficiency of planning by applying a search strategy has been addressed in a number of times in literature [4–9]. However, most of this literature concerns classical state-based planning. Few studies have been performed to analyze the efficiency of search strategies for hierarchical planning.

Therefore, we have motivation to focus on the different aspects of exploiting available pre-processing information during the process of plan generation. We introduce a number of new domain-independent search strategies that base their decisions on pre-processing information and thereby guide the hierarchical planning processes more efficiently towards a solution of a planning problem.

Due to the complexity of real-world domain in recent years, increasing interest has been devoted to the study of the multi-agent based planning approach. In general, the multi-agent planning approach deals with multiple agents having their own goals, and it is often impractical or undesirable to create a plan for all the agents centrally. These agents may be people or companies who simply demand to plan their actions themselves, or refuse to make all necessary information available to someone else. Consequently, such agents want to be able to independently make their own plans without taking the plans of the other agents into consideration. This is not a compelling reason to differentiate between planning and multi-agent planning, yet in many cases dependencies between the tasks of the agents make independent planning impossible. More specifically, if the agents do not take all the dependencies between their plans into account, they might come into conflict when they try to execute their plans. To resolve their dependencies, agents must coordinate their efforts. Therefore, the last contribution in this dissertation will present novel techniques to split the planning problem into a set of sub-problems, and then introduce a new architecture to integrate the pre-processing technique in the context of hierarchical planning approach with a

multi-agent based planning approach.

1.2 Organization

A brief review of the state of the art in Artificial Intelligence Planning shows that these motivations have not been considered so far. In order to achieve the above motivation, this dissertation is broken down into six chapters (See Figure 1.1 which describes the dependency between these chapters):

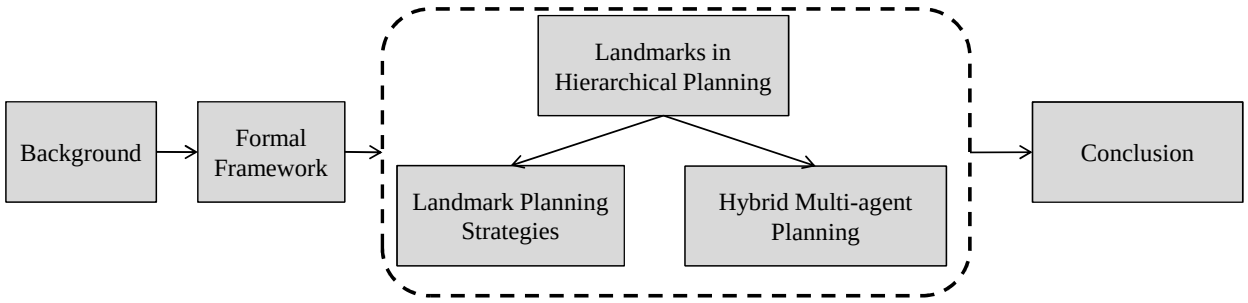


Figure 1.1 Dependencies among the chapters.

- **Background.**

  Through out this chapter we will give an overview of the work which has been done in the area of artificial intelligence planning. We will describe what are the roots of classical artificial intelligence planning, as well as discussing a brief overview of the common planning mechanisms that evolved from these roots.

- **Formal Framework.**

  This chapter presents the underlying formal framework that covers our hierarchical planner. We will focus on the syntax and semantics of planning data structures. We also describe how to formalize the planning problem, the set of solution criteria and how to transform the

planning problem to a solution plan. At the end of this chapter we will present a general algorithm for solving hierarchical planning problems.

- **Landmarks in Hierarchical Planning:**

  Before introducing the concept of landmarks and their extraction in hierarchical planning domains, we will briefly review the concept of landmarks in classical state-based planning. A hierarchical landmark is a new technique aiming at an increase in performance of a hierarchical planner.

  In hierarchical planning, landmarks are mandatory abstract or primitive tasks, i.e. tasks that have to be performed by any solution plan. For an initial task network that states a current planning problem, a pre-processing procedure computes the corresponding landmarks. 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 network, the procedure follows the way down the decomposition hierarchy until no further abstract tasks qualify as landmarks. As for primitive landmarks, a reachability test is accomplished; a failure indicates that the method which introduced the primitive landmark is no longer eligible. This information is propagated back, up the decomposition hierarchy and serves to identify all methods that will never lead to a solution of the current planning problem. Being able to prune useless regions of the search space this way, a hierarchical planner performs significantly better than it does without exploiting the landmark information.

- **Landmark Planning Strategies**

  The information about landmarks can be exploited in two ways: The first way is the reduction of domain models or, more precisely, the transformation of a universal domain model into one that includes problem-specific pruning information. The second is to deduce heuristic guidance from the knowledge about which tasks have to be decomposed on refinement paths that lead towards a solution *(Search Strategy)*. In this chapter, we will focus on latter by

presenting novel domain-independent strategies that exploit landmark information to speed up the planning process. In our approach, the search strategy is divided into two component. The first one is called *modification selection strategy* which examines the set of modification options for the given plan. The second one is called *Plan selection strategy* which navigates the search space of modification plans that are under examination in order to choses the route through the refinement space.

- **Hybrid Multi-agent Planning**

  In this chapter we will introduce a new approach, the so-called *Hybrid Multi-agent Planning (HMAP)*. It integrates a pre-processing technique in the context of hierarchical planning approach *(Hierarchical landmark technique)* with a multi-agent based planning approach. This integration will be done by constructing a set of non-cooperative agents which are executed concurrently. Each one of them performs hierarchical planning with considering landmark-domain model reduction in order to generate its own individual plan. By means of the HMAP approach we introduce two different techniques namely, `Dependent` and `Independent` to break up the planning problem into a set of sub-problems.

  At the end of each chapter we will present experimental results from a set of benchmark problems of the *UM-Translog* and *Satellite* domains, which give evidence for the considerable performance increase gained through our approach.

- **Conclusion**

  Finally, we conclude with general possible extensions of our approach for future developments.

Chapter 2

Background

Planning is considered a central active field of research since the beginning of **A**rtificial **I**ntelligence **(AI)**, not only because of its connection between fundamental issues in AI knowledge representation and computation, but also because of its practical importance. The output of plans in AI is usually viewed as a sequence of actions that are able to change the environment *(current state)* from one state to desired state *(goal state)*. The process of generating a plan is called **planning process**. In general, a planning problem is represented by an initial world state, goal state descriptions as well as a domain model. The latter typically is a knowledge base containing action specifications. In order to find a solution plan for a planning problem, we need a planning system, a so-called ***Planner*** (see Figure 2.1). A planning system searches in the space of states for action sequences that induce a state transformation that reaches the goal state. This is a hard problem because the possible sequences in the search space can be very large and sometimes infinite. If the search space has been exhausted or the allowed maximum CPU time has been reached, no solution is found.

A large variety of approaches have been developed in the past to study the problem of constructing plans. This chapter surveys an overview of different planning approaches in order to provide the context within which the contribution of this dissertation can be evaluated.

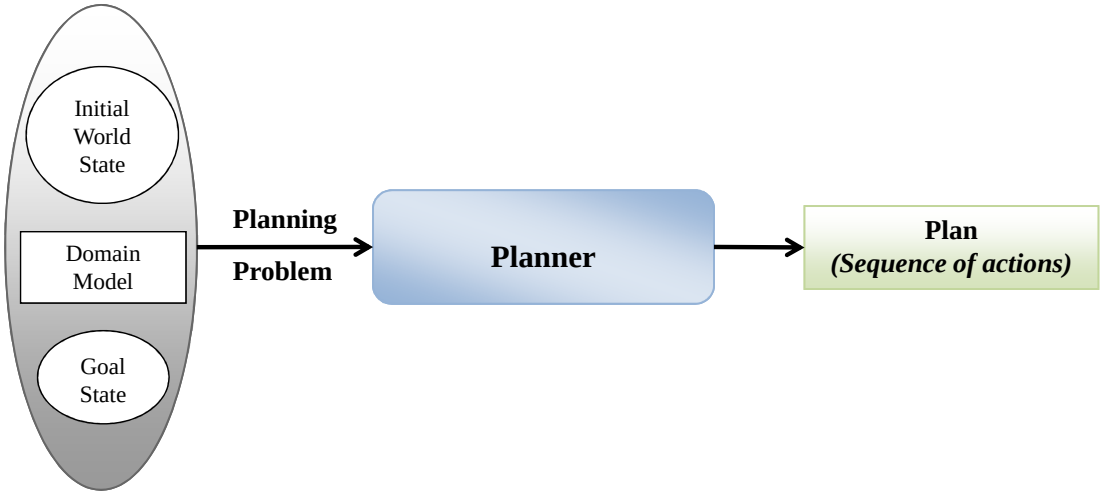


Figure 2.1 Planning process

2.1 AI Planning

The planning contributions of this dissertation are in the area of hierarchical planning. In principle, the approaches of AI planning are divided into the following main categories:

1. Classical state-based planning
2. Partial order planning
3. Hierarchical planning
4. Hybrid planning

We will focus on these categories in the following sections.

2.1.1 Classical State-based Planning

The STRIPS system[1] [10] is the first approach that has been used to study the problem of constructing plans in the classical paradigm. The STRIPS system is a well-known planning framework and

[1] STRIPS $\Longrightarrow$ **ST**anford **R**esearch **I**nstitute **P**roblem **S**olver

the formal representative language that was used for STRIPS is common to most classical state-based planning frameworks. The STRIPS planning uses first order predicate logic language in order to encode the world state. The STRIPS planning framework has been formalized and analyzed by Bernhard Nebel [11].

In the STRIPS paradigm, a domain model formalism contains general information such as types, predicates and operators *(actions)* (See Table 2.2). A predicate is a statement or a relation between different objects in the application domain. It may take on the values true or false depending on its parameters. A state s in STRIPS paradigm is represented by a collection of binary variables so -called *facts*. Facts are obtained from the predicates by instantiating their parameters with constant symbols. Note that the true facts are represented explicitly in the state, while the false facts are unspecified in the state according to the so-called *Closed World Assumption* [12].

An operator is the system's representation of actions that may be executed in the application domain. Each operator o is represented by four arguments: (i) the operator name, (ii) a set of parameters, (iii) the pre-condition *(pre(o))*, and (iv) the post-condition or effect *(eff(o))*. The pre-condition *(pre(o))* is a conjunction of predicates. They must be true in the world state for the action to be applicable. The effect *(eff(o))* consists of two terms: a conjunction of the positive effects *(Add(o))*, and a conjunction of the deleted effects *(Del(o))*. However, effects *(either Add(o) or Del(o))* represent the changes in the world state as a direct result of the operator execution. It is interesting to note that $Del(o) \cap Add(o) = \emptyset$.

The action is applied to the current state by using the transition function **Result**:

$$Result : S \times A \longrightarrow S$$

Where, S is the set of states, and A is the set of instantiated actions.

An instantiated action is obtained from the operator by instantiating all its parameters with constant symbols. If all pre-conditions *pre(a)* of action a are true in the state, then action a is applicable to state s, and the $Result(s, a)$ is defined. The result of applying an action a in the state s will remove

the literals which exist in the negative effect *(Del(a))* from the current state as well as adding the literals in the positive effect *(Add(a))* to generated a new state.

Definition 1 (**Applicable Action**). *The result of applying a STRIPS action a to the state s is defined as the following:*

$$Result(s,a) = \begin{cases} (s \cup (Add(a)))/Del(a) & if\ pre(a) \subset s \\ s & Otherwise \end{cases}$$

As depicted graphically in figure 2.2, the result of applying a sequence of more than one action $\langle a_1,\ a_2,\ \cdots,\ a_n \rangle$ to the state s recursively defined as the following:

$$Result(s, \langle a_1,\ a_2,\ \cdots,\ a_n \rangle) = Result(Result(s, \langle a_1,\ a_2,\ \cdots,\ a_{n-1} \rangle), a_n).$$

The planning problem in classical state-based is represented by describing the known part of the world state, the so-called *initial state*[2], as well as the desired state, the so-called *goal state*. The planner aims to find out the sequence of actions in order to generate the goal state by performing these actions in the initial state. Therefore, in STRIPS formalism the plan is a solution to a given planning problem if the goals of the problem match a subset of the world state immediately after the last action in the plan is executed.

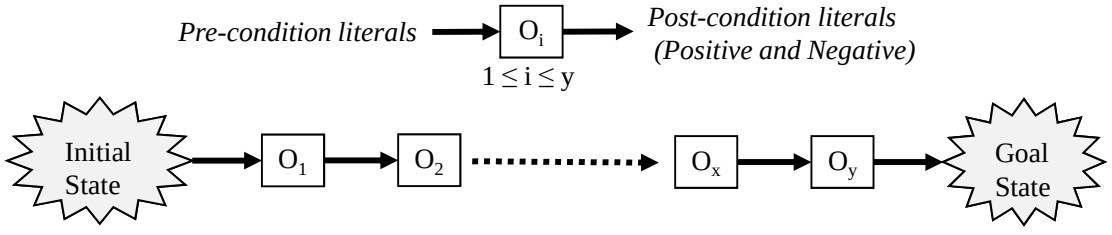


Figure 2.2 Plan in classical state-based planning

[2]An initial state is a set of ground positive atoms which identify what conditions are true initially.

In order to illustrate how the classical planner works, let us consider a simple planning problem as presented in table 2.1 and a simple domain model[3] which we call *SimpleTransportation* as shown in table 2.2. Suppose, we would like to transport a package P_1 from a location L_1 in the initial state to a location L_2 by using truck T_1 which is initially located at location L_1. Suppose there are three actions in the *SimpleTransportation* domain that can change the world state.

Table 2.1 A problem instance for SimpleTransportation domain.

```
(define (problem PackageTransportation)
    (:domain SimpleTransport)
    (:Objects
        P1 - Package L1, L2 - Location T1 - Vehicle
    )
    (:init
        (At_package P1 L1)
        (At_vehicle T1 L1)
    )
    (:goal
        (At_package P1 L2)
    )
)
```

(i) As depicted graphically in figure 2.3, truck T_1 can move from one location L_1 to another location L_2 by applying the action $Move(T_1, L_1, L_2)$. The moving action requires that truck T_1

[3]In all running examples in this dissertation, variable symbols are written with a preceding question mark to distinguish them from constant symbols.

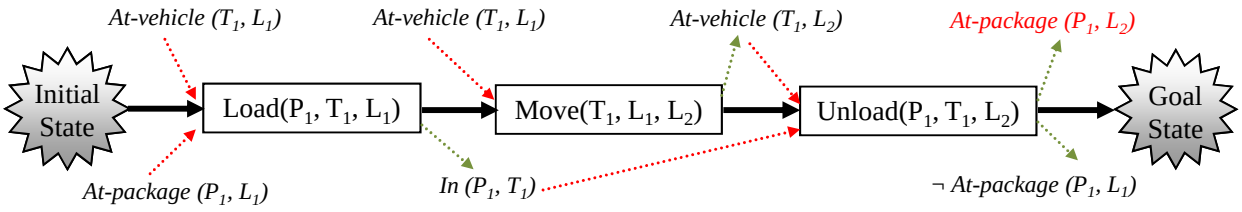


Figure 2.3 Transport package example: Red arrows represent pre-conditions, while Green arrows represent post-conditions

is located at location L_1, and ensures that, in the resulting state, truck T_1 is located at location L_2 and not at location L_1. (ii) The desired package P_1 is loaded into truck T_1 by applying the action $Load(P_1, L_1, T_1)$. The loading action requires that the truck T_1 as well as package P_1 are located at location L_1, and ensures that in the resulting state package P_1 is in the truck T_1 and not at location L_1. (iii) The package P_1 can be removed from the truck by applying the action $Unload(P_1, L_2, T_1)$. The unloading action requires that truck T_1 being at the location L_2 as well as package P_1 being in truck T_1. Hence, it ensures that the new state holds package P_1 at location L_2 and not in the truck T_1.

STRIPS planners proceed in one of two ways: The backward *(regression)* search strategy or the forward *(progression)* search strategy. The backward search strategy starts from a goal state specification, by choosing the suitable action that satisfies the current sub-goal. Applying the selected action will generate new sub-goals, and then the algorithm is called recursively with these new sub-goals. The backward search technique terminates successfully when it reaches an action that is performed directly in the initial state. The forward search strategy updates the initial state by applying the suitable actions and producing a new state each time until the goal state is established. In general, the STRIPS algorithm assumes that the sub-goals in the intermediate states are independent from each other and can be performed in any order. Therefore, the STRIPS algorithm produces a totally ordered *(linear)* plan. A lot of approaches have been introduced to solve the dependency between sub-goals such as INTERPLAN [13] which tries to analyze the sub-goals in

Table 2.2 A simple domain model in STRIPS.

```
(define (domain SimpleTransportation)
    (:types Package Location Vehicle)
    (:predicates
        (At_Package ?P - Package ?L - Location)
        (At_Vehicle ?T - Vehicle ?L - Location)
        (In ?P - Package ?T - Vehicle)
    )
    (:action Move
        :parameters (?t - Vehicle ?loc - Location ?L - Location)
        :pre-condition (and (At_vehicle ?t ?loc))
        :effect (and (not (At_vehicle ?t ?loc)) (At_vehicle ?t ?L))
    )
    (:action Load
        :parameters (?P - Package ?L - Location ?t - Vehicle)
        :pre-condition (and (At_package ?P ?L) (At_vehicle ?t ?L))
        :effect (and (In ?P ?t) (not (At_package ?P ?L)))
    )
    (:action Unload
        :parameters (?P - Package ?L - Location ?t - Vehicle)
        :pre-condition (and (In ?P ?t) (At_vehicle ?t ?L))
        :effect (and (not(In ?P ?t)) (At_package ?P ?L))
    )
)
```

the intermediate states and then find out a sequence of sub-goals which solve the interaction between them. Afterwards, a new direction formulates the planning process as a search in the space or constructs partial plans [14–17].

2.1.2 Partial Order Plan

The STRIPS planner preserves a total order list of all actions in its plan a so-called *total-order-planner* or *linear planner*. As opposed to this, a partial order planner *(non-linear planner)* preserves temporal constraints between pairs of actions. These temporal constraints means that an action a_j comes after an action a_i, but not necessarily immediately after it [18]. Partial order planners are often referred to as *Least Commitment Planners*[4]. They involve deferring decisions about the temporal orderings and variable bindings until they are required to resolve the conflicts in the partial plan. It allows the planner to be flexible about parts of its plan until it has enough information to work out the best possible course of action.

A partial order plan can be represented as a directed acyclic graph $P_{pop} = \langle \gamma, \zeta \rangle$. The set of vertices γ represents plan steps *(an instance of one of the action)* in the plan and arcs ζ represent a set of temporal constraints between plan steps. It is furthermore important to mention that, there are two special plan steps in the partial plan: the initial plan step that does not have pre-conditions and considers the initial state as post-conditions, and the goal plan step that has goal literals as pre-conditions and does not have post-conditions.

Corkill [19] inspired a new data structure, the so-called *Procedural net* that formulates a plan as a partial ordering of actions. He introduced a new search technique in his NOAH system[5], the search in *plan space* instead of the search in *state space* as in STRIPS planner.

As depicted in Figure 2.4 (b), the search space of a plan space planner is a set of partial plans and

[4]The principle of least commitment has been used in a lot of fields of AI, including computer vision, planning and theorem proving.

[5]NOAH $\Longrightarrow$ **N**ets **O**f **A**ction **H**ierarchies

the plan itself is handled by adding new plan steps or constraints in order to generate new plans in the search space. As opposed to this, a state space planner (See figure 2.4 (a)) searches through the space of possible states of the world. This means, state space planners search for a path that solves the problem by using forward search or backward search techniques.

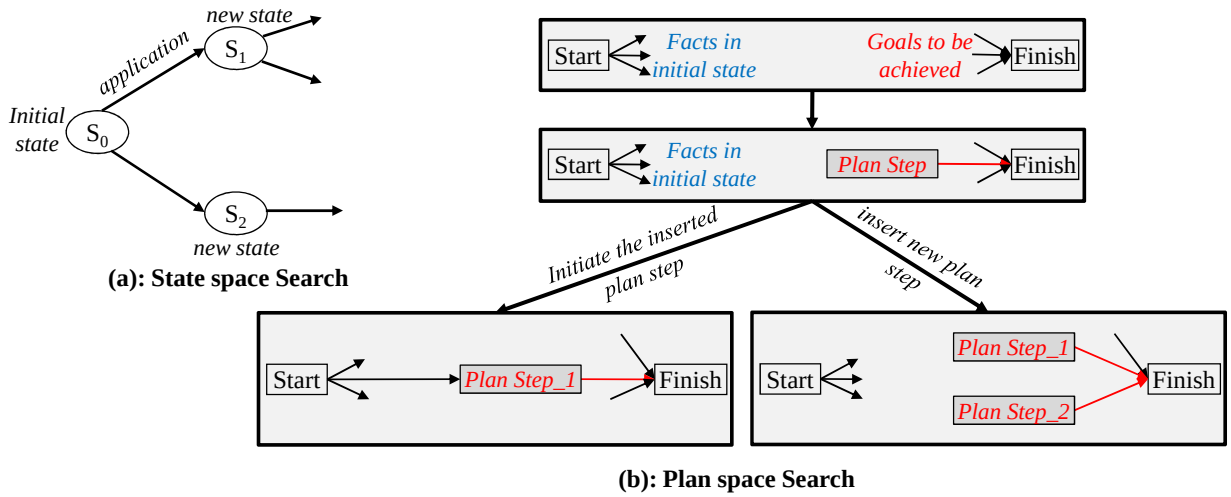


Figure 2.4 State space search versus plan space search

Afterwards, Socerdoti [20] proposed new constraints between plan steps in partial order planing, the *causal link constraints*. A causal link constraint has the form $\langle s_i, \varphi, s_j \rangle$. It specifies a pair of plan steps s_i and s_j as well as a literal φ, where the literal φ is a pre-condition of the second plan step s_j *(consumer plan step)* and at the same time it is a post-condition of the first plan step s_i *(producer plan step)*. Generally, causal link constraints are formulated to preserve the literals that are accomplished so far. A planner can benefit from causal link constraints in two different ways: First, it can introduce an **Establishment** that handles an unsolved pre-condition *(open pre-condition)* p_j of plan step s_j. This can be done by building a new causal link c_j from a suitable plan step s_i to produce the required pre-condition p_j for plan step s_j $(c_j : \langle s_i, p_j, s_j \rangle)$, or by inserting a new plan step that carries the required pre-condition in its post-condition. Second, **clobbering(Threat Removal)** that removes the causal link threat. The causal link $c_j : \langle s_i, p_j, s_j \rangle$

is threatened by another plan step s_k when it deletes the protected literal p_j of the causal link c_j, and the plan step s_k is ordered between the two plan steps of the causal link. The threat is solved by propagating the order constraint between the threat task *(i.e., plan step s_k)* and the causal link components, that means, the order constraint between the consumer plan step and the threat task $(s_j \prec s_k)$ or between the threat task and the producer plan step $(s_k \prec s_i)$ is propagated.

One of the earliest planners that lifted STRIPS *"total order planners"* and maintains a partial order on plan steps was the NONLIN planner [18]. It derived from NOAH planner in order to complete the step towards plan space planning. It introduced the partial plan data structure which includes plan steps, ordering constraints on the plan steps, and parameter bindings constraints. In addition, it applied a general backtracking schema over the plan generation process to re-construct alternative ways in case a particular choice lead to a dead end. After that, Chapman [15] demonstrates in his planner TWEAK an alternative way, the so-called *model-truth-criterion*, for determining whether a partial plan can achieve a given pre-condition at a given step. However, it uses the *model-truth-criterion* in order to generate a plan by incrementally adding new plan steps and adding or modifying constraints. The produced plan will continue to achieve the next subgoals and so on. When all goals of the plan are achieved, the given planning problem is solved. TWEAK depends on the result of the model truth criterion in order to choose the appropriate plan step s_j. The result of model truth criterion is true if and only if: (1) the pre-condition p_j of the current plan step s_j has been achieved by another plan step s_i, (2) the plan step s_i is performed before the plan step s_j $(s_i \prec s_j)$, and (3) there is no other plan step s_k preceding the plan step s_j that might remove pre-condition p_j.

The most popular partial order causal link planners [6] are SNLP [7] [21] and UCPOP [8] [17].

SNLP is considered a further development of Tate's planner *(NONLIN)* [18,20]. The technical dif-

[6] *All planners that apply causal link paradigm called **P**artial **O**rder **C**ausal **L**ink (POCL)*

[7] SNLP $\Longrightarrow$ **S**ystematic **N**on-**L**inear **P**lanner

[8] UCPOP $\Longrightarrow$ **U**niversal **C**onditional **P**artial **O**rder **P**lanner

ference between them lies in its threat formulation. NONLIN considers the plan step s_k a threat to a causal link $\langle s_i, \varphi, s_j \rangle$ only if s_k deletes the protected literal φ. SNLP, however, is more restricted. It considers s_k to be a threat when it deletes or adds the protected literal φ. This restriction provides a more systematic algorithm, because it does not allow to duplicate a plan in the plan space.

The UCPOP planner discusses the foundations of partial order planning. It is the first non-linear planner for which soundness and completeness were implemented. In addition, UCPOP's domain model is formulated in ADL [22]. The theoretical foundation of total order planning with ADL[9] was inspired by the works of Pednault [23, 24]. Afterwards this foundation was developed by Pednault [22], in order to handle the partial order plans. The UCPOP planner makes ample use of some of the characteristics of ADL, such as representing actions with conditional effects and universal quantification.

Utilizing universal quantification within actions helps preserving the truth value of specific literals after performing the corresponding action. For example, when moving a truck from the current location to a new one, the literals that describe the contents of this truck will be preserved without change.

The conditional effects provide more effects for a particular action in case the given condition is achieved. A lot of planners have been established to extend UCPOP. Most of them focused on implementing efficient planning strategies such as the VHPOP[10] system [25]. It extends the capabilities of POCL planners by also considering durative actions *(i.e., temporally extended actions)* [26]. In addition, it competed well with other heuristic state space planners at the 3rd **I**nternational **P**lanning **C**ompetition (IPC3). For a detailed discussion on search strategies, we refer the reader to chapter 5 of landmark planning strategies.

The GRAPHPLAN planner is considered a new generation of the classical planning algorithm [27]. The GRAPHPLAN planner is a general purpose planner for STRIPS-style domains. GRAPH-

[9] ADL $\Longrightarrow$ **A**ction **D**escription **L**anguage

[10] VHPOP $\Longrightarrow$ **V**ersatile **H**euristic **P**artial **O**rder **P**lanner

PLAN planner is considered one of the most important developments in AI planning because it is simple and uses a relaxed representation of action sequences and reachable world states. In a lot of cases GRAPHPLAN generates plans faster than previous planners such as SNLP and UCPOP. GRAPHPLAN consists of two different phases: (1) Constructing a planning graph and (2) extracting a solution plan.

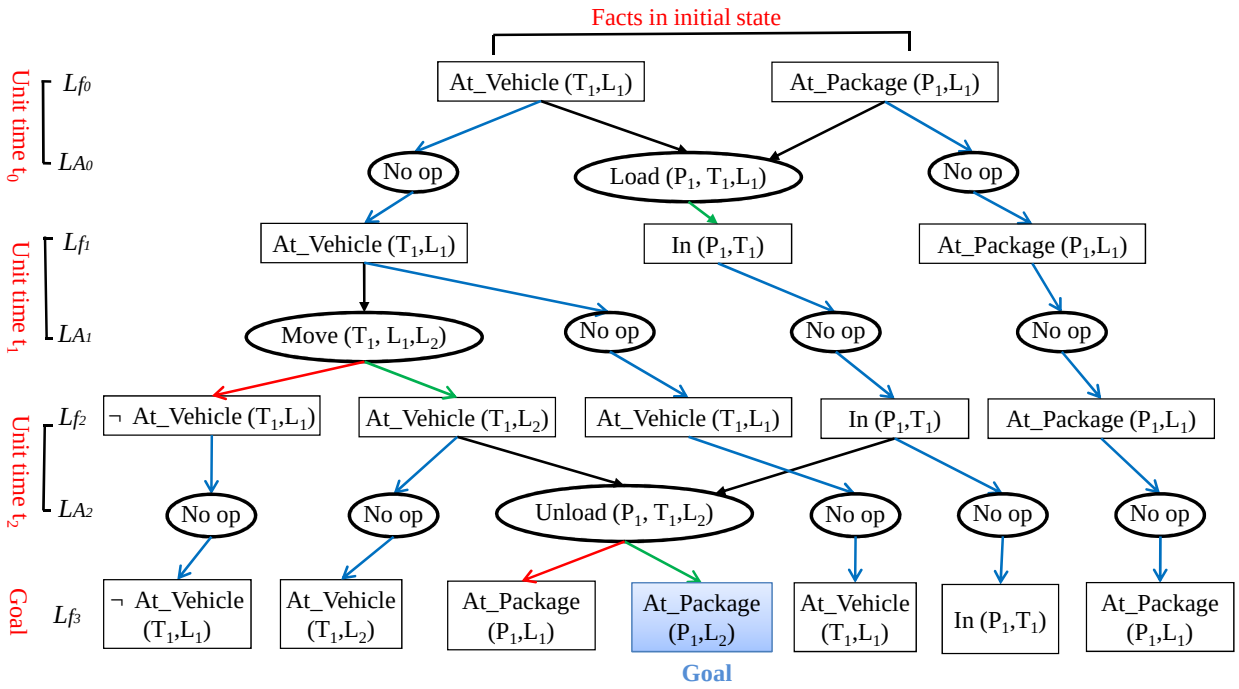
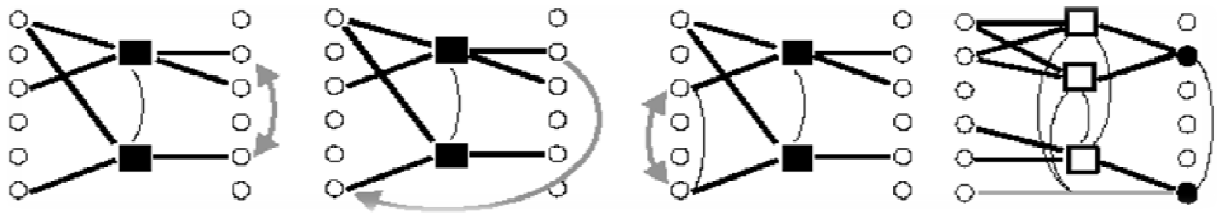


Figure 2.5 Planning graph for a simple transport package problem: Blue lines represent no-operation. Green lines represent adding effects while red lines represent deletion of effects.

In the first phase a planning graph is constructed by adding different types of nodes and edges to the graph. The levels of a planning graph consist of interleaving layers of facts that represent possible future states *(fact nodes)* and actions which contain possible choices for actions *(action nodes)*. For example, the action nodes in action level i are connected by fact nodes in fact level i *(which represent the pre-conditions of the intended action)* and by a set of facts in the fact nodes at

level $i + 1$ *(which represent the intended action post-condition)*.

In general, the GRAPHPLAN algorithm depends on the *Forward chaining* function in order to construct a planning graph. It starts by considering all facts in the initial state which represent first fact level L_{f_0} and then applies those actions for which all pre-conditions exist in the fact level L_{f_0} to construct the first action level A_{f_0}. Afterwards, the next fact level L_{f_1} is generated by adding all the literals which exist in the post-condition of the applied actions. This is done until a fact level has been reached that contains all facts of the goal state. As depicted graphically in figure 2.5, each planning graph iteration extends the graph by one unit of time or two levels.



(a) : Inconsistent effect **(b) : Interference** **(c) : Competing Needs** **(d) : Inconsistent support**

Figure 2.6 Mutual exclusion *(mutex)*: Circles represent facts, boxes represent actions and gray arrows represent mutexes.

Actually, a planning graph also includes a number of conflicts between nodes at the same level, the so-called *mutual exclusion (mutexes)*. Two actions are mutex if one of three conditions hold: (1) The post-condition of one action is deleted by another action's post-condition as depicted in figure 2.6(a), (2) Figure 2.6(b) shows another type of mutex between actions so-called *(Interference)*, where the post-condition of one action deletes the pre-condition of another action, (3)The mutex which called *Competing needs* represents actions that have mutex pre-conditions at the previous fact level(see Figure 2.6(c)). Two facts are mutex if all the actions that could produce these facts are mutex *i.e. actions in the previous level which achieve these facts are mutex and so-called inconsistent support* (see Figure 2.6(d)).

Once the planning graph has been constructed, the solution plan extraction *(second phase)* is performed. The solution plan extraction phase depends on the regression technique in order to find a solution. It starts with a set of facts at the last level L_{f_n} of the planning graph *(goal facts)*, and then attempts to find a set of actions at level $L_{A_{n-1}}$ that have these goals as add effects. After that, at level $L_{f_{n-1}}$, the pre-conditions of the specified actions are considered as new goals and a new set of actions has to be found and so forth until the first fact level L_{f_0} has been reached. Otherwise, GRAPHPLAN tries to find a different set of actions until it succeeds or proves that the given problem is not solvable.

The common problems for all planning algorithms that have been discussed are the complexity of the planning problems and they do not have high flexibility to express more actions and states.

2.1.3 Hierarchical Planning

The common approach to improve the efficiency of planning is to use *Hierarchical planning*. In general, hierarchical planning is categorized into two approaches, based on the kind of abstraction: **state abstraction** and **action abstraction**.

In hierarchical planning, the state abstraction is a powerful method for reducing the planning search space from exponential to linear time under specific conditions such as a hierarchy that satisfies DRP[11]. The DRP condition guarantees that no backtracking occurs between abstraction levels. DRP is formalized in ABSTRIPS[12]-style hierarchies [28].

The ABSTRIPS planner is the earliest system that deals with state abstraction. It is built on top of the STRIPS planning system. The abstraction hierarchy for the problem space in ABSTRIPS is constructed by assigning a number of so-called *criticality values* to the pre-conditions of each operator. Through these critical values, ABSTRIPS generates a plan by starting with the highest

[11] DRP $\Longrightarrow$ ***D**ownward **R**efinement **P**roperty*

[12] ABSTRIPS $\Longrightarrow$ ***A**bstraction-**B**ased **STRIPS***

critical value. Afterwards, this abstract plan is refined by considering now the pre-conditions that have the next critical value and so forth until the lowest level of criticality is reached.

One of the most important hierarchical planners that generates abstraction for solving hierarchical problems automatically and has better abstraction than ABSTRIPS is ALPIN [29]. It relies on the *ordered monotonicity property* which ensures that the structure of the abstract plan will be preserved while the plan is refined *i.e. all the refinement plans of the abstract plan leave the literals that have already been achieved in the abstract space without any change*.

Although those planners that provide hierarchical planning by state space abstraction reduce the search space successfully, they lack in semantics because they are defined as a search algorithm and not modeled as a planning domain model.

As opposed to this, the second category of hierarchical planning depends on the hierarchies of the abstraction of actions, it is also known as **H**ierarchical **T**ask **N**etworks (HTN)[13]. In general, action abstraction systems organize the description of the actions in a hierarchical form. Specifically, the more abstract or complex action is placed on the top of hierarchy *(high or abstract level)*, and the more specific one is placed on the lower level in the hierarchy and so forth, until the lowest level is reached, which is called the *primitive level*. Therefore, there are two types of tasks: primitive tasks that can be performed directly like operators in the STRIPS paradigm, and abstract tasks that must be decomposed into smaller sub-tasks *(either abstract or primitive)* during the planning process. As depicted in figure 2.7 the *Flying, Driving and Sailing* tasks are organized as the sub-tasks of the abstract task *Traveling*, and the *Buying-ticket* task is arranged as a part of a higher abstract task *Traveling* or sub-task of the *Flying* task. Now, a more general task subsumes more specific tasks. It is interesting to note that the tasks on the primitive level should be executed in a specific order to achieve the purpose of an abstract task. For example, the *Flying* task has *Buying-ticket, Go-to-the-security-office* and *Boarding* tasks in its decomposition. Then, the *Buying-ticket* task

[13]HTN is common the approach of *"planning by action abstraction"*

should be preceded by *Go-to-security-office* as well as performing the *Boarding* task after the *Go-to-security-office* task.

All information about the task hierarchy and how to implement these tasks in the plan are organized in the domain model. Besides the set of tasks in the hierarchical planning domain model, it also includes a data structure so-called *methods*. Each abstract task may have more than one applicable method, so the relevant pre-conditions and effects are not always known in advance. Each method specifies a pre-defined abstract solution or implementation of the corresponding abstract task. It is important to notice that, the conflicts which are introduced during the decomposition process are resolved as they are in *Least Commitment Planning* by adding temporal or variable constraints.

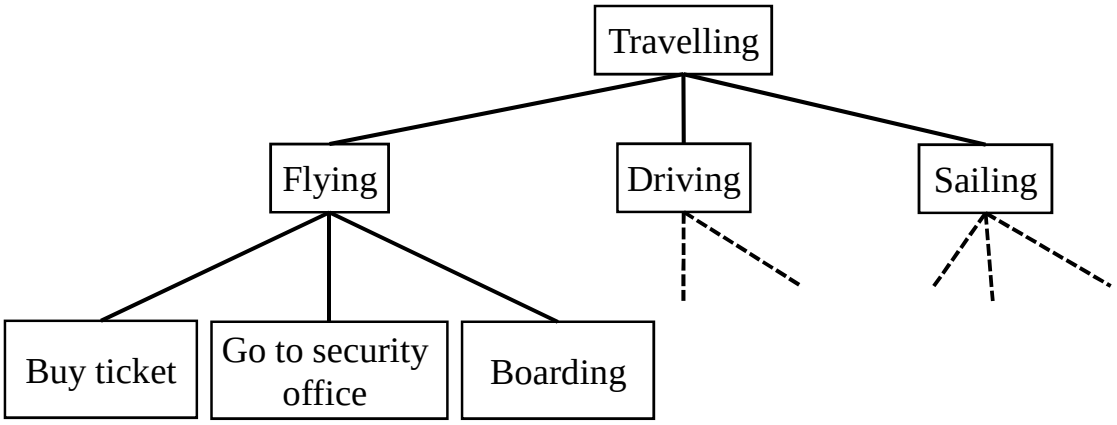


Figure 2.7 Action hierarchy

Opposite to the STRIPS planning system which defines a planning problem as an initial state, a goal state and a set of actions that achieve a given goal state, HTN planning defines a planning problem as an initial state and an initial plan. The initial plan is a non-empty plan containing a set of tasks *(abstract or primitive tasks)* that need to be performed. The solution plan is found by incrementally decomposing the abstract tasks in the initial plan until a primitive plan that has only primitive tasks has been reached, that is executable in the initial state, and that has consistent constraint sets. The decomposition process is called refinement or expansion process. The de-

composition process works by replacing the abstract task by a set of less abstract tasks. Attention should be paid to the fact that, each task may have a set of alternative expansions (see Figure 2.8).

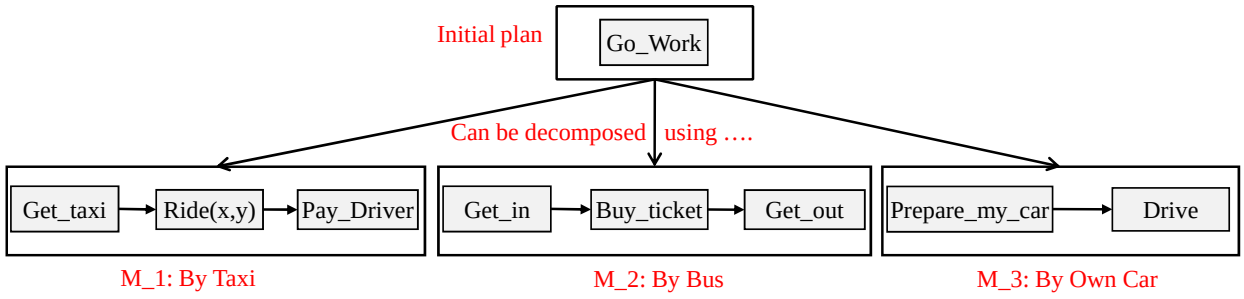


Figure 2.8 Planning by action abstraction: Alternative expansions of abstract task

One of the most commonly known HTN planners is O-plan [30]. O-plan is a domain independent general planning framework. It follows NONLIN [16] in using a tightly constrained method to generate plans that compose the search space. Therefore, O-plan represents abstract actions by introducing *condition types* in the abstract expansion schemata instead of propagating pre-conditions and post-conditions in the abstract task [31]. These condition types are similar to causal link constraints. They describe the relationships between different actions in the plan. These conditions cause difficulties for the design of O-plan domain models because the domain designer has to satisfy the all conditions on actions.

Wilkins et al. [32] introduced an HTN-system called SIPE [14]. This system places a restriction on the possible ways to order actions *i.e. for a given two sub-plans that are unordered with respect to each other, SIPE orders them by putting one sub-plan before or after the other.*

Afterwards, SIPE was further developed into a practical HTN planning system, called SIPE-2 [33]. To achieve the given goals in diverse problem domains, SIPE-2 provides a domain independent formalism for describing actions, and utilizes knowledge encoded in these actions. This

[14] SIPE $\Longrightarrow$ **S**ystem for **I**nteractive **P**lanning and **E**xecution

is combined with a heuristic search that handles the combinatorics of the problem. In addition, SIPE-2 can reason about resources. It can post and use constraints as well as employ a deductive causal theory to represent and reason about different world states. SIPE-2 has been applied to a lot of domains such as military operation [34] and manufacturing environment [35].

Erol was the first to present a formal syntax and semantics for HTN planning [36]. He introduced a hierarchical planner, the *UMCP planner* [15] which depends on the task decomposition [37]. The UMCP planner represents world state and primitive tasks in a similar way as the STRIPS formalism, whereas the goal is represented by the complex tasks. The set of tasks in the plan are connected by a task network which is used to represent plans and sub-plans. The task network is a tuple $\langle A, T, B \rangle$ where A is a set of tasks, either primitive or complex, T is a set of temporal constraints on the members of A, and B denotes a set of variable binding constraints. UMCP works by recursively expanding each complex task by non-deterministically choosing a method until a primitive plan is produced *i.e. all tasks in the plan are primitive tasks*. Afterwards, it starts to solve conflicts between tasks and extract a solution plan.

Nau et al. introduced not only one of the most important HTN system, but also proved that it is sound and complete [38]. Their planner is called SHOP [16]. SHOP reduces the complexity of reasoning by generating a plan for tasks in the same order in which they will be performed. It enables to build a complete world state at each iteration of the planning process. This is done by applying a forward-chaining search algorithm that starts by selecting the first abstract task according to the step ordering. Then it chooses the suitable method that decomposes the selected task. Note that in the SHOP system, each method has a pre-condition that has to hold in the current state before applying the corresponding method, while primitive tasks do not have pre-conditions. SHOP follows the *if-then-else* paradigm to select a method. Consequently, the appropriate method is checked and the first method for which the respective if-statement evaluates to *"true"* is selected for expanding

[15] UMCP $\Longrightarrow$ **U**niverasal **M**ethod **C**omposition **P**lanner

[16] SHOP $\Longrightarrow$ **S**imple **H**ierarchical **O**rdered **P**lanner

the task.

The process of creating a domain model in SHOP requires a greater effort than what is required for classical state-based planners, because the domain model of SHOP propagates total order among sub-tasks in the decomposable method. Of course this renders it impossible to interleave sub-tasks of methods. For this reason, SHOP has been improved to allow each method to decompose into partially ordered sub-tasks *i.e. in a produced plan it is allowed to interleave sub-tasks from different methods*(SHOP-2 [39])

The crucial difference between classical state-based planning and HTN planning is the solution criterion. Whereas the goal in classical state-based planning is to achieve a desired property, no matter which actions have to be used to accomplish this, the goal in HTN planning is to find a plan that is a valid decomposition for the initial abstract task, such that the resulting plan only contains primitive tasks.

2.1.4 Hybrid planning

The difficulty of solving problems in complex real-world application domains, such as emergency evacuation, crisis management [40, 41], and transportation/logistics problems [42] led to the appearance of a new planning paradigm, so-called *hybrid planning*.

In general, hybrid planning is a combination of hierarchical task network with classical state-based planning approaches, each having been studied separately. This means that the produced system has advantages of both approaches *i.e. it has good modeling and efficient search techniques*.

There are a very few works that discuss hybrid systems, one of them is DPOCL [17] [43]. DPOCL built on top of the SNLP [21] algorithm to handle partial action decompositions. DPOCL represents each action by two parts: First, an *action schema* is a tuple $\langle A, V, P, E, B \rangle$, where A is an

[17] DPOCL $\Longrightarrow$ **D**ecomposable **P**artial **O**rder **C**ausal **L**ink

action type *(primitive or composite*[18]*)*, V is a set of variables, P and E represent the pre-conditions and effects of the current action respectively, and B is the set of variable bindings for the variables in V. Second, each action has a set of decomposition schemata which represent alternative ways to decompose the complex action into more primitive actions. Intuitively, the set of decomposition schemata of primitive tasks are empty, because they are performed directly without any decomposition. The process of generating a plan in DPOCL includes deciding what action should be used to achieve a sub-goal as well as solving interactions between plan steps(tasks).

Kambhampati et al. [44] introduced another hybrid system integrating hierarchical task network planning and refinement planning. To this end, the refinement planning framework for classical state-based planning is extended to include complex actions and the decomposition schemata of these actions as part of the domain specification [45]. In the process of plan generation, the plan is refined by selecting an open pre-condition from the current plan and then closing it by selecting an appropriate task that generates this pre-condition. If the selected task does not close the current condition explicitly then this condition is handled again via a *phantom establishment* process which converts this open pre-condition into a constraint that solved later during plan development.

Recently, Schattenberg et al. [40, 46] introduced a new hybrid planning system, the so-called PANDA system[19]. As mentioned before, POCL planning is a technique used to solve partial order planning problems. The objective is to achieve the goal state by applying actions in a correct order starting in a given initial state. In addition, the POCL technique explicitly shows causal dependencies between actions. The key feature of HTN planning is action abstraction which allows representing abstract tasks as well as pre-defined abstract solutions for these tasks. So, HTN planning reflects and employs abstraction hierarchies that are inherited in many domains. Therefore, the PANDA environment combines the key features of POCL and HTN planning techniques. In contrast to the previously discussed techniques, PANDA requires pre- and post-conditions for ab-

[18] Composite action is a complex or abstract task

[19] PANDA $\Longrightarrow$ **P**lanning and **A**cting in a **N**etwork **D**ecomposition **A**rchitecture

stract tasks. It utilizes a mapping between abstract tasks and the alternative ways of implementing these abstract tasks by way of so-called *Decomposition methods*. PANDA defines a planning problem through a domain model as well as an initial partial order plan, and a tuple of initial state and goal state. The initial plan contains the set of the most abstract tasks that are required as well as two artificial tasks t_{init} and t_{goal} that represent the initial and the goal states respectively. The task t_{init} considers the initial state as post-condition and the task t_{goal} considers the goal state as pre-condition. The initial partial plan is stepwise refined by adding new tasks and constraints - causal links, ordering and variable constraints. As depicted in figure 2.9, the abstract task *(Task-A)* is refined by replacing it through an appropriate partial plan as specified in the selected decomposition method [7]. One of the most important features of the PANDA system is that it is implemented in separate modules. So that the implemented system can be employed as a platform to implement and evaluate various planning methods such as purely HTN planning as well as evaluating a lot of different search strategies [8].

In contrast to other systems, which *implicitly* define their control search strategy by their search procedure, the PANDA planning environment *explicitly* defines the search strategy. A search strategy in the PANDA planning environment is a combination of the used modification and plan selection functions[20]. Let us take a look at a simple example strategy for clarification: To perform a depth first strategy, the plan selection strategy has to be the identity (i.e., $f^{PlanSel}(\overline{P}) = \overline{P}$ for any plan sequence $\overline{P}$), whereas the modification selection strategy f^{ModSel} can be arbitrarily chosen (but decides, which branches to visit first). Thus, the plan selection strategy is used to prioritize the plans; several strategies can be concatenated for tie-braking. The plan selection strategy uses its input sequence for tie-braking as well: If two plans are still invariant after applying the plan selection function, the order given in the input is used. A number of different plan and modification selection strategies have been described and evaluated in the work of Schattenberg [7, 8, 46].

[20]For more details refer to chapter 5

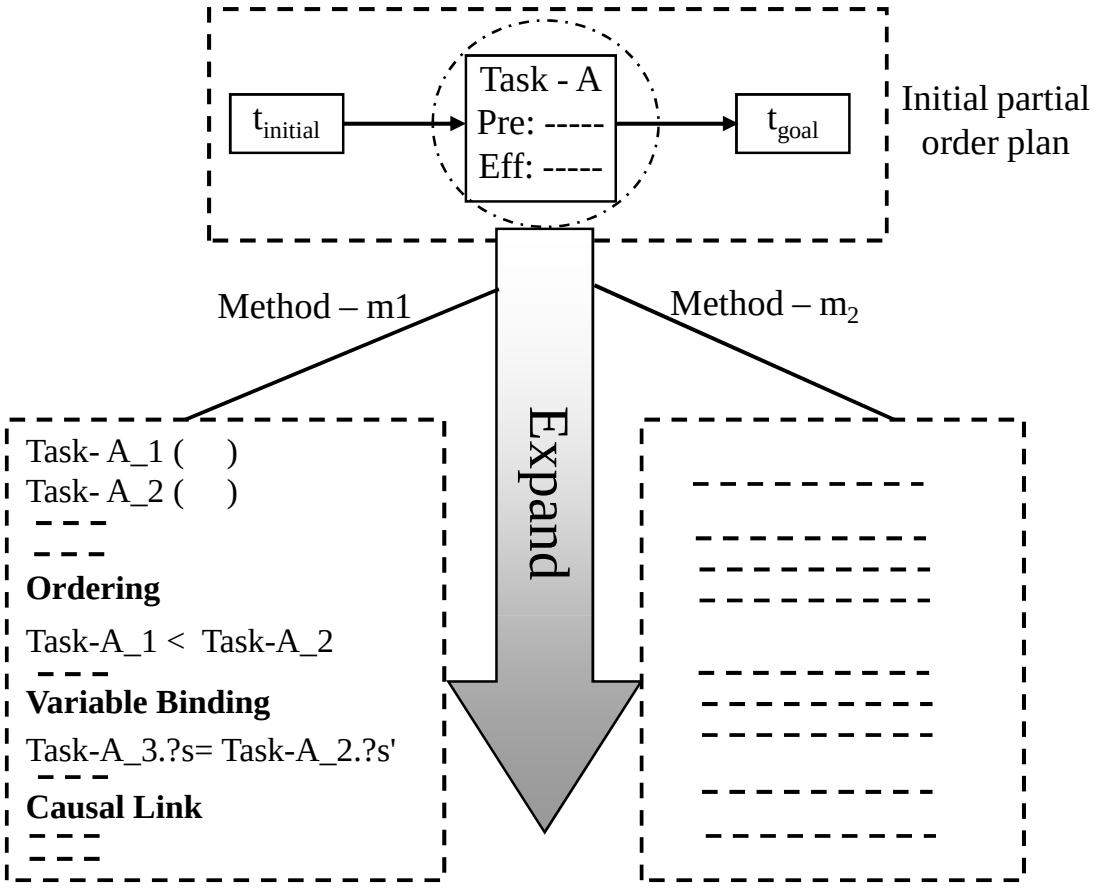


Figure 2.9 Expand abstract task in PANDA

Therefore, due to the great properties of the PANDA planning environment, our planning framework will be an adaptation of the hybrid formalization of PANDA[21].

[21] For more detail refer to chapter 3

Chapter 3

Formal Framework

This chapter presents our underlying formal framework. Hierarchical Task Network (HTN) planning is based on the concepts of tasks and methods [37]. Rather than planning to achieve a set of goals such as classical state-based planning, hierarchical planning performs a set of tasks. In hierarchical planning tasks subsume actions and goals. In general, there are two types of tasks in HTN planning: abstract tasks which represent compound activities such as making a business trip or transporting certain goods to a specific location and primitive tasks which correspond to classical state-based planning operators, which can be executed directly *(i.e., they do not need any decomposition)*. Hierarchical domain models hold a number of *methods* for each abstract task. Each method provides a task network, also called partial plan, that *implements* the abstract task and thus specifies a pre-defined *(abstract)* solution of the corresponding abstract task. Consequently, an HTN planning problem is a (initial) task network.

3.1 Basic HTN Planning Algorithm

An HTN planning problem is solved by repeatedly substituting the abstract tasks in the initial task network with their implementations until the network contains only primitive tasks and is consis-

tent w.r.t. their ordering and causal structure.

Algorithm 1: The Basic HTN Planning Algorithm

Input : $P = \langle D, N \rangle$: Planning Problem

Output: `Solution Plan` P `or Failure.`

```
1  begin
2      if (All tasks in P are primitive tasks) then
3          Resolve the conflicts between the primitive tasks in P
4          return Solution Plan
5          if (conflicts unresolvable) then
6              return Failure
7      else
8          Select an abstract task t in P.
9          Select an expansion for task t.
10         Replace t with the expansion sub-tasks.
11         Detect conflicts between tasks in P.
12         Suggest the possible ways to resolve the detected conflicts.
13         Apply on of the above suggestions (in line 12) on P.
14         return BasicHTN(P)
15 end
```

Algorithm 1 introduces the basic HTN planning procedure which is considered to be a core of all HTN planning systems. The input of the basic HTN planning algorithm is a planning problem $P = \langle D, N \rangle$, where D is a domain model and N is a task network. It returns a solution plan if the plan P is a primitive plan and consistent *(i.e., all tasks in the plan are primitive, and all the conflicts between them are resolved)* (lines 2 to 4). If the primitive plan P is inconsistent

(i.e., the conflicts between tasks can not be resolved), then a failure is returned (lines 5 and 6). Otherwise, in lines 8 to 9, an abstract task t in P and the suitable decomposition method that matches the task t are selected. After that, the selected decomposition method is applied to the current plan P by replacing the specified abstract task with the set of sub-tasks in the selected method(line 10). Worth mentioning is that adding new tasks in the plan P may produce conflicts between tasks. These conflicts are addressed in line 11. All the possible ways to handle these conflicts are computed in line 12. In general, there is more than one way to decompose an abstract task and more than one way to resolve conflicts *(e.g., task interactions)* in a plan. Then, in line 13, we select one of the suggested ways *(in the previous line)* in order to solve the conflicts in the plan P. Finally, in line 14, the basic HTN algorithm is called recursively with the refined plan *(i.e., plan after expansion)* in order to make a new expansion.

Our approach relies on a *hybrid* planning formalization [40], which combines HTN planning with concepts of partial-order-causal-link (POCL) planning. The resulting systems integrate task decomposition with explicit causal reasoning. Therefore, they are not only able to use pre-defined standard solutions as it is the case in pure HTN planning, but also to develop (parts of) a plan from scratch or to modify a default solution in cases where the initial state deviates from the presumed standard. It is this flexibility that makes hybrid planning particularly well suited for real-world applications [41, 47]. Note that in our hybrid planning framework one can specify a goal state as well as an initial plan containing one or more abstract tasks that need to be performed. Since our contribution in this thesis relies on pure hierarchical planning, the goal state is omitted.

In this chapter we will cover the underlying logical language of our framework, tasks, domain model entities, plans, planning problems and solutions as well as illustrate how to refine the current plan in order to generate the final solution plan.

3.2 Logical language

Our framework is adapted from the formalization of PANDA hybrid planning [40]. It relies on an sorted first-order logic [48]. The syntax of our framework is given by the *logical language* $\mathbb{L} = \langle Z, \prec, R, Const, V, O, T, E, L \rangle$.

In sorted logics, all variables and constants are of some sort $z \in Z$. As an example some sorts of the *UM-Translog domain* are shown in table 3.1. In addition, order-sorted logics impose a hierarchy on sorts which allows for more adequate and concise formalizations. This hierarchy on the sort symbols in Z is represented by defining the relation $\prec$. Sorts in Z are super or sub-sorts for each other. Figure 3.1 shows part of the sort hierarchy of *UM-Translog* domain model, where e.g. the sort *Mail* is a super sort for the sub-sorts *Mail_Vehicle* and *Mail_Package*.

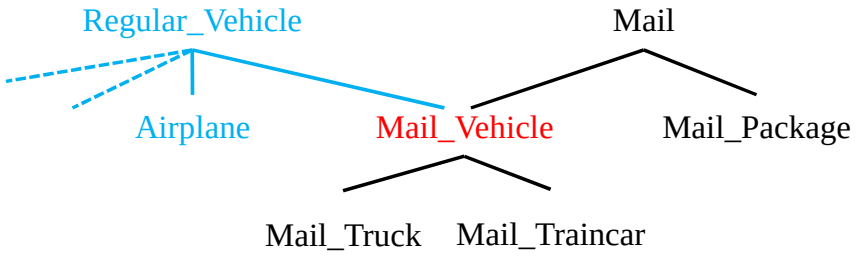


Figure 3.1 Part of the sort hierarchy of our UM-Translog domain model

The *relation symbols* R are used to represent properties of objects in the real world. The relation R is a Z^*-indexed family of finite disjoint sets of *relation symbols*. Each *relation* R can be either rigid or flexible. In general, the term rigid is applied to those literals or relations that cannot be modified or added during the planning process. While flexible terms are those literals that can be added or modified during the planning process. $Const$ is Z-indexed family of finite disjoint sets of *constant symbols* which represent objects in the real world. For example, suppose we want to express in our model that a specific transport package is associated with a specific source and destination **location**, we would use a constant pck_1 of sort **package** to represent the

Table 3.1 A subset of sort elements of the logical language $\mathbb{L}$, which are used to model the functionality of transportation package.

Sorts Z

Name	Description
Package	Includes a subset of the sorts of package types such as Liquid and Valuable package.
Route	Includes different kinds of routes such as Air_Route and $Rail\_Route$.
Customer_Location	A type of location
Vehicle	A super sort of different kinds of vehicles like car and $Hopper\_truck$.
Armored_Vehicle	A type of vehicle
Chemicals	A type of package
⋮	⋮

Relations R

Name	Signature	Description
Connects	$Route \times Location \times Location$	A rigid relation, true iff there is a route between the associated locations.
PV_Compatible	$Package \times Vehicle$	A rigid relation, true iff the associated package is compatible with the attached vehicle.
RV_Compatible	$Route \times Vehicle$	A rigid relation, true iff the associated route and vehicle are compatible.
At_Vehicle	$Vehicle \times Location$	A flexible relation, true iff the associated vehicle is located at the specified location.
⋮		⋮

Constants $Const$

Name	Signature	Description
Money	Valuable	The value of the Valuable package which will be transported is named *Money*.
Ulm	City	Name of the city.
Loc1, Loc2,...	Location	Locations in a specific city.
⋮	⋮	⋮

transported package and constants loc_1 and loc_2 to represent its source and destination **location** respectively. The expression $At\_package(pck_1, loc_1)$ means that the package pck_1 is at location loc_1. V is a Z-indexed family of infinite disjoint sets of *Variable symbols*. The O and T represent finite disjoint sets of operator and task symbols respectively. The symbol E denotes a Z^*-indexed family, the so-called *elementary operation symbols*. They provide for each flexible symbol R a so-called *add operation* $(+R)$ and a *delete operation* $(-R)$. Finally, L is an infinite set of labels used for identifying different occurrences of identical tasks.

3.3 Tasks

In artificial intelligence planning, changes in the real world are represented by actions or tasks. In our framework, a task schema $t(\bar{\tau})$ is specified by a tuple $\langle type, prec(t(\bar{\tau})), eff(t(\bar{\tau}))\rangle$.

Definition 2 (Task). *For a given logical language* $\mathbb{L}$*, a task schema is defined by a structure* $t(\bar{\tau}) = \langle type, rec(t(\bar{\tau})), eff(t(\bar{\tau}))\rangle$

where

- t is a task symbol,
- $type$ is a kind of task: abstract task or primitive task.

Table 3.2 Primitive task

Task Arguments	Task
Task Name	$Deliver$
Parameters	$?pck$: **Package**
Type	Primitive
Pre-condition	$\{Fees\_collected(?pck), Have\_Permit(?pck)\}$
effects: eff$^-$ **effects: eff$^+$**	$\{\neg Fees\_collected(?pck), \neg Have\_Permit(?pck),$ $delivered(?pck)\}$

- $\bar{\tau} = \tau_1, \tau_2, \cdots, \tau_n$ are the list of variables which belong to the set of variables V. They are called *task parameters*.
- $prec(t(\bar{\tau}))$ specifies the pre-conditions of the task $t(\bar{\tau})$.
- $eff(t(\bar{\tau}))$ specifies the post-conditions of the task $t(\bar{\tau})$. It consists of two parts: $eff^+(t(\bar{\tau}))$ which adds new relations or properties to the current world state, the so-called *positive effects*, and $eff^-(t(\bar{\tau}))$ which removes existing relations or properties from the current world state, the so-called *negative effects*. Attention should be paid to the fact that the set of pre- and post- conditions are sets of literals over the relation R in the logical language $\mathbb{L}$.

In our approach, both primitive and abstract tasks show pre-conditions and effects. For example, as shown in table 3.2, a primitive task *Deliver* states that delivering a package requires collecting the fees as well as taking the permissions that are needed to deliver the specified package. As a result, the specified package is delivered.

As documented in table 3.3, an abstract task *Load* states that the process of loading a package from specific location to a vehicle requires the package and the vehicle to be at the same location and that the identified vehicle should be compatible with the specified package. As a result, the

Table 3.3 Complex or abstract task

Task Arguments	Task
Task Name	$Load$
Parameters	$?pck$: **Package**, $?v$: **Vehicle**, $?l$: **Location**
Type	Abstract
Pre-condition	$\{At\_package(?pck, ?l)$, $At\_vehicle(?v, ?l)$, $PV\_compatible(?pck, ?v)\}$
effects: eff$^-$ **effects: eff$^+$**	$\{\neg At\_package(?pck, ?l)$, $At\_package(?pck, ?v)\}$

specified package is loaded into the vehicle.

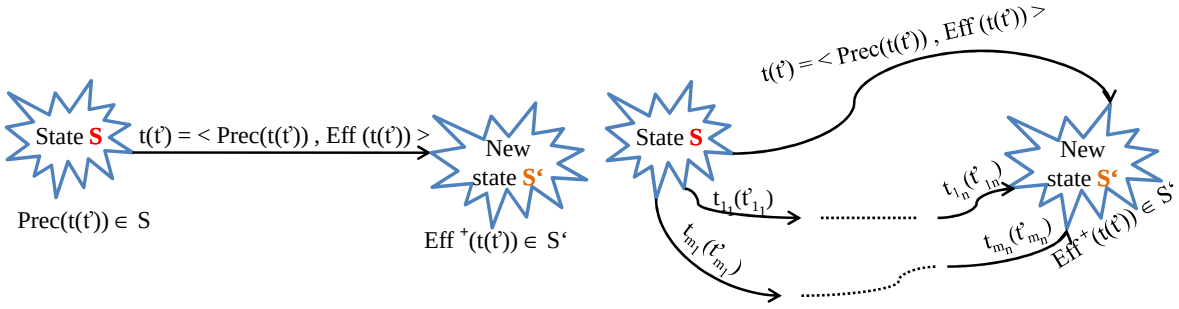
A state s is a finite set of ground atoms[1] in $\mathbb{L}$. A state tells us which ground atoms are currently true: if α is a ground atom, then α is true in the state s if and only if $\alpha \in s$. Therefore, a task $t(\bar{\tau})$ is called *applicable* in a state s, if the literals[2] of its pre-conditions $prec(t(\bar{\tau}))$ are present in the state s. If a task $t(\bar{\tau})$ is applicable in a state s, its application leads to the new state $s^{'}$.

$$s^{'} = (s \cup eff^{+}(t(\bar{\tau}))/eff^{-}(t(\bar{\tau})))$$

It is important to note that an instance of the task schema is a copy of the schema where all task parameters are substituted by new variables through a well sorted variable replacement. The semantics of abstract tasks are based on a sequence of tasks which are provided by the available primitive task schemata. As depicted in figure 3.2(a) and (b), a primitive task is performed directly, while an abstract task requires further decomposition to be executed.

[1] An atom is a predicate symbol followed by a list of terms.

[2] A literal is either an atom *(in which case we say the literal is positive)*, or the negation of an atom *(in which case we say the literal is negative)*



(a) Implementations of a primitive task **(b) Implementations of an abstract task**

Figure 3.2 Implementations of tasks (primitive and abstract)

3.4 Domain Model

As has been previously described, the term *domain model* refers to all the knowledge regarding the real world application area. This knowledge is necessary for generating a solution plan for the given planning problem.

Definition 3 (Domain Model). *For a given logical language* $\mathbb{L}$*, a domain model is defined as a tuple* $D = \langle \mathbb{Q}, \mathbb{M}, \mathbb{T} \rangle$.

$\mathbb{Q}$ represents the set of all objects and relations which exist in the application domain, $\mathbb{M}$ is a model structure that represents decomposition methods and $\mathbb{T}$ represents a set of abstract and primitive task schemata. As we said, abstract tasks do not correspond to a single primitive task in the real world and are thereby not directly executable by human users. Instead, abstract tasks can be seen as constraints for the plans that require and achieve pre-conditions and post-conditions of these abstract tasks and can thus be regarded as pre-defined standard solutions.

A decomposition method $m = \langle t(\bar{\tau}), LVC, p \rangle \in \mathbb{M}$ relates an abstract task $t(\bar{\tau})$ to a partial plan p that implements an abstract solution for a task $t(\bar{\tau})$. Additionally, each method includes a set of local variable constraints LVC to map variables in the abstract task $t(\bar{\tau})$ to variables in the task network *i.e. set of tasks in a partial plan* p. The set of decomposition methods covers all possible

state-refinements of the corresponding abstract task. Therefore, each abstract task has a number of different methods that can be used for its implementation.
For example, table 3.4 shows three different methods from *UM-Translog* domain used to refine the abstract task *load* (See Table 3.3).

Finally, a domain model D constitutes the terminology, concepts, and the relationships between objects for the corresponding course of actions. Now we are ready to introduce the notion of a plan in the next section.

3.5 Plans

In classical state-based planning, a plan is a sequence of actions with completely ordered plan steps. The plan in our approach, however, is a partial order plan. A partial order plan is a plan with partially ordered plan steps.

Definition 4 (Plan). *For a given logical language $\mathbb{L}$, a domain model D, a partial plan P is defined by the tuple $P = \langle TE, C \rangle$*

where,

- TE: is a finite set of plan steps or task expressions $te = \ell : t(\bar{\tau})$, where t is a partially grounded abstract or primitive task. $\ell \in L$ is a unique label in order to distinguish different occurrences of the same task within the same plan. Intuitively, the list of parameter variables $\bar{\tau} = \tau_1, \tau_2, \cdots, \tau_n$ are assumed to be unique in TE.

- C: is a set of constraints, the so-called *Constraint set of* P. These constraints involve three tuples $C = \langle \prec, VC, CL \rangle$: where,

  1. $\prec$: is a finite set of explicit ordering constraints on the plan steps TE. They take the form $te_i \prec te_j$ with task expressions $te_i, te_j \in TE$. The ordering constraints specify

Table 3.4 Decomposition methods for abstract task **Load**

Reference Task	$Load(?pck : Package, ?v : Vehicle, ?\ell : Location)$	
Decomposition Methods		
m_1: Load-Regular-Vehicle	m_2: Load-Tanker-Vehicle	m_3: Load-Airplane
Signature $Load\_Reg\_Veh(?Rp, ?Rv, ?R\ell)$	 $Load\_Tan\_Veh(?Tp, ?Tv, ?T\ell)$	 $Load\_Airplane(?Ap, ?Av, ?A\ell)$
Local Variable Constraints (LVC) $?Rp$: **package**, $?Rv$: **Vehicle**, $?R\ell$: **Location**	 $?Tp$: **package**, $?Tv$: **Vehicle**, $?T\ell$: **Location**	 $?Ap$: **package**, $?Av$: **Vehicle**, $?A\ell$: **Location**
Task Network $(\ell_1 : open\_door(?\ell_1.v))$ $(\ell_2 : load\_pack.(?\ell_2.p, ?\ell_2.v, ?\ell_2.l))$ $(\ell_3 : close\_door(?\ell_3.v))$	 $(\ell_1 : conn.\_hose(?\ell_1.t, ?\ell_1.p))$ $(\ell_2 : open\_valve(?\ell_2.t))$ $(\ell_3 : fill\_tank(?\ell_3.p, ?\ell_3.t, ?\ell_3.l))$ $(\ell_4 : close\_valve(?\ell_4.t))$ $(\ell_5 : disconn.\_hose(?\ell_5.t, ?\ell_5.p))$	 $(\ell_1 : att.\_conv.\_ramp(?\ell_1.pv, ?\ell_1.p, ?\ell_1.l))$ $(\ell_2 : open\_door(?\ell_2.rv))$ $(\ell_3 : load\_package(?\ell_3.p, ?\ell_3.v, ?\ell_3.l))$ $(\ell_4 : close\_door(?\ell_4.rv))$ $(\ell_5 : det.\_conv.\_ramp(?\ell_5.v, ?\ell_5.r, ?\ell_5.l))$
Method Constraints		
Ordering Constraints $(\ell_1 \prec \ell_2), (\ell_2 \prec \ell_3)$	 $(\ell_1 \prec \ell_2), (\ell_2 \prec \ell_3),$ $(\ell_3 \prec \ell_4), (\ell_4 \prec \ell_5)$	 $(\ell_1 \prec \ell_2), (\ell_2 \prec \ell_3)$ $(\ell_3 \prec \ell_4), (\ell_4 \prec \ell_5)$
Variable Constraints $?\ell_1.v = ?Rv, ?\ell_1.p = ?Rp, \cdots,$ $?\ell_2.v = ?Rv, \cdots, ?\ell_3.v = ?Rv$	 $?\ell_1.t = ?Tv, ?\ell_1.p = ?Tp, \cdots$ $, \cdots\cdots, ?\ell_5.p = ?Tp$	 $?\ell_1.v = ?Av, ?\ell_1.l = ?Al, \ell_2.v = ?Rv$ $?\ell_3.p = ?AP, \cdots, \ell_5.l = ?A\ell$
Causal Link $\langle \ell_1, Door\_Open(?\ell_1.v), \ell_3 \rangle$	 $\langle \ell_1, Hose\_Conn.(?\ell_1.t, ?\ell_1.p), \ell_3 \rangle,$ $\langle \ell_2, Valve\_Open(?\ell_2.t), \ell_3 \rangle,$ $\langle \ell_1, Hose\_Conn.(?\ell_1.t, ?\ell_1.p), \ell_5 \rangle,$ $\langle \ell_2, Valve\_Open(?\ell_2.t), \ell_4 \rangle$	 $\langle \ell_1, Ramp\_Conn.(?\ell_1.r, ?\ell_1.pv), \ell_5 \rangle,$ $\langle \ell_2, Door\_Open(?\ell_2.v)\ \ell_4 \rangle$

that the task te_i must finish before the beginning of the task te_j. Note that the set of ordering constraints in $\prec$ are produced as a result of the planning process or pre-defined by the domain model.

2. VC: is a finite set of variable constraints. They are a set of co-designations or non-co-designations used for grounding tasks and to force equality or inequality between variables. More formally, for two tasks t and $\bar{t}$, $\tau_i(t) = \tau_j(\bar{t})$ constraints $\tau_i(t)$ and $\tau_j(\bar{t})$ to be identical and, for co-designating variables with constants, $\tau_i(t) = c$ constraints the variable $\tau_i(t)$ to be equal the constant $c \in C_z$, where $z \in Z$ is the sort of c. Non-co-designations are defined similarly.

3. CL: is a finite set of causal link constraints. They have the form $te_i \xrightarrow{\varphi} te_j$ or $\langle te_i, \varphi, te_j \rangle$, indicating that the task expression $te_i = \ell_i : t_i(\bar{\tau}_i)$ establishes a pre-condition φ of a task expression $te_j = \ell_j : t_j(\bar{\tau}_j)$, where $\varphi \in eff(te_i) \wedge prec(te_j)$.

As an example, figure 3.3 shows the plan p_{load} for loading a package pck into the vehicle car. The following tasks tasks are used:

$te_{init}(\ ) = \langle \{\ \}, \{At\_package(pck, loc), At\_vehicle(car, loc), PV\_compatible(pck, car)\} \rangle$

$Open\_door(?v_1) = \langle \{\neg Door\_open(?v_1)\}, \{Door\_open(?v_1)\} \rangle$

$Close\_door(?v_2) = \langle \{Door\_open(?v_2)\}, \{\neg Door\_open(?v_2)\} \rangle$

$Load\_package(?p, ?v_3, ?\ell) = \langle \{At\_package(?p, ?\ell), At\_vehicle(?v_3, ?\ell), PV\_compatible(?p, ?v_3)\},$
$\{At\_package(?p, ?v_3), \neg At\_package(?p, ?\ell)\} \rangle$

$te_{goal}(\ ) = \langle \{At\_package(pck, car), \neg Door\_open(car)\}, \{\ \} \rangle$.

$p_{load} = \langle TE_{load}, C_{load} \rangle$ is a plan where:

$TE_{load} = \{\ell_1 : te_{init}(), \ell_2 : open\_door(?\ell_2.v_1), \ell_3 : close\_door(?\ell_3.v_2),$
$\ell_4 : Load\_package(?\ell_4.p, ?\ell_4.v_3, ?\ell_4.\ell), \ell_5 : te_{goal}()\}$

$C_{load} = \langle \prec_{load}, VC_{load}, CL_{load} \rangle$, where:

$\prec_{load} = \{\ell_1 \prec \ell_2, \ell_2 \prec \ell_4, \ell_4 \prec \ell_3, \ell_3 \prec \ell_5\}$

$$VC_{load} = \{?\ell_2.v_1 = ?\ell_3.v_2, ?\ell_2.v_1 = \ell_4.?v_3, ?\ell_2.v_1 = car, ?\ell_4.p = pck, ?\ell_4.\ell = loc\}$$

$$CL_{load} = \{\langle \ell_1, \neg Door\_open(car), \ell_2\rangle, \langle \ell_1, At\_package(pck, loc), \ell_4\rangle,$$
$$\langle \ell_1, At\_vehicle(car, loc), \ell_4\rangle, \langle \ell_1, PV\_compatible(pck, car), \ell_4\rangle,$$
$$\langle \ell_2, Door\_open(?\ell_2.v_1), \ell_3\rangle, \langle \ell_4, At\_package(?\ell_4.p, ?\ell_4.v_3), \ell_5\rangle,$$
$$\langle \ell_3, \neg Door\_open(?\ell_3.v_2), \ell_5\rangle\}$$

Although causal link constraints impose a partial order between plan steps, our plan identifies explicit ordering constraints between plan steps if causal threats need to be resolved *(by promotion or demotion)* or if they are already present in a predefined plan of the domain model.
This plan describes which actions need to be taken in order to load a package pck into the vehicle car. The first and last tasks are artificial tasks. The **Open_door** task corresponds to the action of opening the door of car which is used to load the package pck into it. For this being possible, the vehicle has to be in the state that actually allows to load the package. The **Load_package** task has the pre-conditions *At_package(pck,loc)*, *At_vehicle(car,loc)* and *PV_compatible(pck,car)*. After that, the **Close_door** task which corresponds to the action of closing the door of the vehicle car will be executed. Adding a task that produces the pre-condition literals *At_package(pck,car)* and ¬ *Door_open(car)* for the final task te_{goal} completes the plan.

3.6 Planning Problems and Solutions

A planning problem in HTN planning is formulated over a domain model D and consists of the initial world state description s_{init} which is a set of positive atoms that represent what conditions are true initially, and the initial plan p_{init}. Formally we define:

Definition 5 (Planning Problem). *For a given logical language $\mathbb{L}$ and domain model D, a planning problem Π is defined by the tuple $\Pi = \langle D, s_{init}, p_{init}\rangle$, where, s_{init} is an initial state description and p_{init} is an initial partial plan.*

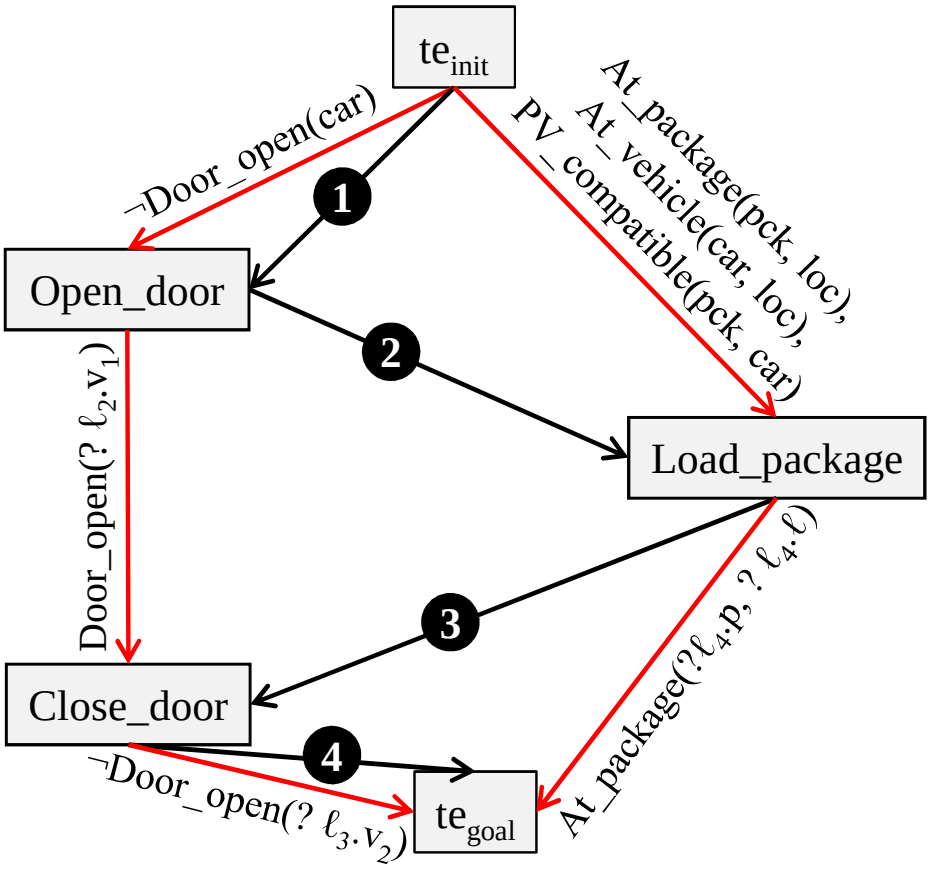


Figure 3.3 The Plan p_{Load} describes how the package pck can be loaded into the vehicle car. Red lines represent causal link constraints between tasks and black lines represent ordering constraints between tasks.

The initial plan $p_{init} = \langle TE_{init}, C_{init} \rangle$ is a consistent partial plan.

Where, The task network TE_{init} contains two artificial tasks te_{init} and te_{goal} which are used to provide the initial and goal state, respectively. The facts in s_{init} are used as effects of the task te_{init} while task te_{goal} has the desired goal state as a pre-condition. All other tasks get ordered in between.

C_{init} represents a set of initial plan constraints $C_{init} = \langle \prec_{init}, VC_{init}, CL_{init} \rangle$. It is important to notice that our planner assumes any atom which is not in the initial state s_{init} to be false.

For example, in the *UM-Translog* domain, the definition of the planning problem Π_{load} can be given as follows: $\Pi_{load} = \langle D, s_{init}, p_{init} \rangle$, where

$$s_{init} = \{At\_package(pck, loc), At\_vehicle(car, loc), PV\_compatible(pck, car)\}$$

and the initial partial plan $p_{init} = \langle TE_{init}, C_{init} \rangle$ in the planning problem Π_{Load} is formalized as follows :

$$TE_{init} = \{\ell_1 : te_{init}(), \ell_2 : Load(?\ell_2.p, ?\ell_2.v, ?\ell_2.\ell), \ell_5 : te_{goal}()\},$$

and the set of initial constraints C_{init} are:

$$\prec_{init} = \{(\ell_1 \prec \ell_5), (\ell_1 \prec \ell_2), (\ell_2 \prec \ell_5)\},$$
$$VC_{init} = \{?\ell_2.p = pck, ?\ell_2.v = car, ?\ell_2.\ell = loc\},$$
$$CL_{init} = \emptyset$$

As mentioned before, the tasks te_{init} and te_{goal} in TE_{init} are artificial tasks used to provide the beginning and end of a plan, respectively. They occur only once in each plan and all other tasks such as $\ell_2 : Load(?\ell_2.p, ?\ell_2.v, ?\ell_2.\ell)$ are always ordered in between. The facts in initial state s_{init} provide the effects of te_{init} task. The pre-conditions of the task te_{goal} correspond to the desired goal state. In our running example, any valid decomposition of the initial plan p_{init} solves the given planning problem without the need to explicitly satisfy a goal state.

A plan $p_{sol} = \langle TE_{sol}, C_{sol} \rangle$ is a *solution* to the planning problem $\Pi = \langle D, s_{init}, p_{init} \rangle$ if the following solution criteria are met:

1. A plan p_{Sol} is a refinement of p_{init}. Informally, we call a plan p is a refinement of p_{init} if the plan p results from applying plan modifications to the plan p_{init}. A plan modification is the insertion of a plan element, *i.e., an element from the set of task expressions, temporal orderings, variable constraints and causal links*. The only modification that is not a pure insertion is the application of a method: it replaces an abstract task by implementing a task network and adapts the variable constraints and causal links. The formal description of the modification is discussed in details in section 3.7.2.

2. All plan steps in the task networks of a plan p_{sol} are primitive tasks.

3. All pre-conditions of every plan step in the task network of a plan p_{sol} are supported by a causal link, *i.e., for each pre-condition φ of a plan step $te_j \in TE_{sol}$ there exists a causal link $\langle te_i, \varphi, te_j \rangle \in CL_{sol}$ with $te_i \in TE_{sol}$.*

4. none of the causal links in CL_{sol} is threatened, *i.e., for each causal link $\langle te_i, \varphi, te_j \rangle \in CL_{sol}$ the ordering constraints in $\prec_{sol}$ ensure that no plan step te_k with an effect that implies $\neg\varphi$ can be consistently placed between plan steps te_i and te_j.*

5. The ordering and variable constraints in a plan p_{sol} are consistent, i.e., there is no plan step $te \in TE_{sol}$, such that $te \in^*_{CL} te$ *($\prec$ does not induce cycles on plan steps TE)* and no $v \in V$ for $z \in Z$, such that $VC \models v \neq v$.

6. All tasks in a plan p_{sol} are grounded. That is, all variables are co-designated to some constant.

In our approach, any solution must be a decomposition of the initial plan p_{init}. Since abstract tasks are regarded as non-executable, criterion 2 ensures that only executable tasks, *i.e., primitive tasks*, are part of a solution plan. Criterion 3 ensures the applicability of tasks in a plan: in order for a task to be applicable in a state s, all its literals of its pre-condition must hold in s. This can be ensured by establishing appropriate causal links. Criterion 4 guarantees that every plan step in all linearizations of a plan is applicable in the sense of criterion 3: causal threats can cause a literal of the pre-condition of a plan step to be false in some linearizations although it is supported by a causal link. Since we require every linearization of a plan to be a valid solution, causal threats have to be eliminated. Criterion 5 is obviously necessary for constituting meaningful plans since a task can neither be ordered before itself nor can a constant be different from itself. Criterion 6 maps the variables used by tasks onto the objects available in the modeled world. In the next section we will discuss how to generate a solution plan p_{sol} from the initial plan p_{init}.

3.7 Plan Generation

The process of generating a plan is adapted from the work of Schattenberg [46]. Plan generation means a stepwise refinement of the initial plan p_{init} into a partial plan p until the solution plan p_{sol} is reached. Before presenting our refinement algorithm in more detail, we will illustrate the main requirements of our refinement algorithm: flaws and plan modifications.

3.7.1 Flaws

A flaw is a data structure that refers to all plan elements which violate the solution criteria.

Definition 6 (Flaw). *For a given planning problem specification* Π *and a partial plan* $P = \langle TE, C \rangle$ *that is not a solution to* Π*, a flaw* $f = \{f_1, f_2, \cdots, f_n\}$ *consists of a set of critical plan components* f_i *of* P*. It represents a defect in which these components are involved.*

Let F be the set of all flaws. Its subsets F_x represent classes[3] of flaws for a partial plan P. Then, a flaw class is the set of all possible flaws of a specific type in the current plan such as flaw class of abstract tasks $f_{AbstractTask}$ which refers to the abstract tasks in the current plan like *Load task* in Π_{load} and $f_{OpenPreCondition}$ which points to those tasks that are not fully supported *i.e., tasks that still have pre-condition not achieved by another task.* The set of all flaw classes is computed through a detection module f_x^{det}. A detection module $f_x^{det}$is a function that, given a partial plan P, a domain model D, and a planning problem specification Π, returns all flaws of type x that are present in the current plan.

3.7.2 Plan Modification

Each refinement in our approach focuses on a single plan defect *(flaw)* and generates a new plan for each way of resolving that flaw. However, refinement steps include the decomposition of

[3]The complete definitions of various flaw classes can be found in [7].

abstract tasks by selecting the appropriate methods, the insertion of causal links to support open pre-conditions of plan steps as well as the insertion of ordering and variable constraints. We call such a refinement step a *plan modification*.

Definition 7 (Plan Modifications). *For a given partial plan $P = \langle TE, C \rangle$ and a domain model D a plan modification is defined as the structure $\omega = (E^{\oplus}, E^{\ominus})$, where $E^{\oplus}$ and $E^{\ominus}$ are disjoint sets of elementary additions and deletions of plan elements, respectively, over the partial plan P and domain model D.*

All elements in $E^{\ominus}$ are elements in plan P *($E^{\ominus} \subseteq P$)* such as plan steps in TE, or constraints elements in constraints set C which includes constraints $\prec$, VC and CL, while $E^{\oplus}$ consists of new elements such as new plan steps or new constraints *($E^{\oplus} \cap P = \emptyset$)*. This generic definition makes all changes a modification imposes on a plan explicit. Applying a modification $\omega = (E^{\oplus}, E^{\ominus})$ to a current plan P returns a new plan $P^{'}$ that is obtained from P by adding all elements in $E^{\oplus}$ and removing those of $E^{\ominus}$.

The set of all modifications is denoted by Ω. Its subsets Ω_y represent modification classes. The set of all modification classes is computed through a modification module f^{ModGen}. Each plan modification class Ω_y can be used to handle a specific flaw class F_x. Therefore, the plan modification class $\Omega_y \in \Omega$ is suitable to solve the flaw class $F_x \in F$ iff there exists a partial plan p which contains a flaw $f \in F_x$ and a plan modification $\omega \in \Omega_y$, such that the new plan $P^{'}$, which is produced by applying a plan modification ω on a plan P, does not contain flaws f anymore. A plan modification might introduce new flaws.

Therefore, in order to separate the computation of flaws from the computation of modifications, a modification triggering function α is introduced to relate each flaw class to those modification classes that are suitable for generating refinements that solve the respective flaw such as $\alpha(F_{AbstractTask}) = \Omega_{ExpandTask}$ (See Table 3.5). Note that the inconsistent ordering flaw can not be resolved by adding constraints. Therefore, it does not have a modification function to solve it.

Table 3.5 This table lists the different kinds of flaw classes and the corresponding ways to solve those flaws

Flaw Class F_x	Modification class Ω_y
$F_{AbstractTask}$	$\Omega_{ExpandTask}$
$F_{OpenPrecondition}$	$\Omega_{AddCausalLink}, \Omega_{ExpandTask}, \Omega_{InsertTask}$
$F_{CausalThreat}$	$\Omega_{ExpandTask}, \Omega_{AddOrdering}, \Omega_{BindVariable}$
$F_{InconsistentOrdering}$	–
$F_{UnboundedVariable}$	$\Omega_{BindVariable}$

For example, assume the abstract task flaw $F_{AbstractTask}$ for a task $te = \ell : t(\bar{\tau}) \in TE,\ \ell \in L$ can be resolved by applying the plan modification $\Omega_{ExpandTask}$ for the appropriate method m.

$m = \langle t, \langle TE_m, C_m \rangle \rangle \in \mathbb{M}$ where $C_m = \langle \prec_m, VC_m, CL_m \rangle$.

$(ExpandTask, \{\ominus te\} \cup \{\oplus te_m | te_m \in TE_m\} \cup \{\oplus(te_{m1} \prec te_{m2}) | (te_{m1} \prec te_{m2}) \in \prec_m\} \cup$

$\{\oplus(te_{m1} \xrightarrow{\varphi} te_{m2}) | (te_{m1} \xrightarrow{\varphi} te_{m2}) \in CL_m\} \cup \{\oplus(v = \tau) | (v = \tau) \in VC_m\} \cup$

$\{\oplus(v \neq \tau) | (v \neq \tau) \in VC_m\} \cup \{\ominus(\acute{te} \prec te), \oplus(\acute{te} \prec te_m) | (\acute{te} \prec te), te_m \in TE_m\} \cup$

$\{\ominus(T \prec \acute{te}), \oplus(te_m \prec \acute{te}) | (te \prec \acute{te}), te_m \in TE_m\} \cup$

$\{\ominus(te \xrightarrow{\varphi} \acute{te}), \oplus(te_m \xrightarrow{\varphi} \acute{te}) | (T \xrightarrow{\varphi} \acute{te}) \in CL, te_m \in TE_m, \varphi \in eff(te_m)\} \cup$

$\{\ominus(\acute{T} \xrightarrow{\varphi} te), \oplus(\acute{te} \xrightarrow{\varphi} te_m) | (\acute{te} \xrightarrow{\varphi} te) \in CL, te_m \in TE_m, \varphi \in prec(te_m)\})$

Note that during an expansion process the abstract task te is replaced by the decomposition network TE_m with all its sub-tasks being ordered between the predecessors and successors of the abstract task and all other constraints are modified according to the new sub-tasks (See Figure 3.4).

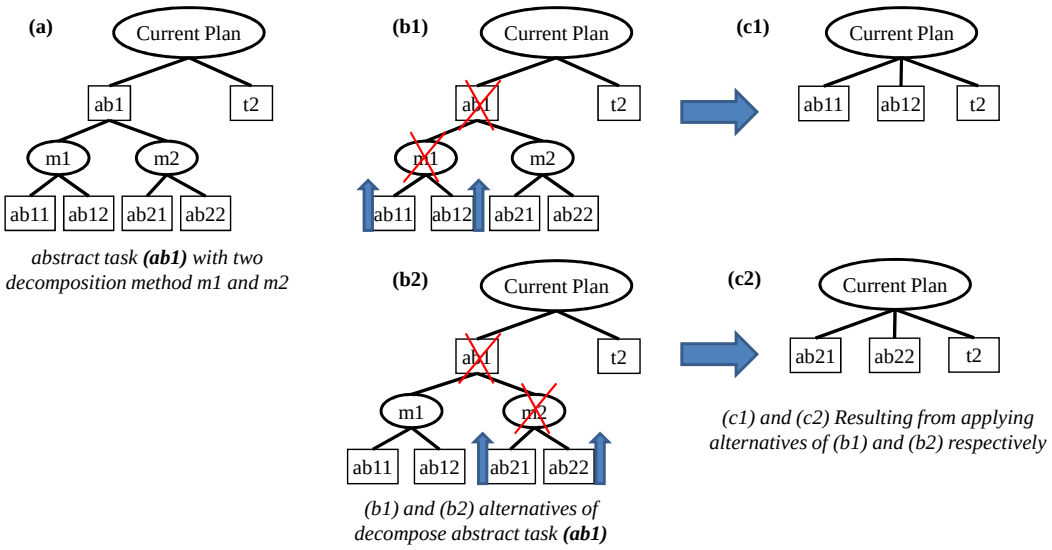


Figure 3.4 How to refine an abstract task

3.7.3 Search Strategy

As we mentioned before, the detection module is used to detect the set of flaws in the current plan, while the refinement or modification module is used to generate refinement alternatives. Then a search strategy[4] used to decide which paths in the refinement space to pursue. In general, our refinement planning divides the search strategy into two processes: the process of selecting the refinement options, the so-called *modification selection strategy* and the process of selecting the path that is to be followed in the *Induced search space* (Definition 9) which is generated by refinement options, the so-called *plan selection strategy*. The modification and plan selection strategies impose a partial order on the respective input refinement options and these strategies are therefore suited for a sequenced arrangement.

A basic form of a modification selection strategy is either to prefer specific classes of plan modifications, *e.g., we prefer the expansion of tasks or we try to delay an assignment of variables to*

[4]For more details about search strategies refer to chapter 5

constants as long as possible.

In the framework used here, Schattenberg [8] encoded the preference of a modification class as follows: Let Ω_P be the modification class that is to be prefered by the modification-selection module $f^{ModSel}_{Pref-\Omega_P}$. Then the corresponding function is defined as in definition 8[5].

Definition 8 (**Modification Selection Preference**). *For a partial plan p, sets of flaws $f_1, f_2, \cdots, f_m \in F$, and sets of plan modifications $\omega_1, \omega_2, \cdots, \omega_n \in \Omega$, a modification selection preference is $\langle \omega_i, \omega_j \rangle \in f^{ModSel}_{Pref-\Omega_P}\ (p, \{f_1, f_2, \cdots, f_m\}, \{\omega_1, \omega_2, \cdots, \omega_n\})$ if $\omega_i \in \Omega_P$ and $\omega_j \notin \Omega_P$.*

A modification selection strategy can be specified as the combination of more than one base class such as $\Omega_P = \Omega_{P_a} + \Omega_{P_b}$. In this case the modification function Ω_{P_a} is called the first *(Primary)* strategy, while Ω_{P_b} is the *secondary* strategy. If the primary strategy does not prefer one option over another option, the secondary strategy is followed and so forth until finally a random preference is assumed.

3.7.4 Refinement Algorithm

Before we present our refinement planning algorithm in more detail, we define the search space induced by the HTN planning problem Π.

Definition 9 (**Induced Search Space**). *Let $\mathcal{P}_\Pi = \langle \mathcal{V}, \mathcal{E} \rangle$ be the directed acyclic graph which represents the (possibly infinite) search space induced by a planning problem Π. Then, the set of vertices $\mathcal{V}$ is the set of plans in the search space and the set of edges $\mathcal{E}$ corresponds to the set of used plan modifications. By abuse of notation, we write $P \in \mathcal{P}_\Pi$ to state $P \in \mathcal{V}$. The root of $\mathcal{P}_\Pi$ is the initial plan of Π. The direct successors of a plan $P \in \mathcal{P}_\Pi$ are all plans P', such that P' resulted from P by applying a plan modification ω to P. Then, $\omega \in \mathcal{E}$.*

[5] Note that plan selection strategy is built accordingly.

With the defined flaw and modification detection modules as well as a search strategy, we can now illustrate our refinement planning algorithm (Algorithm 2) which takes the initial plan p_{init} of the planning problem Π as an input and refines it stepwise until a solution plan p_{sol} is found. Our algorithm performs an informed search, guided by search strategies in the search space induced by the HTN planning problem Π.

Algorithm 2: Refinement Planning Algorithm

Input : The sequence $\texttt{Fringe} = \langle p_{init} \rangle$.

Output: `Solution Plan` (p_{sol}) or `fail`.

1 **while** $\texttt{Fringe} = \langle p_1 \dots p_n \rangle \neq \varepsilon$ **do**

2 $\quad F \longleftarrow f^{\text{FlawDet}}(p_1)$

3 $\quad$ **if** $F = \emptyset$ **then return** p_1

4 $\quad \langle \omega_1 \dots \omega'_n \rangle \longleftarrow f^{\text{ModOrd}}(\bigcup_{\texttt{f} \in F} f^{\text{ModGen}}(\texttt{f}))$

5 $\quad \texttt{succ} \longleftarrow \langle \mathit{apply}(\omega_1, p_1) \dots \mathit{apply}(\omega'_n, p_1) \rangle$

6 $\quad \texttt{Fringe} \longleftarrow f^{\text{PlanOrd}}(\texttt{succ} \circ \langle p_2 \dots p_n \rangle)$

7 **return** *fail*

The fringe in the algorithm is a sequence of plans $\langle p_1 \dots p_n \rangle$ ordered by the deployed search strategy. It contains all unexplored plans that are direct successors of visited non-solution plans in $\mathcal{P}$. According to the used search strategy, a plan p_i leads more quickly to a solution than plans p_j for $j > i$. The current plan under consideration is always the first plan of the fringe. The planning algorithm loops 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 plan component that is involved in the violation of a solution criterion. In HTN planning, the presence of an abstract task raises a flaw that includes that task, a causal threat

consists of the causal link and the threatening plan step, and so on. If no flaws can be found, the plan is a solution and returned (line 3). In line 4, all plan modifications are calculated by the modification generating function f^{ModGen}, which addresses all published flaws. 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 calculated (line 5) and are put in front of the fringe in line 6. Second, the plan ordering function f^{PlanOrd} orders the updated fringe according to its strategy. This step can also be used in order to discard plans (i.e., to delete plans permanently from the fringe). This is useful for plans that contain unresolvable flaws like an inconsistent ordering of tasks. If the fringe becomes empty, no solution exists and `fail` is returned.

Chapter 4

Landmarks in Hierarchical Planning

In recent years, the exploitation of knowledge gained by pre-processing planning domains and / or problem descriptions has proven to be an effective means to improve planning efficiency. In this chapter, we would like to give an overview over a hot pre-processing technique which plays an important role in the field of AI planning, as well as introduce our novel landmark pre-processing technique in the context of hierarchical planning approaches.

4.1 Pre-processing Planning Techniques

A pre-processing process is an automatic method used to analyze the planning domain and to extract knowledge about the domain and problem description. The outcome of a pre-processing technique is always a logical consequence of the domain and problem description, so it is different from machine learning in that nothing is learned. In general, the pre-processing technique can be done either "off-line"– only once for each domain, or "on-line"– for every problem instance. There are a lot of pre-processing techniques described in literature, most of which are integrated into a specific planning system and are not available as separate modules because the outcome of the pre-processing technique does not directly produce control knowledge. It needs some sort of

control search technique to be effective.

Fox et al. [49, 50] introduced a pre-processing technique to detect and eliminate symmetries from planning problems. A planning problem has a high degree of symmetry if there are a lot of functionally identical objects that can not be usefully distinguished. Symmetries are automorphisms on the structures that characterize a planning problem *i.e. symmetries represent redundancy in the structure of the problem definition*. Fox's technique can discover different kinds of symmetry such as: *Plan permutation* which means two plan fragments can generate the same facts from the same state. As depicted graphically in figure 4.1(a), a plan which uses truck T_1 to transport package P_1 from location L_1 to L_3 contains the same actions of the plan that uses truck T_2 to transport the same package P_1 from location L_1 to L_3. Figure 4.1(b) shows another kind of symmetry, the so-called *Structure symmetry*, which means several objects have identical structure, such as the set of trucks T_1, T_2, T_3, and T_4 which are not functionally equivalent because they are connected to a location L_1 by different roads, but each one of them can generate a plan that achieves the original goal.

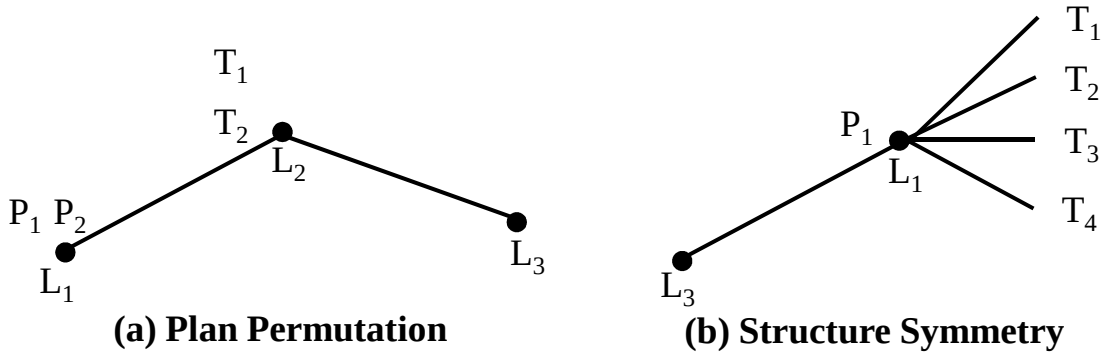


Figure 4.1 Different kinds of symmetry.

STAN planner[1] is a domain independent planner and depends on these kinds of symmetries [51]. It is built on the graph construction and search technique of GRAPHPLAN planner as well as performing pre-processing analysis on the domain model before planning. Although GRAPH-

[1] STAN $\Longrightarrow$ **ST**ate **AN**alysis

PLAN is the most efficient planner in classical state-based planning, it may possibly lead to problems if the planning task description includes irrelevant information. Therefore, Nebel et al. [52] introduced another pre-processing technique depending on removing irrelevant facts and operators from planning problems. After that, Haslum et al. [53] introduced a new technique to change the shape of the search space by changing the representation of facts. They focused on the idea of removing redundant operators in order to reduce the branching factor and speed up the search process. To this end, they defined the notion of *redundant operator sets*. The operator o is redundant with respect to the sequence of operators if it does not add any new effects to the sequence of operators. Thus, the set of operators can be reduced by enumerating the redundant operators and removing them. The results of reducing the set of operators are not unique and each have a different size. It depends on the order of enumeration. For example, suppose we have the following set of operators O:

$$\begin{aligned} O \ = \{ & o_1 : \ (\{p\}, \{q\}, \{p\}), \\ & o_2 : \ (\{q\}, \{p\}, \{q\}), \\ & o_3 : \ (\{p\}, \{r\}, \{p\}), \\ & o_4 : \ (\{r\}, \{p\}, \{r\}), \\ & o_5 : \ (\{q\}, \{r\}, \{q\}), \\ & o_6 : \ (\{r\}, \{q\}, \{r\}), \} \end{aligned}$$

The set of operators O can be reduced in a number of different ways. One reduction yields the set of operators $\{o_1,\ o_2,\ o_3,\ o_4\}$ because the sequence of operators o_2 and o_3 implements operator o_5, while the sequence of operators o_4 and o_1 implements operator o_6. Another reduction given as the set of operators $\{o_1,\ o_4,\ o_5\}$ because the sequence of operators o_5 and o_4 implements operator o_2, while the sequence of operators o_1 and o_5 implements operator o_3 and the sequence of operators o_1 and o_4 implements operator $o_6 \cdots$ etc. Of course, the efficiency of planning performance will be increased when such redundancies are discovered.

Additionally, a number of research efforts have been made to discuss the problem of interactions among the goals of a planning problem. Some of them perform a structure analysis for the planning task before starting the planning process in order to propose new constraints concerning the best order in which to achieve the goals.

Koehler et al. [54, 55] introduced the notion of *reasonable and forced orders* between the goals in the planning problem. The goal G_1 is ordered reasonably prior to goal G_2 *(i.e.,* $G_1 \prec_r G_2$*)*, if the goal G_2 has been achieved and there is no action that can achieve goal G_1 without violating goal G_2. The *Forced goal* $(G_1 \prec_f G_2)$ is defined to avoid deadlock, where goal G_1 is ordered forcedly prior to G_2 when there is no plan anymore that can achieve G_1 in case of achieving goal G_2 first. The set of orders between goals in the given planning problem is collected in a set, the so-called *goal agenda*. Any planning system can use the goal agenda in order to generate the solution plan. Recently, another hot approach, the so-called *Landmark* has been invented to propagate orders between planning goals not only over top goals as it is the case using the *goal-agenda* technique, but also over subgoals that will appear during the planning process.

4.2 Landmarks in Classical State-based Planning

The concept of landmarks in classical state-based planning has received widespread attention and it has been widely discussed how to extract and exploit landmark literals during the planning process. In this section we will briefly illustrate the common landmark approaches (see Figure 4.2).

Initially, the *landmark technique* for classical state-based planning was inspired by the work of Porteous et al. [3] and studied again in more detail in the work of Hoffmann et al. [56]. They define landmarks as a set of literals that have to hold in some intermediate state for every solution plan that solves the given planning problem (see Figure 4.3).

The landmark extraction algorithm proceeds in three phases. (1) Build a **R**elaxed **P**lanning

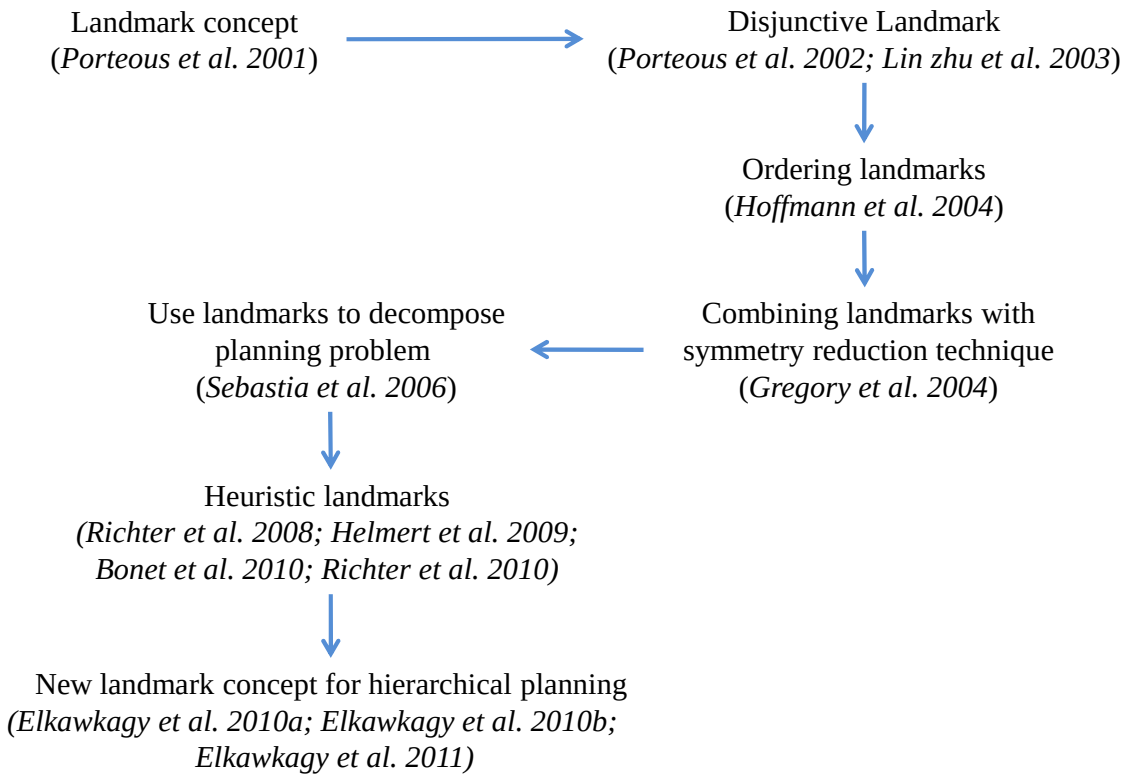


Figure 4.2 A chronology of landmark concept

Graph *(RPG)*. According to the **FF** Planner [2] [57], the relaxation is achieved by simply ignoring the delete lists of all operators. After that, the RPG is built by a forward-chaining technique beginning at the initial world state until all goals are solved or a fixed point is reached. The latter indicates that the relaxed planning problem is unsolved. (2) A regression technique is applied on RPG to extract landmark literals. For each goal in the RPG level, the shared pre-conditions of the actions that achieve the current goal are identified *(i.e., shared pre-conditions are those literals or propositions that are a pre-condition for each of the actions)*. The literals in the set of shared pre-conditions are considered landmark literals and added to the so-called *goal list* for further consideration in the next level. In addition, further landmark literals are found by combining the

[2] FF $\Longrightarrow$ **F**ast **F**orward

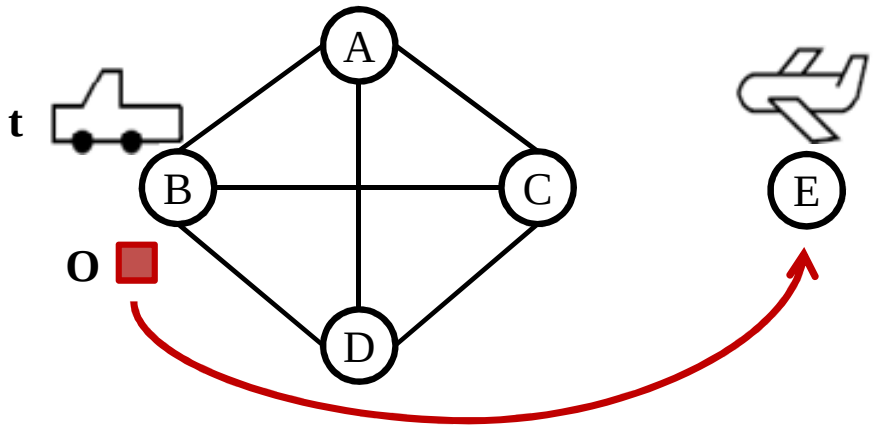


Figure 4.3 Landmark literals: consider a transportation task where the goal is to have package O at location E by airplane P. Package O and truck t are initially located at location B. Then the landmark literals are: literals in the initial state, literals in the goal state as well as the intermediate subgoals such as $Load(O,t)$, $Drive(t,C)$, $Unload(O,C)$, $Fly(P,C)$, $Load(O,P)$, $Fly(P,E)$.

other literals except those in the shared pre-conditions set into a new set, the so-called *union set*. After that, the set of shared pre-conditions of actions which achieve these literals in the union set are computed and considered new landmark literals. Hence, the next lower level in the RPG is considered in order to extract the remaining landmark literals and so on until the initial level is reached. (3) Finally, a filtering procedure is used to evaluate the landmark literals which are produced in the second phase and remove those literals which fail in the test.

For the purpose of exploiting landmark literals during the planning process, Hoffmann et al. [56] introduced some ordering relations between landmark literals. As depicted in figure 4.4, they used landmark literals in order to produce a graph, the so-called ***L**andmark **G**eneration **T**ree (LGT)*, where *Nodes* represent the landmark literals and *Edges* represent the order between them.

There are three types of order between landmark literals. The order between landmark literals l and $\bar{l}$ is called *natural order* $(l \prec \bar{l})$ if and only if l is achieved by a sequence of operators at time i before achieving literal $\bar{l}$ at time j where $(i \prec j)$. On the other hand, the order between them

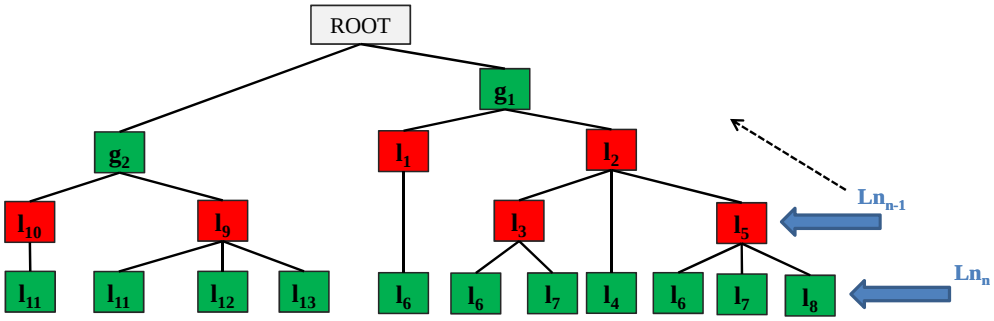


Figure 4.4 Landmark generation tree (LGT): *Red box* represents landmark literals which are extracted, while *green* box represents trivial landmark literals (initial and goal).

is the so-called **necessary order** $(l \prec_n \bar{l})$ if and only if the literal l is immediately pre-requested in the preceding state which achieves landmark literals $\bar{l}$. Finally, **a reasonable order** between landmarks l and $\bar{l}$ $(l \prec_r \bar{l})$ is categorized into two different types. The first type is composed of the top goal g and the landmark literals which are ordered naturally. For example, suppose l and $\bar{l}$ are landmarks, and there is a natural order between them $(l \prec \bar{l})$, where l and the top level goal g are inconsistent, then the landmark literal $\bar{l}$ is ordered reasonably with goal g $(\bar{l} \prec_r g)$. The second type is based on the interactions between landmark literals which have *natural order* and the side effects of the literal in the reasonable ordering. For example, suppose l and $\bar{l}$ are landmark literals and there is a natural order between l and the top level goal g $(l \prec g)$, suppose also there is a reasonable order between $\bar{l}$ and g $(\bar{l} \prec_r g)$. Therefore, a reasonable order between $\bar{l}$ and l $(\bar{l} \prec_r l)$ is propagated in case there is an inconsistency between l and a side effect of $\bar{l}$ *i.e., if the landmark literal* l *is achieved before the landmark literal* $\bar{l}$ *it would have to be destroyed and reestablished again in order to achieve the literal* g.

After that, Hoffmann et al. introduced a simple recursive algorithm to exploit landmark literals during the planning process. The Hoffmann algorithm loops in two phases; First, submitting the leaf nodes Ln_n in the LGT (see Figure 4.4) to any planner in order to achieve literals in these nodes. Once these literals in leaf nodes are achieved and a new state LGT_{s_1} is produced the second

phase is started by removing leaf nodes Ln_n from LGT. After that, the algorithm is called again with new leaf nodes Ln_{n-1} in the LGT and a new state LGT_{s_1}. The algorithm terminates successfully when LGT becomes empty.

Unfortunately, the work of Porteous et al. [3] has overlooked some potentially useful literals such as *disjunctive landmarks* because there is an arbitrary choice of an object involved in an action. For example, consider the following problem from the *UM-Translog* domain, assume a package Pck_1 is at airport A_1 in the initial state and we would like to transport it to airport A_2 using a plane P_1 or P_2, which is initially located at airport A_1. Porteous's extraction algorithm couldn't extract literal $in(pck_1, P_1)$ or $in(pck_1, P_2)$ in its set of landmark literals because pck_1 may be transported by either plane. Therefore, in later work, the landmark concept was extended to extract *disjunctive landmarks* [58]. *A disjunctive landmark* is a set of literals any of which has to be satisfied in the course of a valid plan.

A new propagation algorithm to compute the set of landmark literals is proposed by Zhu et al. [59]. This propagation algorithm eliminates the need for heuristic candidate generation which is used in the work of Porteous et al. [3]. It computes landmark literals by propagating labels *(action or literal label)*, where every action node in the initial graph level is labeled with itself and each literal node in the graph is labeled with the intersection of the labels on its predecessor action nodes. In later levels, an action node in the graph is labeled with the union of the labels on its predecessor literal node. At the end, all labels in the final level *(goal literals)* are landmarks for any solution plans achieving the given goal in the planning problem.

Gregory et al. [60] combine symmetry breaking techniques [49] with landmark analysis to extract disjunctive landmarks. They reduce the planning problem by using symmetry breaking techniques and then perform the landmark procedure on a reduced problem. Afterwards, Sebastia et al. [61] proposed a pre-processing technique that arranges landmark literals into consistent groups with minimum interaction between them. These groups are called ***I**ntermediate **G**oals (IGs)*. Then, a set

of sub-problems is created. Each sub-problem $p_i = \langle A, IS_{i-1}, IG_i \rangle$ consists of three tuples; a set of actions A, the intermediate state IS_{i-1} which generated from solving the previous sub-problem p_{i-1}, as well as the specified goals in the current intermediate goal IG_i. These sub-problems are solved sequentially by any planner and their solution can be easily combined to constitute a solution plan for the original planning problem. In the same work, the method of solving sub-problems is developed to solve these sub-problems concurrently by approximating the intermediate state IS_i that would result after solving each intermediate goal IG_i.

Recently, Richter et al. [62] proposed an alternative algorithm to identify landmarks for a multi-valued state variable representation of planning tasks (SAS^+ planning formalism [63, 64]). They combined landmark literals with a heuristic search framework in order to improve the planning performance. In their work, landmark information can be used during the planning process in two ways: The goal distance of a current state S is computed by an estimation function which returns the number of landmarks that still need to be achieved from the current state S. Or landmarks are combined with other heuristic techniques such as the preferred operators technique [65]. Preferred operators are operators that are believed to be useful for improving the heuristic value of the given state. In the case of landmarks, an operator is preferred in a state if it achieves landmark literals in the next state.

Furthermore, a new planner is constructed by Richter, the so-called **LAMA** [66]. LAMA is a propositional planning system built on the Fast Downward system [65]. The LAMA's search engine is tailored to use a number of heuristic estimators such as the FF heuristic [57, 67] and the landmark heuristic [62]. Hence, LAMA generates a new state by evaluating the heuristic values for all alternative states and then chooses the state that has a higher priority than other states.

A generalization of landmarks resulted in the notion of so-called *action landmarks* [62, 68, 69]. They represent landmark facts by actions that are appropriate to achieve them.

Most recent approaches use landmark information to compute heuristic functions for a forward

searching planner [62,68] and investigate their relations to critical-path-, relaxation-, and abstraction-heuristics [70].

In summary, the use of landmark information significantly improves the performance of classical state-based planners. All the above pre-processing techniques have been proposed for classical state-based planning, where they serve to compute strong search heuristics. As opposed to this, pruning the search space of the hierarchical planner by pre-processing the underlying HTN-based domain description has not been considered so far. Therefore, in the next sections we will introduce a novel landmark extraction procedure for hierarchical planning and show how landmark information can be exploited during the hierarchical planning process.

4.3 Landmarks in Hierarchical Planning

For a given hierarchical planning problem $\Pi = \langle D, s_{init}, p_{init} \rangle$, *landmarks* are the tasks that occur in every sequence of decompositions leading from the initial task network p_{init} to a solution plan. We will define landmark tasks more formally in the next subsections. Our landmark extraction algorithm operates on the so-called ***T**ask **D**ecomposition **T**ree* (TDT) of a planning problem Π.

4.3.1 Task Decomposition Tree

Figure 4.5 depicts such a tree schematically. The TDT of Π is an AND/OR tree that represents all possible ways to decompose the abstract tasks of the initial partial plan p_{init} by methods in the domain model D until a primitive level is reached or a task is encountered that is already included somewhere in the TDT. Each level of a TDT consists of two parts: a task and a method level. Method nodes are AND nodes, because their children are the set of tasks that occur in the partial plan of the respective method, all of which have to be performed in order to apply the corresponding method. Task nodes, on the other side, are OR nodes, because their children are the

methods that can be used to decompose the respective task. A TDT is built by forward chaining from the (grounded) abstract tasks in the initial task network until all nodes of the fringe are leaf nodes. The root node on level 0 is an artificial method node that represents the initial partial plan p_{init}.

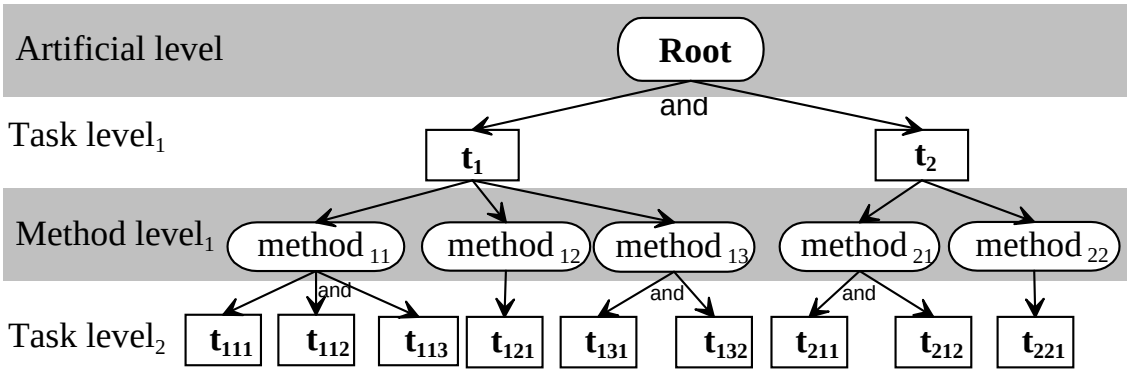


Figure 4.5 A schematic task decomposition tree.

To avoid loops or **Recursive task decompositions** in the TDT, each abstract task is decomposed only once in the TDT; hence, abstract tasks that are already decomposed in the upper level in the TDT become leaf nodes. Other leaf nodes are primitive tasks. This is obvious, because each primitive task will be executed directly without any decomposition. Note that a task $(t(\bar{\tau}))$ has a recursive decomposition if it can be decomposed or expanded into a history of tasks containing the same task $(t(\bar{\tau}))$, because the same methods that can be applied to the ancestor can be applied to the descendant.

4.3.2 Landmark Extraction Algorithm

The formal definition of landmarks in hierarchical planning (Definition 11) depends on the proper description of the paths in the search space and in particular on those refinement paths that lead from a problem specification to its solution.

Definition 10 (Plan and Solution Sequences)**.** *For a planning problem* Π *and a given plan* $P \in \mathcal{P}_\Pi$*, let* $Seq_\pi(P)$ *be the set of all sequences of plans that correspond to paths in the search space that are rooted in plan* P*. That means, given* $\mathcal{P}_\Pi = \langle \mathcal{V}, \mathcal{E} \rangle$ *let* $\langle P_1 \dots P_n \rangle \in Seq_\pi(P)$ *if and only if* $(P_i, P_{i+1}) \in \mathcal{E}$ *for all* $1 \leq i < n,\ P = P_1$*, and there is no* P' *such that* $(P_n, P') \in \mathcal{E}$*.*

The set of all solution sequences rooted in P *is then characterized by* $\mathcal{S}ol\mathcal{S}eq_\Pi(P) = \{\langle P_1 \dots P_n \rangle \in Seq_\pi(P) | P_n$ *solution of* $\Pi, n \geq 1\} \subseteq Seq_\pi(P)$*.*

Definition 11 (Landmark)**.** *A landmark is a grounded(i.e., fully instantiated) task that occurs in every sequence of decompositions leading from the problem's initial task network to a solution. That is, the task* $t(\overline{\tau})$ *is called a landmark of a planning problem* $\Pi = \langle D, s_{init}, p_{init} \rangle$*, if for every sequence of plans* $\langle P_1 \dots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(p_{init})$ *there is an* $1 \leq i \leq n$*, such that* $t(\overline{\tau}) \in Tasks(P_i)$*.*

While a landmark has to occur in every decomposition sequence of a solution (which is rooted in the initial plan), a *local* landmark only has to occur in each solution sequence rooted in a plan containing a specific task $t(\overline{\tau})$.

Definition 12 (Local Landmark of an Abstract Task)**.** *For a given grounded abstract task* $t(\overline{\tau})$ *, let* $\mathcal{P}_\Pi(t(\overline{\tau}))$ *be the set of all plans in* $\mathcal{P}_\Pi$ *containing* $t(\overline{\tau})$*, i.e.,* $\mathcal{P}_\Pi(t(\overline{\tau})) = \{P \in \mathcal{P}_\Pi | t(\overline{\tau}) \in Tasks(P)\}$

We call the grounded task $t'(\overline{\tau}')$ *a local landmark of* $t(\overline{\tau})$*, if for all* $P \in \mathcal{P}_\Pi(t(\overline{\tau}))$ *it holds, that for all sequences* $\langle P_1 \dots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_\Pi(P)$ *there is a* P_j *with* $j > 1$ *such that* $t'(\overline{\tau}') \in P_j$*.*

We use the next definition to calculate all tasks that occur in all available methods for the same abstract task $t(\overline{\tau})$.

Definition 13 (Mandatory Task Set Operator $\widehat{\cap}$)**.** *Let* $t(\overline{\tau})$ *be an abstract task in the TDT and* $m_i = \langle t(\overline{\tau}), \langle TE_i, C_i \rangle \rangle$ *with* $C_i = \langle \prec_i, VC_i, CL_i \rangle$ *and* $m_j = \langle t(\overline{\tau}), \langle TE_j, C_j \rangle \rangle$ *with* $\langle \prec_j, VC_j, CL_j \rangle$ *two of its methods in the TDT. That is, both* $t(\overline{\tau})$ *and its methods are fully grounded. Then the*

Mandatory Task Set Operator $\widehat{\cap}$ *of* m_i *and* m_j *is defined as*

$$m_i \,\widehat{\cap}\, m_j = \textit{Tasks}(TE_i) \cap \textit{Tasks}(TE_j)$$

Using this definition, we can calculate the mandatory set of an abstract task $t(\overline{\tau})$, $M(t(\overline{\tau}))$, by intersecting all available methods. Obviously, the tasks contained in $M(t(\overline{\tau}))$ are local landmarks, because these tasks are contained in all solution sequences that are rooted in a plan containing $t(\overline{\tau})$. It should also be noted, that all tasks in $M(t(\overline{\tau}))$ are local landmarks of $t(\overline{\tau})$ if $t(\overline{\tau})$ is not contained in any solution sequence[3] (i.e., if for all $\langle P_1 \dots P_n \rangle \in \mathcal{S}ol\mathcal{S}eq_{\Pi}(p_{init})$ there is no $P_i, 1 \leq i \leq n$ such that $t(\overline{\tau}) \in \textit{Tasks}(P_j)$).

However, not all local landmarks of an abstract task can be detected that way because not all local landmarks have to be in such a mandatory set.

Based on the definition of the mandatory task set operator, we will now define the Optional task set operator which calculates the set of tasks in which two (grounded) methods differ.

Definition 14 (Optional Task Set Operator $\widehat{\cup}$)**.** *Let* $t(\overline{\tau})$ *be an abstract task in the TDT and* $m_i = \langle t(\overline{\tau}), \langle TE_i, C_i \rangle \rangle$ *with* $C_i = \langle \prec_i, VC_i, CL_i \rangle$ *and* $m_j = \langle t(\overline{\tau}), \langle TE_j, C_j \rangle \rangle$ *with* $\langle \prec_j, VC_j, CL_j \rangle$ *two of its methods in the TDT. Then, the Optional Task Set Operator* $\widehat{\cup}$ *of* m_i *and* m_j *is defined as*

$$m_i \,\widehat{\cup}\, m_j = \{\textit{Tasks}(TE_i) \setminus (m_i \,\widehat{\cap}\, m_j), \textit{Tasks}(TE_j) \setminus (m_i \,\widehat{\cap}\, m_j)\}$$

Analogously to the mandatory set $M(t(\overline{\tau}))$ of an abstract task $t(\overline{\tau})$, we can define its optional tasks $O(t(\overline{\tau}))$, by applying the optional task set operator to all methods of $t(\overline{\tau})$ in the TDT. $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$ can be regarded as a partition of the methods of $t(\overline{\tau})$ in the TDT, i.e., $\{M(t(\overline{\tau})) \cup r | r \in O(t(\overline{\tau})), \text{if } O(t(\overline{\tau})) \neq \emptyset \text{ or } r = \emptyset, \text{else}\} = \{\textit{Tasks}(P) | \textit{there is a method } m = \langle t(\overline{\tau}), P \rangle \textit{ in the TDT}\}$ holds.

The landmark extraction algorithm (Algorithm 3) calculates for each abstract task that occurs in the TDT these two sets and stores it into a so-called **landmark table**. Table 4.1 shows such a

[3] In fact, *all* grounded tasks t' are local landmarks of $t(\overline{\tau})$ if $t(\overline{\tau})$ is not contained in any solution sequence.

Table 4.1 A schematic landmark table, showing a ground instance of an abstract task in each line, the mandatory set of its decompositions and the optional task sets.

abstract Tasks	**Mandatory**	**Optional**
$t_1(\bar{\tau})$	$M(t_1(\bar{\tau}))$	$O(t_1(\bar{\tau}))$
$t_2(\bar{\tau})$	$M(t_2(\bar{\tau}))$	$O(t_2(\bar{\tau}))$
$\vdots$	$\vdots$	$\vdots$
$t_n(\bar{\tau})$	$M(t_n(\bar{\tau}))$	$O(t_n(\bar{\tau}))$

landmark table schematically. The algorithm takes a TDT, which is computed before the algorithm is called, as input and returns a landmark table after its termination [71].

Intuitively, the algorithm simply tests all primitive tasks for the relaxed reachability (Algorithm 5), starting with the initial plan *(the root of the TDT)* and proceeds level by level through the TDT. If a task can be proven to be unreachable, the method introducing this task is pruned from the TDT and all its sub-nodes *(and so forth)*. After all infeasible methods of an abstract task $t(\overline{\tau})$ have been pruned from the TDT, this task, which is mandatory, and the optional tasks are stored in the landmark table.

Algorithm 3: Landmarks Extraction Algorithm

Input : A task decomposition tree TDT.

Output: The filled landmark table LT.

LT $\leftarrow \emptyset$, `infeasible` $\leftarrow \emptyset$
for $i \leftarrow 1$ **to** *TDT.maxDepth()* **do**
  foreach *abstract task* $t(\overline{\tau})$ *in level* i *of TDT* **do**
    if *LT contains an entry for* $t(\overline{\tau})$ **then continue**
    repeat
      Let M be the methods of $t(\overline{\tau})$ in the TDT.
      $M(t(\overline{\tau})) \leftarrow \widehat{\bigcap}_{m \in M} m$
      $O(t(\overline{\tau})) \leftarrow (\widehat{\bigcup}_{m \in M} m) \setminus \{\emptyset\}$
      foreach *primitive task* $t'(\overline{\tau}') \in M(t(\overline{\tau}))$ **do**
        if $t'(\overline{\tau}')$ *can be proven infeasible* **then**
          remove all $m \in M$ from the TDT, including all sub-nodes.
          break
      foreach *optional task set* $r \in O(t(\overline{\tau}))$ **do**
        foreach *primitive task* $t'(\overline{\tau}') \in r$ **do**
          if $t'(\overline{\tau}')$ *can be proven infeasible* **then**
            remove the method $m = \langle t(\overline{\tau}), P \rangle$, with *Tasks*$(P) = M(t(\overline{\tau})) \cup r$ from the TDT, including all sub-nodes.
            continue
    until *no method was removed from TDT*
    LT $\leftarrow$ LT $\cup \{(t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau})))\}$
    if $M(t(\overline{\tau})) = O(t(\overline{\tau})) = \emptyset$ **then**
      `infeasible` $\leftarrow$ `infeasible` $\cup \{t(\overline{\tau})\}$
return ***propagate****(LT,TDT,*`infeasible`*)*

Now we will have a look at how this is achieved by our algorithm: First, the landmark table and a set for backward propagation get initialized (line 1). Afterwards, each abstract task, which is not yet stored into the landmark table is considered level by level of the TDT (line 2 to 4). For the current abstract task at hand, line 6 to 8 calculate the mandatory and the optional tasks in the yet unpruned TDT according to Definition 13 and 14. In line 8, we subtract the empty set from $O(t(\overline{\tau}))$, because we are only interested in those tasks that are actually optional. If there are no optional tasks, $O(t(\overline{\tau}))$ should be empty, instead of containing an empty set. After the tasks introduced by decomposition of $t(\overline{\tau})$ have been partitioned into $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$, these sets are analyzed for infeasibility. This test is performed by a relaxed reachability analysis[4]. First we study the primitive tasks of $M(t(\overline{\tau}))$ (line 9). If such a task can be proven to be infeasible, *all methods* of $t(\overline{\tau})$ become obsolete and can hence be pruned from the TDT (line 10 and 12). After this test, each optional task set is tested for reachability. If an infeasible task can be found, only this specific method gets pruned from the TDT (line 13 to 17). If anything was pruned, the loop (line 5 to 18) enters another cycle, because the set $M(t(\overline{\tau}))$ might have grown. If no more pruning is possible, the mandatory and optional task sets for $t(\overline{\tau})$ are stored into the landmark table in line 19. When storing an entry in line 20, it is checked whether the stored abstract task is feasible or not (an abstract task is infeasible if it does not have any methods left, i.e., if $M(t(\overline{\tau}))$ and $O(t(\overline{\tau}))$ are empty). If some abstract task could actually be proven infeasible, it is stored for backward propagation, because again all methods containing this abstract task can be pruned from the TDT and from the landmark table. Finally, if all abstract tasks are checked, the backward propagation procedure is called with the current landmark table and TDT in line 22.

propagate procedure (Procedure 4) takes as input the already filled landmark table, the possibly pruned TDT, and a set `infeasible` of abstract tasks which have been proven to be infeasible

[4]We will come back to the relaxed reachability analysis in the next section

due to the fact that no methods remain in the TDT. It works tail-recursively and returns the final landmark table as soon as no propagation is possible (line 1). To this end, it first takes and removes some arbitrary task $t'(\overline{\tau}')$ from the set `infeasible`. Because this abstract task was proven infeasible, all methods that contain it have to be removed from the TDT. As a consequence of this pruning, the mandatory and optional task sets have to be updated; additionally, further propagation is now possible. To calculate the methods that can possibly be pruned, all parent tasks of $t'(\overline{\tau}')$ are identified (line 3). Then, for all these parents (line 4), the respective methods are removed in line 5. Because methods were removed, the mandatory and the optional task sets could have changed again. Hence, they are recalculated in line 6 to 8. Next, the the old landmark table entry of the current parent $t(\overline{\tau})$ is removed and replaced by the new one (line 9). In line 10, the new landmark table entry is tested again whether it corresponds to an infeasible abstract task. If it does, it is put into the set `infeasible` for later testing. The procedure is then called with the modified parameters in line 1. Without a formal proof, we want to mention that algorithm 3 (i.e., the landmark table calculation as well as the backward propagation) always terminates. For the first part of the algorithm, this is easy to see because both loop conditions (line 2 and 3) cannot be modified within the loops. For the second part, i.e., the propagate procedure, we have to show that the set `infeasible` eventually becomes empty. This is the case because each task gets inserted at most once and will be removed at some point. After the algorithm terminates, the TDT does not have to be considered anymore. All necessary information is encoded in the landmark table.

Procedure `propagate(`*LT,TDT,*`infeasible)`

Input : A landmark table LT, a task decomposition tree TDT, possibly pruned, and a set of abstract tasks `infeasible`, which have been proved infeasible.

Output: the updated landmark table LT, in which methods are pruned that contain infeasible abstract tasks.

if $infeasible = \emptyset$ **then return** *LT*

infeasible $\leftarrow$ infeasible $\setminus \{t'(\overline{\tau}')\}$, where $t' \in$ infeasible.

parents $\leftarrow \{t(\overline{\tau}) | (t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))) \in LT, t'(\overline{\tau}') \in M(t(\overline{\tau})) \cup \bigcup_{r \in O(t(\overline{\tau}))} r\}$

foreach $t(\overline{\tau}) \in parents$ **do**

  Remove all methods from the TDT, that contain $t'(\overline{\tau}')$ in its plan, i.e., all $m = \langle t(\overline{\tau}), P \rangle$ with $t'(\overline{\tau}') \in \mathit{Tasks}(P)$.

  Let M be the methods of $t(\overline{\tau})$ in the TDT.

  $M(t(\overline{\tau})) \leftarrow \widehat{\bigcap}_{m \in M} m$

  $O(t(\overline{\tau})) \leftarrow (\widehat{\bigcup}_{m \in M} m) \setminus \{\emptyset\}$

  LT $\leftarrow$ (LT$\setminus\{(t(\overline{\tau}), M'(t(\overline{\tau})), O'(t(\overline{\tau}))) \in$ LT$\}) \cup \{(t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau})))\}$

  if $M(t(\overline{\tau})) = O(t(\overline{\tau})) = \emptyset$ **then**

    infeasible $\leftarrow$ infeasible $\cup \{t(\overline{\tau})\}$

return ***propagate****(LT,TDT,*`infeasible`*)*

4.3.3 Relaxed Reachability Analysis

A great challenge in our algorithm is the difficulty of deciding the reachability of a primitive task. We addressed a way to solve this problem. However, the pre-conditions of any primitive task are a set of literals. These literals may be rigid or flexible. In order to decide the reachability value, the literals in the pre-conditions of the primitive task must be examined.

Algorithm 5: *Relaxed reachability analysis Algorithm*

Input : T: Primitive Task, s_{init}: Initial State, LT: LandmarkTable

Output: Boolean value(True or False)

```
1  Flag ⟵ false
2  while (pre-condition of T contains rigid literals ℓ) do
3      if (predicate_symbol(ℓ) ∉ s_init) then
4          return false
5      else
6          Let comp is a result of comparison between the argument type in ℓ and argument type in the
           corresponding ground atom in s_init.
7          if (comp==true) then
8              Flag ⟵ true
9              continue
10         else
11             return false
12 while (pre-condition of T contains flexible literals ℓ) do
13     if (pre-condition of T has a negative form) then
14         return true
15     else if (T ∈ consumer-causallink) then
16         Find its producer task Tp.
17         if (Tp ∈ LT) then
18             return true
19         else
20             return false
21     else
22         return true
23 if (flag==true) then
24     return true
```

Rigid Literals

Rigid literals are those literals whose truth value can't be changed by any primitive task. Therefore, this kind of literals must be achieved in the initial state. If at least one rigid literal of a pre-condition of a primitive task is not achieved in the initial state, this primitive task cannot appear in any solution plan. As we see in algorithm 5, we can't rely on the predicate symbol of the literal alone when checking whether the pre-condition in question actually holds. Instead we have to check for type and variable constraints in addition in order to find out whether a ground atom of the initial state is compatible with the required pre-condition (lines 2 to 11).

For example, in our representation, a fact is described by the formula $R(arg_1{:}sort_1, \cdots, arg_n{:}sort_n)$ where R is a predicate symbol and $arg_1, arg_2, \cdots, arg_n$ represent the parameter values and $sort_1, sort_2, \cdots, sort_n$ denote the types of the respective arguments. Tasks are defined through task schemata $t(\widetilde{\tau}) = (prec(t(\widetilde{\tau})), add(t(\widetilde{\tau})), del(t(\widetilde{\tau})))$ where the task t belongs to the set of tasks T and $\widetilde{\tau} = \tau_1, \tau_2, \cdots, \tau_n$ is a parameter list, each τ_i being a constant or variable for $1 \leq i \leq n$ and n being the number of arguments of the task t. The $prec(t(\widetilde{\tau}))$ represents the pre-conditions of task t which must be true before t can be executed, $add(t(\widetilde{\tau}))$ and $del(t(\widetilde{\tau}))$ represent the effects the execution of t will have. Suppose, the fact $In\_City(L_1 : Customer\_Location, C_1 : City)$ holds in an initial state and we have the following primitive task: $go\_through\_two\_tcenters(var_1 : Tcenter, var_2 : Tcenter, var_3 : City, var_4 : City)$ with pre-conditions $In\_City(var_1, var_3)$ and $In\_City(var_2, var_4)$. If the predicate symbol ***In_City*** of the literal in the pre-condition of the primitive task is only compared to the ground atom in the initial state then this literal is accepted. This would be wrong, however, because argument L_1 is of the sort *a Customer Location* and not *Tcenter* as required in the pre-condition of the primitive task. In contrast, if we also compare the argument types then this task will be rejected, because the sort of L_1 is *Customer Location* and not *Tcenter*. But if we have the ground atom $In\_City(A_1 : Airport, C_1 : City)$ in the initial state, the previous literal $In\_City(var_1, var_3)$ will be accepted, because the predicate symbol In_City

exists in the initial state and if we compare its sort we find *Airport* is a *Tcenter* and C_1 is a *City* and therefore this literal is accepted.

Flexible Literals

Flexible literals are those literals whose truth values of which can be changed by basic actions, *i.e. by primitive tasks*. Algorithm 5 shows how to check feasibility of a primitive task which has flexible literals in its pre-condition. It implements a heuristic estimation of reachability rather than a decision procedure in order to keep the pre-processing effort of our approach manageable. The reachability analysis algorithm handles flexible literals as follows: If a pre-condition literal of the task has a negative form, feasibility is assumed (lines 13 and 14). This is motivated by the fact that we adopt the closed world assumption, *i.e. every literal that does not appear in the initial state is considered false*. At this point we could not decide exactly whether the truth value of this literal will be changed by the plan or not. Otherwise, if a primitive task appears as a consumer task in a causal link of the pre-condition literal, we will search for its producer in the LT. If the producer task appeared previously in the LT, feasibility is assumed (lines 15 to 20). For example, suppose the $method\_helper\_move\_traincar$ contains two tasks in its subtasks: the first task is the abstract task $move$ and the other is the primitive task $attach\_traincar$. Furthermore, there is a causal link stating that the $move$ task produces a literal $at\_vehicle(var_1 : Vehicle, var_2 : Location)$ for the primitive consumer task $attach\_Traincar$. Therefore, if the $move$ task has previously been added to LT, we conclude that the primitive task $attach\_Traincar$ is feasible. In contrast, if the current primitive task does not appear as a consumer task in any causal links then the pre-condition literal may depend on some other task in some other method. In this case, this primitive task may be achieved during the planning process and it is considered feasible (line 22). For example, in the *method_carry_between_tcenters_carry_direct* there are two sub-tasks: the abstract task *carry_direct* and the primitive task *go_through_two_tcenters*, where the primitive task has a flexible literal in its

pre-conditions and does not appear as a consumer task in a causal link.

4.3.4 Example

In order to illustrate our landmark extraction technique, let us consider a simple example in the UM-Translog domain [42]. Assume a package P_1 is at location L_1 in the initial state and we would like to transport it to a customer location L_3 in the same city. Figure 4.6 shows a part of the task decomposition tree for this example.

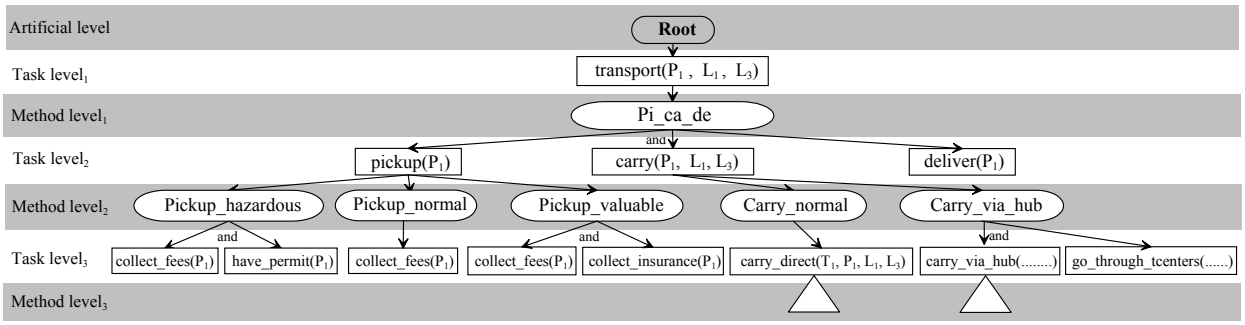


Figure 4.6 Part of the TDT for the transportation task

The landmark extraction algorithm detects that the first level in the TDT contains only one abstract task t = *transport(*P_1,L_1,L_3*)* and that there is only one method, *Pi_ca_de*, that can decompose the task into a partial plan and which contains the subtasks *pickup(*P_1*), carry(*P_1, L_1, L_3*)*, and *deliver(*P_1*)*. $M(t)$ becomes {*pickup(*P_1*),carry(*P_1, L_1, L_3*),deliver(*P_1*)*} and $O(t) = \emptyset$. The current abstract task and the sets $M(t)$ and $O(t)$ are entered into the first row of the landmark table as shown in table 4.2.

The landmark extraction algorithm then takes the (unchanged) TDT to investigate the next tree level. The abstract tasks to be inspected on this level are *pickup(*P_1*) and carry(*P_1, L_1, L_3*)*. The primitive task *deliver(*P_1*)* is tested and considered executable. Suppose, the task t = *pickup(*P_1*)* is chosen first in line 3 of algorithm 3. As shown in figure 4.6, the TDT accounts for three methods

Table 4.2 Example landmark table: It contains the first three entries for the transportation task illustrated in figure 4.6. The sets in the right most column are indexed by the method's name that contains its tasks.

abstract Task	Mandatory	Optional
transport(P_1, L_1, L_3)	{pickup(P_1),carry(P_1, L_1, L_3),deliver(P_1)}	$\emptyset$
pickup(P_1)	{collect_fees(P_1)}	$\{\{\text{have\_permit}(P_1)\}_{\text{Pickup\_hazardous}}\}$
carry(P_1, L_1, L_3)	{{carry_direct(T_1, P_1, L_1, L_3)}}	$\emptyset$

to decompose this task: *Pickup_hazardous*, *Pickup_normal*, and *Pickup_valuable*. By computing the mandatory task set and the optional task sets we get $M(t) = \{$*collect_fees(*P_1*)*$\}$, and $O(t) = \{\{$*have_permit(*P_1*)*$\}, \{$*collect_insurance(*P_1*)*$\}\}$. At this point, the relaxed reachability analysis is performed. First, *collect_fees(*P_1*)* is being tested, because it is contained in the mandatory set $M(t)$. Suppose, this task can not be proven to be infeasible. Then each primitive task in each set $r \in O(t)$ has to be checked. Assume the primitive task *have_permit(*P_1*)* is feasible, whereas *collect_insurance(*P_1*)* is not. The method *Pickup_valuable* is therefore deleted from the TDT. After an additional iteration in which $M(t)$ and $O(t)$ get recalculated, the current abstract task t = *pickup(*P_1*)*, the set $M(t)$, and the modified set $O(t)$ are added to the landmark table as depicted in the second line of table 4.2. From the fact that $O(t)$ contains only one set r we can conclude that there is another method with no remaining tasks (if there were no such method, the tasks of $r \in O(t)$ would be contained in $M(t)$.

In the second iteration (line 3) the abstract task t = *carry(*P_1, L_1, L_3*)* is considered. The methods *Carry_normal* and *Carry_via_hub* are available to decompose this task. We obtain $M(t) = \emptyset$ and $O(t) = \{\{$*carry_direct(*T_1, P_1, L_1, L_3*)*$\}, \{$*carry_via_hub(*$\dots$*), go_through_tcenters(*$\dots$*)*$\}\}$. Suppose the primitive task *go_through_tcenters(*$\dots$*)* is infeasible. The sub tree with root *carry_via_hub(*$\dots$*)*

has then to be removed from the TDT. Because the TDT was changed, the iteration (line 5 to 18) enters another cycle. Because there is now only one method left, $M(t)$ now contains all tasks of this remaining method. Hence, the current abstract task t = *carry*(P_1, L_1, L_3) together with the modified $M(t)$ and $O(t)$ are added to the landmark table as shown in the last line of table 4.2.

4.4 Landmark Exploitation

The information about landmarks can be exploited in two ways: The first is to deduce heuristic guidance from the knowledge about which tasks have to be decomposed on refinement paths that lead towards a solution. But before we further investigate this matter, we will present a second way of landmark exploitation, namely the reduction of domain models or, more precisely, the transformation of a universal domain model into one that includes problem-specific pruning information.

4.4.1 Domain Model Reduction

During the construction of the landmark table, the feasibility check and the consecutive propagation of its result into the abstract task level lead to pruning of the task decomposition tree. The result of this analysis implies that if a method has been removed from the TDT during the operation of our landmark extraction algorithm, it can be safely ignored as a refinement option during plan generation.

We consequently supply our refinement generating module with the landmark table for the current planning problem and verify for every incoming abstract task flaw which of the methods specified in the domain model are reasonably applicable. However, the landmark table is built from grounded tasks, while the plan generation procedure operates on lifted instances for which the final grounding is yet to be computed. We therefore calculate all groundings of the abstract task at hand that are consistent with the current variable constraints and match these grounded

tasks $t(\bar{\tau})$ with the entries in the landmark table. The union of the (lifted) method schemata that constitute the (grounded) instances in the Optional Task Sets $O(t(\bar{\tau}))$ is the set of method schemata that we consider for application to the currently flawed abstract task. Obviously, the earlier a task is addressed in the planning process, the less variable constraints are typically introduced in the partial plan, and the more task groundings are implied by the lifted instance, and, consequently, the less likely it is that one of its methods is pruned by this technique. To this end, our approach is fully implemented using the adopted framework in chapter 3.

Based on flaw detection, modification module and search strategies f^{ModSel} and $f^{PlanSel}$ definitions, algorithm 21[5] sketches a generic hybrid planning algorithm. The procedure is initially called with the partial plan P_{init} of a planning problem Π as a unary list of plans and with the problem itself. This list of plans represents the current plan development options in the fringe of the search space. An empty fringe ($n = 0$) means, that no more plan refinements are available. Lines 5-8 call the detection functions to collect the flaws in the current plan P_{current}. If P_{current} is found to be flawless, it constitutes a solution to Π and is returned. If not, lines 9-16 organize the flaws class-wise and pass them to the α-assigned modification generation functions, which produce plan modifications that will eliminate the flaws. Any flaw that is found unsolvable will persist and P_{current} is hence discarded [46]. The plan selection strategy $f^{planSel}$ is responsible for choosing a plan from the fringe with which to continue planning.

If appropriate refinements have been found for all flaws, the modification selection function f^{modSel} is called in line 17. Based on the current plan and its flaws, it selects and prioritizes those plan modifications that are to be used for generating the refinements of the current plan. The chosen modifications are applied to P_{current} and the produced successor plans are inserted into the search space fringe. The algorithm is finally called recursively on an updated fringe in which the

[5] It requires the sets of flaw detection and modification generation modules $\mathfrak{Det}$ and $\mathfrak{Mod}$, the strategies f^{modSel} and $f^{planSel}$

strategy function $f^{planSel}$ determines the next focal plan.

Note that the algorithm allows for a broad variety of planning strategies [7, 8], because the planning procedure is completely independent from the flaw detection and the modification generating function.

Since our approach is based on a *declarative* model of task abstraction, the exploitation of knowledge about hierarchical landmarks can be done *transparently* during generation of task expansion modifications: First, the respective modification generation function $f_y^{mod}$is deployed with a reference to the landmark table of the planning problem, which has been constructed off-line in a pre-processing phase. During planning, each time an abstract task flaw indicates an abstract plan step $t(\bar{\tau})$ the function f_y^{mod}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 landmark table.

It is important to see that the overall plan generation 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). In principle, non-declarative hierarchical planners, like the SHOP family [72] can also profit from our landmark technique. The benefit will however be reduced due to the typically extensive usage of method application conditions, which cannot be analyzed during task reachability analysis, in particular if the modeller relies on side effects of the method processing.

Algorithm 6: *Refinement*$(P_1 \dots P_n, \Pi)$

Input : $P_1 \dots P_n$: Sequence of Plans, $\Pi = \langle D, S_0, p_{init} \rangle$: Planning Problem

Output: Plan or failure

begin
  if $n = 0$ **then**
    return failure
  $P_{\text{current}} \leftarrow P_1$; Fringe $\leftarrow P_2 \dots P_n$; $F \leftarrow \emptyset$
  forall $f_x^{det} \in \mathfrak{Det}$ **do**
    $F \leftarrow F \cup f_x^{det}(P_{\text{current}}, \Pi)$
  if $F = \emptyset$ **then**
    return P_{current}
  $M \leftarrow \emptyset$
  forall $F_x = F \cap \mathtt{F_x}$ *with* $F_x \neq \emptyset$ **do**
    forall $\mathtt{f} \in F_x$ **do**
      forall $f_y^{mod} \in \mathfrak{Mod}$ *with* $\mathtt{M_y} \subseteq \alpha(\mathtt{F_x})$ **do**
        $M \leftarrow M \cup f_y^{mod}(P_{\text{current}}, \mathtt{f}, D)$
    if `f` *was un-addressed* **then**
      $P_{\text{next}} \leftarrow f^{planSel}(\text{Fringe})$
      return Refinement$(P_{\text{next}} \circ (\text{Fringe} - P_{\text{next}}), \Pi)$
  forall $\mathtt{m} \in f^{modSel}(P_{\text{current}}, F, M)$ **do**
    Fringe $\leftarrow$ *apply*$(\mathtt{m}, P_{current}) \circ$ Fringe
  $P_{\text{next}} \leftarrow f^{planSel}(\text{Fringe})$
  return Refinement$(P_{\text{next}} \circ (\text{Fringe} - P_{\text{next}}), \Pi)$
end

4.5 Experimental and Empirical Analysis

This section presents experiments that evaluate the mechanism of our landmark pre-processing technique in context of hierarchical planning approaches. The objective of our evaluation is to measure the increase in planning efficiency gained through our technique. To this end, we measure the **efficiency** factor. The efficiency is defined as a ratio of obtained solution quality to the required efforts. The common efficiency metrics are:

- **Search Space Size (SSS)** – is the total number of plans visited for obtaining the first solution.
- **Planning Time (CPU-Time)** – is the total running time of the planning system in seconds.

All experiments were run on a machine with 3 GHz CPU and 256 MB Heap memory for the Java VM. Before analyzing our results in sections 4.5.4 and 4.5.5, we will illustrate the hierarchical planning domains and problem instances which are used in our experiment.

4.5.1 Experimental Domains

As sophistication and capabilities of planning systems increase, planning domains with matching complexity need to be devised to assist in the analysis and evaluation of planning systems and techniques. Therefore, we use two different domains in our experiments: *UM-Translog* and *Satellite*.

UM-Translog Domain

UM-Translog domain was introduced by Andrews et al. [42]. It is used to evaluate hierarchical task network planning systems [73–76].

The *UM-Translog* domain has also been adapted for hybrid planning by Schattenberg [46] in order to evaluate his PANDA planner [7].

In general, the *UM-Translog* domain illustrates different scenarios of the transportation and logistics process. Actually, it describes the ways in which different kinds of transportation goods are delivered to specific locations by different types of vehicles *(trucks, train,$\cdots$,etc.)*, through appropriate infrastructure *(roads, railroad lines, transport centers,$\cdots$, etc)*. In this domain we can examine a considerable number of different tasks such as loading goods, finding a path through the infrastructure, and finally delivering specific goods such as hazardous and valuable packages.

Entities and Relations The *UM-Translog* domain provides a lot of entities or sorts *(Location, Package, Rout and vehicle)* and relations (or predicates) as well as tasks and methods which can be used to specify rather complex planning problems with a variety of plan interactions. Each entity is organized as a sort hierarchy as follows:

- **Location Sorts**

  As shown graphically in figure 4.7, location sorts are organized in a sort hierarchy. For example, *city_location* is the super sort of sub-sorts transport center **(TCenter)** and not a transport center **(Not_TCenter)**. The sort location TCenter includes sub-sorts **(Airport)**, **(Hub)** and **(Train_Station)**. In general, a transportation center TCenter can be used for indirect air/rail transportation. In contrast, Not_TCenter is a super sort for sub-sorts **(Post_Office)** and **(Customer_Location)**.

  The relations between different locations are expressed by predicates. For example, the **In_Region** relation is a rigid predicate. It determines the relation between city and region *(i.e., which city is located in a specific region)*. Note that a region can contain one or more cities, while a city can include one or more city locations which are specified by rigid relation **In_city**. The rigid predicate **hub** is used to connect a transport center with an airport or train_station whenever this transport center serves specific cities via the **Serves** relation.

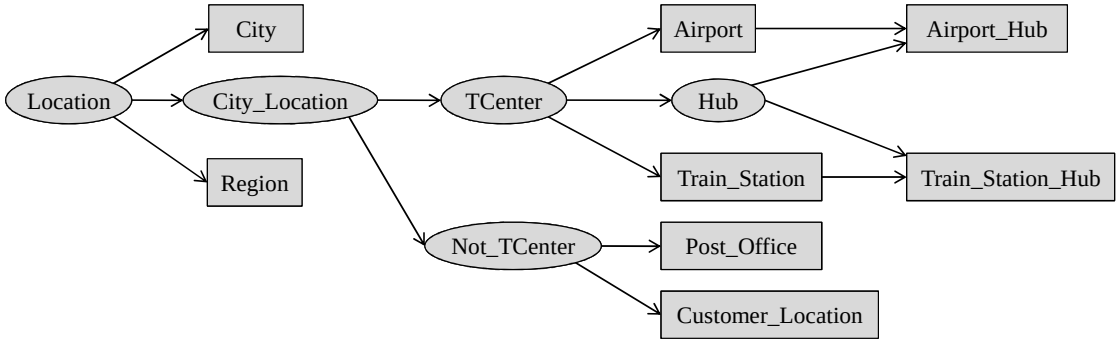


Figure 4.7 UM-Translog domain: Location sort hierarchy – ellipse shape nodes represent abstract sorts, while box shape nodes represent concrete sorts *(sorts for which constants can be provided)*.

- **Route Sorts**

  Figure 4.8 shows the relationships between different routes. In general, all locations are assumed to be connected by routes. For example, the sub-sorts of route are **Rail_Route**, **Road_Route** and **Air_Route**, while the **Local_Road_Route** is a sub-sort of **Road_Route**. The rigid predicate **Connect** provides a link between two routes as the source and destination route respectively. In order for a specific route to be usable, it should be available by the flexible predicate **Available**.

- **Vehicle Sorts**

  Figure 4.9 shows the different kinds of vehicles. For example, the **Truck** vehicle sort has four sub-sorts **Regular_Truck**, **Auto_Truck**, **Armored_Regular_Truck** and **Mail_Truck**. Compatibility between routes and the kind of vehicle is determined via the rigid relation **RV_Compatible**, while the flexible relation **At_Vehicle** determines the location of vehicle.

- **Package Sorts**

  The *UM-Translog* domain has a number of package types such as **Liquid, Valuable, Haz-**

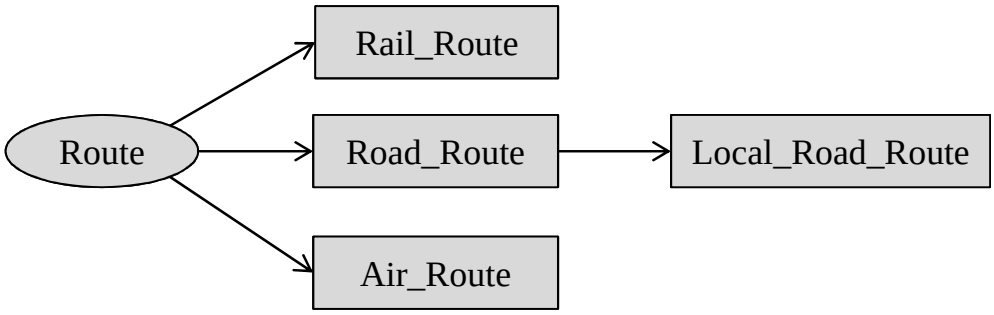


Figure 4.8 UM-Translog domain: Route sort hierarchy – ellipse shape nodes represent abstract sorts, while box shape nodes represent concrete sorts.

ardous, $\cdots$, etc. For example the sub-sorts of a **Perishable** package are **Food** and **Chemicals** (see Figure 4.10). The flexible relation **At_Package** determines the location of package, while the rigid relation **PV_Compatible** determines the compatibility between package and vehicle types.

In general, the sort hierarchy in our *UM-Translog* domain model is too large to display here. It includes about 96 sorts. Therefore, our figures 4.7, 4.8, 4.9 and 4.10 only represent parts of the *UM-Translog* sort hierarchy.

Tasks and Methods The *UM-Translog* domain has a deep expansion hierarchy in 51 methods for decomposing 21 abstract tasks into 51 primitive tasks. We will illustrate some parts of this hierarchy in more detail. The most abstract task in this domain is

$$Transport(?p : Package,\ ?s : Location,\ ?d : Location) =$$
$$\langle\{At\_Package(?p, ?s)\}, \{\neg At\_Package(?p, ?s), At\_Package(?p, ?d)\}\}\rangle$$

The semantic of *transport* task is that the package $?p$ is to be transported from one location $?s$ to another location $?d$. Of course the *transport* task is an abstract task and can not be performed directly. We will introduce only one refinement through *transport method*, the so-called *transport_pi_ca_de*.

$m : transport\_pi\_ci\_de = \langle transport(?p,\ ?s,\ ?d), \langle\{\ell_1 : Pickup(?\ell_1.p),$

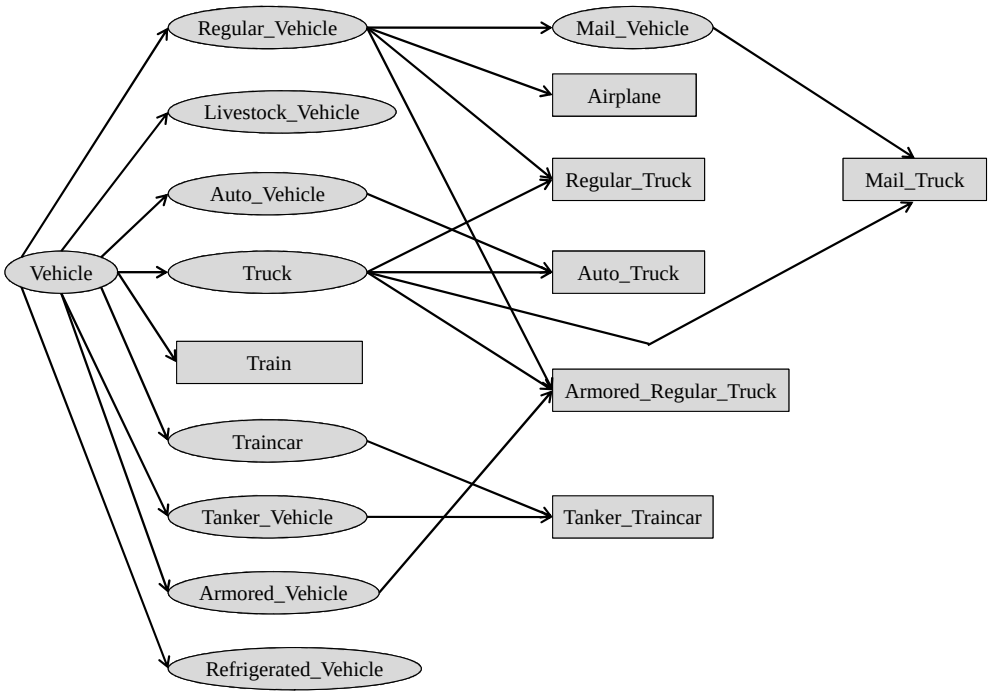


Figure 4.9 UM-Translog domain: Vehicle sort hierarchy – ellipse shape nodes represent abstract sorts, while box shape nodes represent concrete sorts.

$$\ell_2 : Carry(?\ell_2.p, ?\ell_2.s, ?\ell_2.d), \ell_3 : Deliver(?\ell_3.p)\},$$
$$\{\ell_1 \prec \ell_2,\ \ell_2 \prec \ell_3\},$$
$$\{?p = ?\ell_1.p, ?p = ?\ell_2.p, ?p = ?\ell_3.p, ?s = ?\ell_2.s, ?d = ?\ell_2.d, ?s \neq ?d\}$$
$$\{\langle \ell_1, Fees\_Collected(?\ell_1.p), \ell_3\rangle\}\rangle\rangle$$

The *transport task* is decomposed into a task network with three sub-tasks: the *Pick_up* task with which to pick up the package from the source location. After that the package should be moved *(Carrying task)* to the destination location and finally be delivered to its destination by *Deliver* task. The corresponding constraints which exist in the current plan are modified according to the variable, Ordering and causal link constraints in the method. The *Pick_up*, *Carry* and *Deliver* tasks are defined as follows:

$Pick\_up(?p : Package)$ = $\langle\{\neg Fees\_Collected(?p)\}, \{Fees\_Collected(?p)\}\rangle$

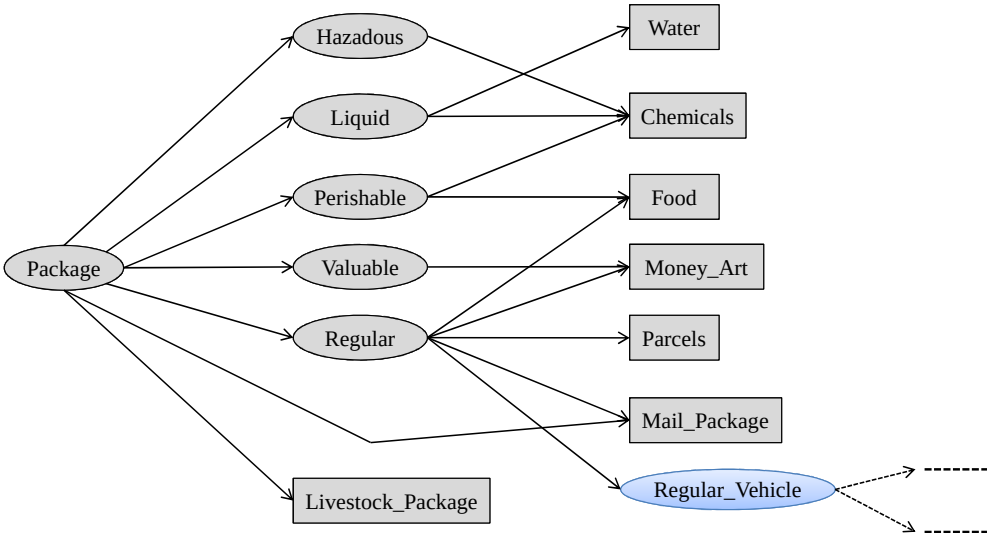


Figure 4.10 UM-Translog domain: Package sort hierarchy – ellipse shape nodes represent abstract sorts, while box shape nodes represent concrete sorts.

$$Carry(?p, ?s, ?d) = \langle\{At\_Package(?p, ?s)\}, \{\neg At\_Package(?p, ?s), At\_Package(?p, ?d)\}\rangle$$

$$Deliver(?p : Package) = \langle\{Fees\_Collected(?p)\}, \{Delivered(?p)\}\rangle$$

The methods that refine the *Pick_up* and *Deliver* tasks are categorized into the context for normal, hazardous and valuable transport goods. That means, the fees should be collected in the normal decomposition, while in the hazardous and valuable decompositions, a transit permission has to be obtained or an insurance has to be covered. The corresponding delivery task requires these facts as their pre-conditions and thereby ensure that the package is registered or insured during the entire transportation process. The decomposition of the abstract task *Carry* involves choosing a suitable path *i.e., a sequence of routes from source to destination*, and moving the package along that path by applying a series of tasks. The remaining part of *Tasks and Methods* in the *UM-Translog* domain covers the different alternatives to refine new abstract tasks which are added from previous refinements.

Satellite Domain

The satellite domain was introduced in IPC[6]-3 by Long et al. [77]. It was motivated by a NASA space application. The *Satellite* domain is an established benchmark in the field of non-hierarchical planning. Its sorts and decomposition structure were adapted to hybrid planning [46]. It handles the problem of managing scientific stellar observation carried out earth orbiting instrument platforms. The equipment that is used to make observations consists of observation instruments each with different characteristics in terms of data production, the so-called *modes e.g. x-ray mode*, and appropriate calibration targets. Satellites can be pointed at different targets by moving them into different directions.
The hierarchical *Satellite* domain consists of 6 sorts, 8 relations, 3 abstract tasks, 5 primitive tasks, and 8 method declarations. We will list some of these components below.

Entities and Relations The *Satellite* domain comprises a set of sorts such as satellites, directions, instruments, and modes. The flexible relation **Pointing** is used for expressing the satellite direction, while the rigid relation **On_Board** determines which instruments the satellite is carrying. The type of sensors an instrument provides is determined by the rigid relation **Supports**. The **Power_Avail** and **Power_On** are two flexible relations reflecting that the energy is a limited resource on observation platforms. An instrument has to be switched off before another instrument can be activated. Finally, the relation **Have_Image** is supposed to hold in a state in which an image of a specified phenomenon is taken.

Tasks and Methods The main task in the *Satellite* domain is the process of taking an image. With specifications for sensor mode and phenomenon, this task has to cover the follows objectives:

1. Choosing a suitable instrument which indirectly determines the satellite that performs the

[6]IPC $\Longrightarrow$ International Planning Competition

Table 4.3 Satellite domain: Abstract tasks.

Task Arguments	Task
Task Name	$do\_observation(?d, ?m)$
Parameters	$?d$: **image_direction**, $?m$: **mode**
Pre-condition	true
effects	$have\_image(?d, ?m)$
Task Name	$activate\_instrument(?s, ?i)$
Parameters	$?s$: **satellite**, $?i$: **instrument**
Pre-condition	$on\_board(?i, ?s)$
effects	$power\_on(?i)$
Task Name	$auto\_calibrate(?s, ?i)$
Parameters	$?s$: **satellite**, $?i$: **instrument**
Pre-condition	$on\_board(?i, ?s)$, $power\_on(?i)$
effects	$calibrated(?i)$

main task.

2. The instrument has to be routed towards the desired direction and then calibrated.

3. The satellite has to be turned towards the direction of the target phenomenon and has to catch an image.

In order to perform the above points, some of the tasks in tables 4.3 and 4.4 are applied. The most abstract task in the *Satellite* domain is **do_observation** with arguments for the required phenomenon to observe and the mode to support. Therefore, the instrument has to prepared by activating the suitable instrument, then the satellite has to be turned in the desired direction and finally

the image has to be taken. Figure 4.11 shows all the possible ways to solve the main abstract task in the satellite domain which called $do\_observation$.

$$m : do\_observation\_ai\_tt\_ti = \langle do\_observation(?d,\ ?m), \langle \{\ell_1 : activate\_instrument(?\ell_1.s, ?\ell_1.i),$$
$$\ell_2 : turn\_to(?\ell_2.s, ?\ell_2.to, ?\ell_2.from),$$
$$\ell_3 : take\_image(?\ell_3.s, ?\ell_3.dir, ?\ell_3.i, ?\ell_3.m)\},$$
$$\{\ell_1 \prec \ell_2,\ \ell_2 \prec \ell_3\},$$
$$\{?d = ?\ell_1.to, ?d = ?\ell_3.dir, ?m = ?\ell_3.m, ?\ell_1.s = ?\ell_2.s, ?\ell_1.s = ?\ell_3.s,$$
$$?\ell_1.i = ?\ell_2.s, ?\ell_1.i = ?\ell_3.s, ?\ell_2.to \neq ?\ell_2.from, ?\ell_2.to \in image\_direction\},$$
$$\{\langle \ell_1, power\_on(?\ell_1.i), \ell_3\rangle, \langle \ell_2, pointing(?\ell_2.s, ?\ell_2.to), \ell_3\rangle\}\rangle\rangle$$

If, however, the instrument is already achieved and it still is calibrated, the satellite just has to move in the desired direction and take the image. If the satellite is already pointing at the desired direction, then the **do_observation** task needs to prepare the instrument and then take the image. The process of preparing the instrument is performed by two different methods *m:activate_instrument_soff_son_ac* and *m:activate_instrument_son_ac*. Through these methods the satellite can encapsulate the different ways of getting system energy (see Figure 4.11).

4.5.2 Planning Problems

In this section, we briefly discuss the structure and solution properties of each planning problem in the *UM-Translog* domain as well as in the *Satellite* domain.

Planning Problems in the UM-Translog Domain

UM-Translog problems 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 the transportation of regular packages in trucks. Therefore, we conducted our experiments on qualitatively different problems by specifying various transportation means and goods without

Table 4.4 Satellite domain: Primitive tasks.

Task Arguments	Task
Task Name	$turn\_to(?s, ?d\_new, ?d\_prev)$
Parameters	$?s$: **satellite**, $?d\_new$: **direction**, $?d\_prev$:**direction**
Pre-condition	$pointing(?s, ?d\_prev)$
effects	$\neg pointing(?s, ?d\_prev), pointing(?s, ?d\_new)$
Task Name	$switch\_on(?i, ?s)$
Parameters	$?i$: **instrument**, $?s$: **satellite**
Pre-condition	$on\_board(?i, ?s)$, $power\_avail(?s)$
effects	$power\_on(?i)$, $calibrated(?i)$, $\neg power\_avail(?s)$
Task Name	$switch\_off(?i, ?s)$
Parameters	$?i$: **instrument**, $?s$: **satellite**
Pre-condition	$on\_board(?i, ?s)$, $power\_on(?i)$
effects	$\neg power\_on(?i)$, $power\_avail(?s)$
Task Name	$calibrate(?s, ?i, ?cd)$
Parameters	$?s$: **satellite**, $?i$: **instrument**, $?cd$: **calib_direction**
Pre-condition	$on\_board(?i, ?s)$, $calibration\_target(?i, ?cd)$, $pointing(?s, ?cd)$, $power\_on(?i)$
effects	$calibrated(?i)$
Task Name	$take\_image(?s, ?d, ?i, ?m)$
Parameters	$?s$:**satellite**, $?d$:**image_direction**, $?i$:**instrument**, $?m$:**mode**
Pre-condition	$calibrated(?i)$, $pointing(?s, ?d)$, $on\_board(?i, ?s)$, $power\_on(?i)$, $supports(?i, ?m)$
effects	$have\_image(?d, ?m)$

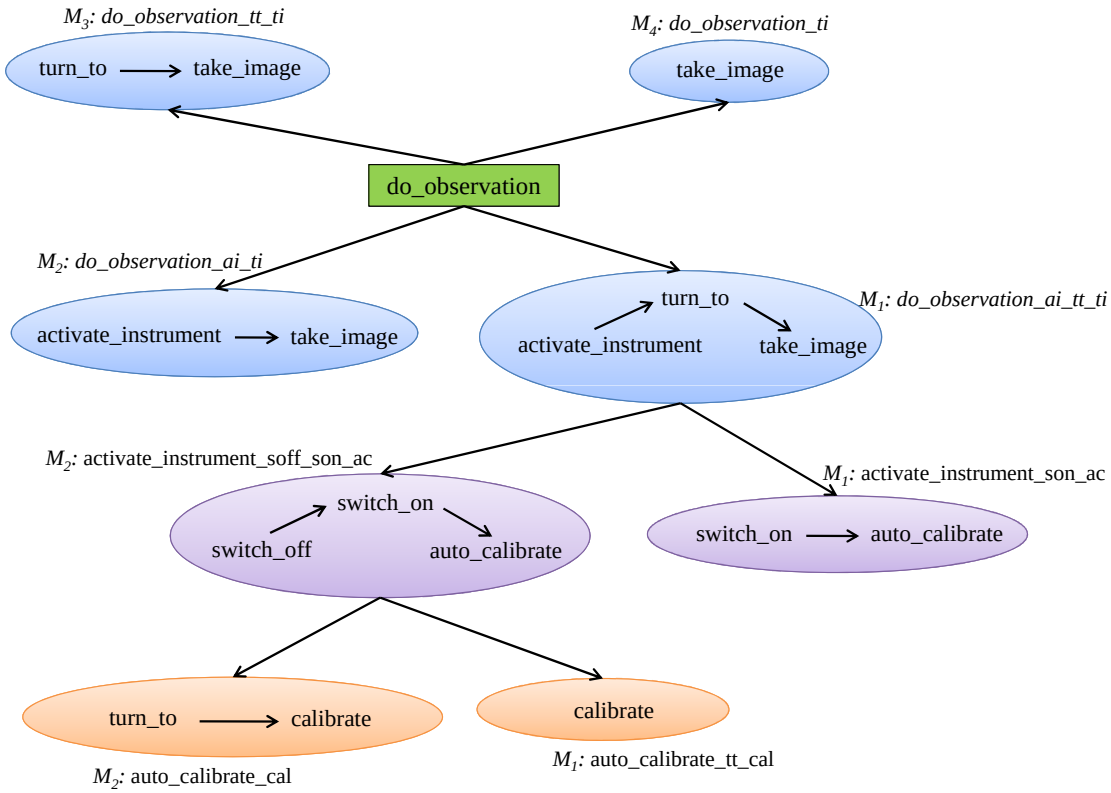


Figure 4.11 The Satellite domain: Different levels of the decompostion of the abstract task do_observation.

further constraints such as limited transportation means, unavailability of transportation routes, etc.

- **RegularTruck:** This is the problem of transporting a normal package such as parcels from one city to another.

- **RegularTruck-3Location:** This is an instance of the RegularTruck problem, but the decomposition methods are used to perform further route planning over an intermediate locations in the city.

- **RegularTruck-2:** The RegularTruck-2 problem is an instance of RegularTruck transporting two different packages.

- **HopperTruck:** The hopper truck is the problem of transporting a special kind of package such as sand. Transportation requires a special kind of truck containing container. This truck is loaded and unloaded via a chute that has to be connected to the truck during loading and unloading.
- **FlatbedTruck:** This kind of truck is used to transport a special package such as lumber. Flatbed truck requires a fixed equipment *"crane"* for loading and unloading the required package.
- **AutoTruck:** This is a special kind of truck attached by a trailer to transport cars.
- **ArmoredRegularTruck:** Valuable packages such as money or valuable art is transported by a special truck which is required to be an armored transportation vehicle being under surveillance of special guards during loading and unloading.
- **TankerTruck:** This kind of truck is used to transport hazardous packages. A permission is required for transporting this kind of package. In addition, warning signs have to be affixed to truck and trailer. Additionally, filling and emptying have to be accomplished under safety measures.
- **MailTraincar:** In this problem, an instance of regular package so-called *mail* is transported by train. Using a train for transportation requires a suitable car and a locomotive to which the car can be connected.
- **RefrigeratedTraincar:** This is used to transport food *(i.e. should kept cool)* by train.
- **AutoTraincar:** Like auto truck problem but using a train. The locomotive has to move to the customer site first.
- **AutoTraincar-bis:** Another version of *AutoTraincar* problem, but the locomotive already is at the customer site.

- **Airplane:** Requires conveyor ramp for loading packages into the plane. Then the plane has to be moved to the destination airport and unloaded there.

Planning Problems in the Satellite Domain

Compared to the *UM-Translog* domain which has a deep decomposition hierarchy , the *Satellite* domain is a shallow structure. The satellite planning problems become more difficult by modelling a repetition of observations, which means that a small number of methods is used multiple times in different context of a plan. Therefore, the problems in our satellite domain experiment are variations over the number of observation tasks, the number of available satellites, and the properties of the required image. All satellite planning problems are solvable, that means, the required image mode is supported by at least one instrument of at least one satellite and each instrument has identified a calibration target. It is important to mention that the decision whether to perform the observations sequentially on one satellite or to distribute them over different satellites is made during the process of developing the plan.
The following items show the satellite problems with x-observations, y-satellite, and z-modes.

- **1obs-1sat-1mod:** This problem has one observation, one satellite and the required image has one mode. In order to find a solution, the satellite is moved in a unique calibration direction. Then, the instrument that is attached to the satellite is calibrated, and finally, the image is taken.

- **1obs-2sat-1mod:** This models the problem of one observation, two satellites, and the required image has one mode. The solution of this problem includes competition between two satellites in order to accomplish the required task. Therefore, the winner satellite will perform the required observation as in problem *1obs-1sat-1mod*.

- **2obs-1sat-1mod:** Similar to 1obs-1sat-1mod problem this problem covers two observations

using one satellite and one mode. The solution is found sequentially. For one of these observations, the solution of *1obs-1sat-1mod* is used. Then, the satellite is moved towards the other target in order to take the second image, yet without additional calibration.

- **2obs-1sat-2mod:** In this problem two observations are performed by one satellite which supports two different modes. The solution for one observation is found as with the first problem, followed by changing the satellite direction and making a new calibration for the second instrument in order to take the new image with the required mode.

- **2obs-2sat-1mod:** This problem deals with two observation tasks which are performed by two satellites the required images having the same mode. This problem can be solved by two alternative solutions: either similar to the solution of the problem *2obs-1sat-1mod* or two symmetric *1obs-1sat-1mod* solutions for each satellite.

- **2obs-2sat-2mod:** This problem involves alternative solutions to solve the problem of two observations, two different satellites and each observation has a special mode. First option is similar to the *2obs-1sat-1mod* problem, that means, two observations are accomplished by one satellite. Of course this solution includes applying the necessary calibration procedure. The second alternative contains two *1obs-1sat-1mod* sub-solutions for every satellite, that means, images are taken concurrently.

4.5.3 Experimental Planning Strategies

In order to quantify the practical performance gained by our hierarchical landmark technique, we conducted a series of experiments with a number of planning strategies [8]. Therefore, we will briefly review the ones on which we based our experiments.

Modification selection functions determine the shape of the search space, because they decide the *(priority of the)* newly added plan refinements. We thereby distinguish selection principles that are

based on a priorization of certain flaw or modification classes and strategies that opportunistically choose from the presented set. The latter ones are called *flexible strategies*.

Representatives for inflexible strategies are the classical HTN strategy patterns that try to balance task expansion with respect to other plan refinements.

The second branch in the planning strategy is a plan selection strategy. Plan selection functions control the traversal through the refinement space that is provided by the modification selection functions.

Flexible Strategies

Flexible strategies are capable of operating on a more general level by exploiting information about the flaw and modification. They are neither flaw-dependent as they do not primarily rely on a flaw type preference schema, nor modification-dependent as they do not have to be biased in favor of specific modification types.

As for the flexible modification selections, we included the well established Least Committing First *(LCF)* paradigm, a generalization of POCL strategies [78, 79] that select those modifications that address flaws for which the smallest number of alternative solutions has been proposed. It is identified by the following equation:

$$\langle \omega_i, \omega_j \rangle \in f_{LCF}^{ModSel}(P, \{f_1, \cdots, f_m\}, \{\omega_1, \cdots, \omega_n\})$$
$$if\ \omega_i \in ModsFor(f_a, P),\ \omega_j \in ModsFor(f_b, P)$$
$$and\ |ModsFor(f_a, P)| < |ModsFor(f_b, P)|$$

As we see, the *LCF* strategy does not depend on the types of issued flaws and modifications, it compares the different alternatives of the modification plan in order to keep the branching factor in the search space low.

The more recent strategies are HotSpot-based strategies: HotSpots denote those components in a plan that are referred to by multiple flaws, thereby quantifying to which extent solving one

deficiency may interfere with the solution options for coupled components. The Direct-Uniform-HotSpot *(DU)* strategy consequently avoids those modifications which address flaws that refer to HotSpot plan components.

$$\langle \omega_i, \omega_j \rangle \in f_{DU}^{ModSel}(P, \{f_1, \cdots, f_m\}, \{\omega_1, \cdots, \omega_n\})$$
$$if\ \omega_i \in ModsFor(f_a, P),\ \omega_j \in ModsFor(f_b, P)$$
$$and\ \sum_{f \in (\{f_1, \cdots, f_m\}\ f_a)} |f \cap f_a| < \sum_{f \in (\{f_1, \cdots, f_m\}\ f_b)} |f \cap f_b|$$

For simplicity suppose we have the following example: The flaw class open pre-condition has two flaws $f_1 = \{te_a, \phi_a\}$ and $f_2 = \{te_b, \phi_b\}$ and the abstract task flaw $f_3 = \{te_a\}$. Then, by applying the *DU* strategy, we found that both flaws f_1 and f_3 refer to the same plan step te_a. Therefore, they have the same HotSpot values which equal 1, while the HotSpot value of flaw f_2 is 0 because it does not appear in another flaw class. Therefore, the *DU* function selects the plan modification of flaw f_2 first.

As a generalization of singular HotSpots to commonly affected areas of plan components, the HotZone *(HZ)* modification selection takes the connections between HotSpots into account and tries to avoid selecting modifications that deal with these clusters.

On the other hand, the plan selection strategies in our experimental evaluation were based on the following strategies:

The least commitment principle on the plan selection level is represented in three different ways: (i) The Fewer Modifications First *(FMF)* strategy, which prefers plans for which a smaller number of modification *(mods)* options has been announced, (ii) The Less Constrained Plan *(LCP)* strategy, which is based on the ratio of plan steps to the number of constraints on the plan, and (iii) The Smaller Detection Ratio *(SDR)* strategy, which relates the number of detected flaws to the number of plan steps.

$$\langle P_i, P_j \rangle\ \in f_{FMF}^{PlanSel}(P_1, \cdots, P_n)\ if\ |mods(P_i)| < |mods(P_j)|$$

$$\langle P_i, \rangle \in f_{LCP}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|\prec_i|+|VC_i|+|CL_i|}{|TE_i|} < \frac{|\prec_j|+|VC_j|+|CL_j|}{|TE_j|}$$

$$\langle P_i, P_j \rangle \in f_{SDR}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|flaws(P_i)|}{|TE_i|} < \frac{|flaws(f_j)|}{|TE_j|}$$

According to the definition of the A^* heuristic function[7] that is dependent on the sum of actual and estimated future costs, Gerevini et al. [80, 81] identified two planning strategies. The first strategy, the so-called $CLOC$, uses the number of causal links (CL) as the actual cost and the number of open pre-conditions (OC) as the estimation cost. While the second strategy, SOC, uses the number of plan steps (S) instead of CL as the actual cost.

$$\langle P_i, P_j \rangle \in f_{SOC}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|TE_i|+|flaws(P_i) \cap F_{OpenPrec}|}{|TE_j|+|flaws(P_j) \cap F_{OpenPrec}|} < 1$$

$$\langle P_i, P_j \rangle \in f_{CLOC}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|CL_i|+|flaws(P_i) \cap F_{OpenPrec}|}{|CL_j|+|flaws(P_j) \cap F_{OpenPrec}|} < 1$$

Schattenberg [46] developed $CLOC$ and SOC strategies to take into account wide-ranging changes of task expansions modifications to the plan structure. This development is done by extending the flaw census to the abstract task class (A). Consequently, considering abstract tasks requires some effort because of the need to deal with future tasks and to maintain the causal structure of the expansion networks. Schattenberg's strategies $SOCA$ and $CLOCA$ are represented as follows:

$$\langle P_i, P_j \rangle \in f_{SOCA}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|TE_i|+|flaws(P_i) \cap (F_{OpenPrec} \cup F_{AbstractTask})|}{|TE_j|+|flaws(P_j) \cap (F_{OpenPrec} \cup F_{AbstractTask})|} < 1$$

$$\langle P_i, P_j \rangle \in f_{CLOCA}^{PlanSel}(P_1, \cdots, P_n) \; if \; \frac{|CL_i|+|flaws(P_i) \cap (F_{OpenPrec} \cup F_{AbstractTask})|}{|CL_j|+|flaws(P_j) \cap (F_{OpenPrec} \cup F_{AbstractTask})|} < 1$$

The strategy $SOCA$ depends on the number of plan steps that includes abstract tasks, but it does not take into consideration that the expansion of an abstract task introduces several plan steps. Therefore, it has been improved into a new strategy, the so-called $PSAOCA$. $PSAOCA$ strategy depends on a new heuristic function called PSA to retrieve the number of plan modifications in a plan's history that have added a task expression.

[7] A^* search will expand nodes that have the lowest value for $g(n) + h(n)$, where $g(n)$ is the *(exact)* cost of the path from the initial state to the current node and $h(n)$ is an estimate of the cost of the cheapest path from node n to the goal node.

$$\langle P_i, P_j \rangle \in f^{PlanSel}_{PSAOCA}(P_1, \cdots, P_n) \; if \; \frac{PSA(P_i)+|flaws(P_i) \cap (F_{OpenPrec} \cup F_{AbstractTask})|}{PSA(P_j)+|flaws(P_j) \cap (F_{OpenPrec} \cup F_{AbstractTask})|} < 1$$

The HotSpot concept can be lifted to the level of plan selection: The Fewer HotZone *(FHZ)* strategy prefers plans with fewer HotZone clusters. The rationale for this search principle is to focus on plans in which the flaws are more closely related and are hence candidates for an early decision concerning compatibility of the refinement options. For *(FHZ)* we use

$$\langle P_i, P_j \rangle \in f^{PlanSel}_{FHZ}(P_1, \cdots, P_n) \; if \; |clusters(flaws(P_i))| < |clusters(flaws(P_j))|$$

where the clusters function considers a set of flaws in its input and returns a set of sets of flaws. Each set represents the inter-connected members of one HotZone.

The next planning search strategy operates on the HotSpot principle implemented on plan modifications. The Fewer Modification-based HotSpots *(FMH)* function summarizes for all refinement-operators that are proposed for a plan the HotSpot values of the corresponding flaws. It prefers those plans for which the ratio of plan modifications to accumulated HotSpot values is less. By doing so, this search schema focuses on plans that are expected to have less interfering refinement options.

Inflexible Strategies

Inflexible strategies are a set of strategies representing a fixed preference schema on the flaw type they want to get eliminated primarily and then select appropriate modification methods. In this subsection, we will give a short systematic overview of the possible instances of inflexible selection schemas for the modification selection strategy and the corresponding plan selection strategy.

The first strategy is a strategy for HTN planning systems called *UMCP*. The *UMCP* strategy prefers to decompose abstract tasks until the primitive plan level is reached. We can simply represent it by the following function:

$$f^{ModSel}_{UMCP} = f^{ModSel}_{Pref-M_{ExpandTask}}$$ [8]

[8] Refer to Definition 8 in Chapter 3 which defines the modification selection strategy function

There are several plan selection strategies that can be used in a *UMCP* planner to decide which plan will be selected from plan space refinement such as:

1. **Breadth-first Search:** The plans are picked up from the induced search space in the order in which they are inserted: first in, first out.

$$f_{UMCP}^{PlanSel} = f_{Pref-First\ Plan}^{PlanSel}$$
$$\text{where } \langle P_i, P_j \rangle \in f_{First\ Plan}^{PlanSel}(P_1, \cdots, P_n)\ for\ 1 \leq i < j \leq n$$

2. **Depth-first Search:** The plans are picked up from the refinement search space in the reverse order in which they are inserted: last in, first out.

$$f_{UMCP}^{PlanSel} = f_{Pref-Last\ Plan}^{PlanSel} \text{ where}$$
$$\langle P_i, P_j \rangle \in f_{Last\ Plan}^{PlanSel}(P_1, \cdots, P_n)\ if\ i = n\ or 1 \leq i < j < n$$

3. **Best-first Search:** The plans in the refinement search space are ranked using a heuristic evaluation function and the plan which has the lowest rank is picked up first.

In later work Tsuneto [82] developed a UMCP planner strategy by introducing the *fewest alternatives first* heuristic technique for selecting task expansions which has the minimal number of alternatives[9]. Therefore, the new UMCP selection strategy is:

$$f_{UMCP}^{ModSel} = f_{Pref-M_{ExpandTask}}^{ModSel} + f_{Pref-LCF}^{ModSel}$$

with the same plan selection functions.

It is furthermore important to mention that the strategy functions can be combined into selection cascades (denoted by the symbol $+$) in which succeeding components decide those cases for which the result of the proceeding ones is a tie.

[9] All possible decomposition methods.

The SHOP[10] modification selection strategy prefers task expansion for the abstract tasks in the order in which they are to be executed. The SHOP planning system has a greater expressiveness than other HTN systems because its current world state is tracked and modified, beginning at the initial state until the task that is bound to be decomposed next. The SHOP system's strategy is represented as follows:

$$f_{SHOP}^{ModSel} = f_{Pref-\prec}^{ModSel} + f_{Pref-M_{ExpandTask^{-1}}}^{ModSel}$$
$$\text{with } f_{SHOP}^{PlanSel} = f_{Pref-first\ Plan}^{PlanSel} + f_{PSAOCA}^{PlanSel}$$

Modification selection strategy $f_{Pref-\prec}^{ModSel}$ reasons about the ordering of the plan steps. Examples for doing this ordering range from using heuristics for flaw selection strategies [83, 84] over doing resource planning [85] to using planning algorithms that depend on an early addressing flaws [72, 86]. The modification selection strategy $f_{Pref-\prec}^{ModSel}$ is identified by the following equation:

$$\langle \omega_i, \omega_j \rangle \in f_{Pref-\prec}^{ModSel}(P, \{f_1, \cdots, f_m\}, \{\omega_1, \cdots, \omega_n\})$$
$$if \omega_i \in ModsFor(f_a, P),\ \omega_j \in ModsFor(f_b, P)$$
$$and\ \forall te_a \in (f_a \cap TE),\ \forall te_b \in (f_b \cap TE):$$
$$(f_a \cap TE \neq \phi \wedge f_b \cap TE \neq \phi) \Rightarrow te_a \prec te_b \in \prec$$

Note that the inverse of a strategy, e.g. plan or modification selection strategy, means that the results of said strategy are inverted.

The third inflexible strategy is the expand-then-make-sound *(EMS)* schema which alternates task expansion modifications with other classes of modifications, *e.g. expanding an abstract task in case no other flaws are issued*. It is formalized by simply preferring resolving threats and open pre-conditions over the task expansion of tasks [41, 87].

$$f_{EMS}^{ModSel} = f_{Pref-F_{Threat}}^{ModSel} + f_{Pref-F_{OpenPrec}}^{ModSel} + f_{Pref-M_{ExpandTask}}^{ModSel}$$
$$\text{with } f_{EMS}^{PlanSel} = f_{Pref-first\ Plan}^{PlanSel}$$

[10] It is the well-known hierarchical planning system

4.5.4 Evaluation Results In The UM-Translog Domain

The set of *UM-Translog* domain problems 4.5.2 is used as experimental environment to evaluate our approach. Each problem runs three times on the same configuration *(same modification selection strategy and plan selection strategy)*. We calculated the average values of three runs. Every run of the planning system was limited to a real time consumption of 9,000 seconds and an exploration of at most 5,000 plans. If a run of the planning problem failed to find a solution within these limits it was considered a non-terminating run and excluded from the computation of the average. Dashes indicate that the run of a planning problem exceeded the previous limitation.

The experimental results are categorized according to the modification selection strategy: Inflexible strategy, Flexible strategy and a combination of different strategies.
The inflexible strategy set includes the three basic strategies: f_{UMCP}^{ModSel}, f_{EMS}^{ModSel}, and f_{SHOP}^{ModSel} with corresponding plan selection as mentioned before.

Table 4.5 shows the runtime behavior of our hierarchical Landmark system *(HLM)* in terms of the size of the average search space and CPU time consumption for problems in the *UM-Translog* domain. Note that the CPU-time denotes the total running time of the planning system in seconds, *including* the pre-processing phase. The column labeled *PANDA* shows the reference system's behavior, the column labeled *HLM* shows the performance of a version performing a pre-processing phase.
Reviewing the overall result in table 4.5, it is quite obvious that the landmark pre-processing pays off in all inflexible strategy configurations and problems. It does so in terms of search space size *Space* as well as in terms of runtime *CPU-Time*. The average performance improvement over all inflexible strategies and over all problems in the *UM-Translog* domain is about 52% in search space size and about 49.2% in planning time as is documented in table 4.5.
The biggest gain is achieved by the *UMCP* strategy. It saves about 11% more than other inflexible strategies *(i.e., EMS and $SHOP$ strategies)*.

Table 4.5 Results for the *UM-Translog* domain with inflexible strategies.

Problem	Inflexible Modification Strategy	PANDA		HLM	
		Space	CPU-Time	Space	CPU-Time
Hopper Truck	UMCP	96	187	58	122
	EMS	351	696	147	295
	SHOP	160	323	89	212
Flatebed Truck	UMCP	164	344	63	149
	EMS	–	–	1571	3797
	SHOP	243	595	98	257
Auto Truck	UMCP	216	535	137	408
	EMS	854	1889	405	976
	SHOP	226	558	164	433
Regular Truck3_Location	UMCP	216	551	177	506
	EMS	592	1278	211	507
	SHOP	163	479	146	406
Regular Truck 2_Region	UMCP	150	293	111	219
	EMS	498	918	127	262
	SHOP	146	283	106	241
Regular Truck_2	UMCP	1396	4893	308	1263
	EMS	–	–	–	–
	SHOP	–	–	926	4005
Regular Truck_1	UMCP	110	215	57	127
	EMS	443	876	114	235
	SHOP	409	911	80	177
Mail Traincar	UMCP	397	994	92	229
	EMS	1314	2641	879	1806
	SHOP	632	1911	121	274
Refrig. RegularTraincar	UMCP	400	952	90	244
	EMS	1030	2056	500	1048
	SHOP	777	1735	173	353
Auto Traincarbis	UMCP	413	1168	161	543
	EMS	–	–	2558	6447
	SHOP	541	1282	247	963
AirPlane	UMCP	91	253	70	215
	EMS	–	–	784	2517
	SHOP	335	821	150	450

On the other hand, the flexible strategy set includes the following strategies: f_{LCF}^{ModSel}, f_{HZ}^{ModSel}, f_{DA}^{ModSel} and f_{DU}^{ModSel}. All these strategies implementations use $f_{FMF}^{PlanSel}$ as the plan selection strategy.

The results of our approach using flexible strategy are shown in tables 4.6 and 4.7. The average performance improvement over all flexible strategies and over all problems in the *UM-Translog* domain is about 39.3% in search space size and 34.27% in planning time.

As documented in tables 4.6 and 4.7, the LCF strategy is the strategy that benefits the most from our pre-processing approach. It saves about 35% more time than other flexible planning strategies *(i.e., HZ, DA and DU strategies)*.The LCF strategy is not only the best performing strategy, but also maintains its performance across the domains.

In general, the flexible and inflexible strategies profit from the hierarchical landmark technique. The flexible planning strategies are very powerful general-purpose procedures and in addition offer potential to be improved by pre-processing methods.

The candidate set of strategy combinations includes:

$f_{LCF}^{ModSel} + f_{HZ}^{ModSel}$ with $f_{FMH}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{LCF}^{ModSel} + f_{EMS}^{ModSel}$ with $f_{FMH}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{HZ}^{ModSel} + f_{LCF}^{ModSel}$ with $f_{FMH}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{LCF}^{ModSel} + f_{DU}^{ModSel}$ with $f_{FHZ}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{HZ}^{ModSel} + f_{LCF}^{ModSel}$ with $f_{FMZ}^{PlanSel} + f_{LCP}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{EMS}^{ModSel} + f_{LCF}^{ModSel}$ with $f_{SOCA}^{PlanSel}$

$f_{EMS}^{ModSel} + f_{LCF}^{ModSel}$ with $f_{FMH}^{PlanSel} + f_{FMF}^{PlanSel}$

$f_{HZ}^{ModSel} + f_{LCF}^{ModSel}$ with $f_{PSAOCA}^{PlanSel}$

Tables 4.8, 4.9 and 4.10 show the runtime behavior of our hierarchical Landmark system *(HLM)* with a lot of strategy comboinations.

As documented in tables 4.8, 4.9 and 4.10, our HLM system profits from the strategy combi-

Table 4.6 Results for the *UM-Translog* domain with flexible strategies-1.

Problem	Inflexible Modification Strategy	PANDA		HLM	
		Space	CPU-Time	Space	CPU-Time
Hopper Truck	LCF	100	187	55	118
	HZ	71	149	55	121
	DA	243	598	144	352
	DU	339	692	101	225
Flatebed Truck	LCF	120	264	62	179
	HZ	294	700	159	399
	DA	181	395	99	237
	DU	–	–	1047	2601
Auto Truck	LCF	211	593	155	470
	HZ	223	547	197	527
	DA	1250	4695	644	2077
	DU	742	1904	459	1304
Regular Truck-3_Location	LCF	255	630	162	463
	HZ	299	745	191	474
	DA	257	615	239	562
	DU	–	–	1508	4097
Regular Truck-2_Region	LCF	100	198	78	173
	HZ	75	155	55	117
	DA	157	350	114	257
	DU	447	859	160	460
Regular Truck_2	LCF	1479	5047	327	1278
	HZ	–	–	–	–
	DA	–	–	723	2560
	DU	–	–	–	–
Regular Truck_1	LCF	139	267	127	222
	HZ	75	171	55	137
	DA	252	578	148	352
	DU	198	391	117	258

Table 4.7 Results for the *UM-Translog* domain with flexible strategies-2.

Problem	Inflexible Modification Strategy	PANDA		HLM	
		Space	CPU-Time	Space	CPU-Time
Mail Traincar	LCF	759	1819	79	209
	HZ	104	311	81	224
	DA	873	2604	641	2032
	DU	446	1193	424	1090
Refrig. Regular Traincar	LCF	752	1878	90	225
	HZ	104	273	76	196
	DA	812	2414	588	1958
	DU	536	1319	307	775
Auto Traincar bis	LCF	1280	3573	227	926
	HZ	751	1712	701	1616
	DA	216	732	184	705
	DU	1700	4609	1390	4018
AirPlane	LCF	287	827	247	798
	HZ	509	1517	345	1323
	DA	282	799	172	620
	DU	–	–	643	2135

nations procedure. It saves between 36% and 61% in search space size, and between 26% and 58% in planning time.

In general, our experiments show that transportation tasks that include special goods and transportation means, e.g., the transport of auto-mobiles, frozen goods, and mail via train, achieve better performance than other transportation tasks. Here the gain is between 32% and 69%.

4.5.5 Evaluation Results In The Satellite Domain

Next, we will look at the performance of our Hierarchical landmark pre-processing technique used on planning problems in the *Satellite* domain. Each problem includes a different number of observations, a number of satellites and a number of modes. As mentioned before, the experimental

Table 4.8 Results for the *UM-Translog* domain with strategy combinations-1.

Problem	Mod. Sel.	Plan Sel.	PANDA		HLM	
			Space	CPU-Time	Space	CPU-Time
Hopper Truck	LCF+HZ	FMH+FMF	72	147	41	95
	LCF+EMS	FMH+FMF	101	211	72	174
	HZ+LCF	FMH+FMF	239	724	47	166
	LCF+DU	FHZ+FMF	75	155	46	99
	HZ+LCF	FHZ+LCP+FMF	71	143	54	115
	EMS+LCF	SOCA	505	974	225	525
	EMS+LCF	FMH+FMF	–	–	1290	3811
	HZ+LCF	PASOCA	82	211	59	156
Flatebed Truck	LCF+HZ	FMH+FMF	81	182	58	140
	LCF+EMS	FMH+FMF	120	269	90	216
	HZ+LCF	FMH+FMF	85	235	53	260
	LCF+DU	FHZ+FMF	96	216	54	129
	HZ+LCF	FHZ+LCP+FMF	130	299	69	162
	EMS+LCF	SOCA	–	–	927	2826
	EMS+LCF	FMH+FMF	–	–	1791	5214
	HZ+LCF	PASOCA	131	388	69	206
Auto Truck	LCF+HZ	FMH+FMF	119	301	85	236
	LCF+EMS	FMH+FMF	191	443	114	298
	HZ+LCF	FMH+FMF	221	724	161	691
	LCF+DU	FHZ+FMF	129	314	92	251
	HZ+LCF	FHZ+LCP+FMF	183	469	157	413
	EMS+LCF	SOCA	1629	3949	374	1498
	EMS+LCF	FMH+FMF	–	–	1268	3314
	HZ+LCF	PASOCA	312	969	201	696
Regular Truck-3_Location	LCF+HZ	FMH+FMF	149	377	73	203
	LCF+EMS	FMH+FMF	234	613	105	206
	HZ+LCF	FMH+FMF	112	411	80	278
	LCF+DU	FHZ+FMF	241	483	131	370
	HZ+LCF	FHZ+LCP+FMF	190	458	115	307
	EMS+LCF	SOCA	1317	3053	327	987
	EMS+LCF	FMH+FMF	3163	8448	300	868
	HZ+LCF	PASOCA	292	1161	87	290

Table 4.9 Results for the *UM-Translog* domain with strategy combinations-2.

Problem	Mod. Sel.	Plan Sel.	PANDA		HLM	
			Space	CPU-Time	Space	CPU-Time
Regular Truck-2_Region	LCF+HZ	FMH+FMF	70	142	42	98
	LCF+EMS	FMH+FMF	106	216	81	182
	HZ+LCF	FMH+FMF	60	154	45	141
	LCF+DU	FHZ+FMF	83	160	46	105
	HZ+LCF	FHZ+LCP+FMF	75	152	54	122
	EMS+LCF	SOCA	670	1470	129	308
	EMS+LCF	FMH+FMF	–	–	1260	3283
	HZ+LCF	PASOCA	111	298	59	166
Regular Truck_2	LCF+HZ	FMH+FMF	–	–	275	1237
	LCF+EMS	FMH+FMF	–	–	293	1144
	HZ+LCF	FMH+FMF	–	–	429	2327
	LCF+DU	FHZ+FMF	753	2755	295	1262
	HZ+LCF	FHZ+LCP+FMF	–	–	787	3544
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	–	–
	HZ+LCF	PASOCA	–	–	–	–
Regular Truck_1	LCF+HZ	FMH+FMF	72	149	41	92
	LCF+EMS	FMH+FMF	109	225	78	179
	HZ+LCF	FMH+FMF	140	388	45	138
	HZ+LCF	FHZ+LCP+FMF	74	153	54	120
	LCF+DU	FHZ+FMF	84	173	46	104
	EMS+LCF	SOCA	495	657	141	314
	EMS+LCF	FMH+FMF	–	–	304	790
	HZ+LCF	PASOCA	79	202	59	203
Mail Traincar	LCF+HZ	FMH+FMF	380	1241	89	221
	LCF+EMS	FMH+FMF	590	1805	138	313
	HZ+LCF	FMH+FMF	94	296	84	279
	LCF+DU	FHZ+FMF	559	1450	64	160
	HZ+LCF	FHZ+LCP+FMF	93	213	70	171
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	2131	7937
	HZ+LCF	PASOCA	214	621	172	602

Table 4.10 Results for the *UM-Translog* domain with strategy combinations-3.

Problem	Mod. Sel.	Plan Sel.	PANDA		HLM	
			Space	CPU-Time	Space	CPU-Time
Refrig. Regular Traincar	LCF+HZ	FMH+FMF	384	1240	89	215
	LCF+EMS	FMH+FMF	634	1861	138	315
	HZ+LCF	FMH+FMF	178	621	74	260
	LCF+DU	FHZ+FMF	446	1074	64	159
	HZ+LCF	FHZ+LCP+FMF	92	198	70	172
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	–	–
	HZ+LCF	PASOCA	208	723	172	554
Auto Traincar bis	LCF+HZ	FMH+FMF	342	1137	144	421
	LCF+EMS	FMH+FMF	460	1425	177	477
	HZ+LCF	FMH+FMF	466	1587	456	1362
	LCF+DU	FHZ+FMF	365	1044	107	328
	HZ+LCF	FHZ+LCP+FMF	357	958	278	770
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	2074	7347
	HZ+LCF	PASOCA	683	2411	615	2076
AirPlane	LCF+HZ	FMH+FMF	164	507	141	435
	LCF+EMS	FMH+FMF	142	413	167	471
	HZ+LCF	FMH+FMF	158	588	99	422
	LCF+DU	FHZ+FMF	257	749	200	621
	HZ+LCF	FHZ+LCP+FMF	280	777	240	700
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	–	–
	HZ+LCF	PASOCA	349	1853	271	1006

Table 4.11 Results for the *Satellite* domain with inflexible strategies.

Problem	Inflexible Modification Strategy	PANDA		HLM	
		Space	CPU-Time	Space	CPU-Time
1Obs-1Sat-1Mod	UMCP	83	91	83	91
	EMS	68	74	65	60
	SHOP	61	67	57	61
1Obs-2Sat-1Mod	UMCP	31	36	34	38
	EMS	49	55	47	53
	SHOP	103	109	105	111
2Obs-1Sat-1Mod	UMCP	1132	1336	883	1035
	EMS	1942	2856	1586	2608
	SHOP	251	270	237	264
2Obs-1Sat-2Mod	UMCP	3072	4151	278	1097
	EMS	–	–	1219	1579
	SHOP	–	–	–	–
2Obs-2Sat-1Mod	UMCP	458	1215	278	1097
	EMS	–	–	1219	1579
	SHOP	–	–	1406	1780
2Obs-2Sat-2Mod	UMCP	4376	5891	1062	1270
	EMS	–	–	–	–
	SHOP	–	–	–	–

results are categorized according to the modification selection strategies: Inflexible strategy, Flexible strategy and finally a combination of different strategies.

Although the problem instances of the *Satellite* domain appear to be very small, the resulting search space is surprisingly large. Therefore, the evaluation of the satellite observation planning problems are very difficult.

In general, the *Satellite* domain does not benefit significantly from the landmark technique due to its shallow decomposition hierarchy. We are able, however, to solve problems for which the participating strategies do not find solutions within the given resource bounds such as problems *2Obs-1Sat-1Mod*,*2Obs-2Sat-1Mod* and *2Obs-2Sat-2Mod* with HZ strategy.

Table 4.12 Results for the *Satellite* domain with flexible strategies.

Problem	Inflexible Modification Strategy	PANDA		HLM	
		Space	CPU-Time	Space	CPU-Time
1Obs-1Sat-1Mod	LCF	90	95	86	93
	HZ	62	76	61	64
	DA	57	67	56	60
	DU	137	146	100	107
1Obs-2Sat-1Mod	LCF	144	154	71	77
	HZ	120	132	57	62
	DA	65	75	68	78
	DU	239	270	139	150
2Obs-1Sat-1Mod	LCF	1355	1551	1120	1338
	HZ	–	–	1281	4764
	DA	1239	2136	782	1131
	DU	–	–	–	–
2Obs-1Sat-2Mod	LCF	–	–	3022	4069
	HZ	–	–	–	–
	DA	–	–	832	1301
	DU	–	–	–	–
2Obs-2Sat-1Mod	LCF	–	–	608	1174
	HZ	–	–	1094	1338
	DA	2190	6850	2186	6841
	DU	–	–	–	–
2Obs-2Sat-2Mod	LCF	–	–	–	–
	HZ	–	–	871	1114
	DA	1155	1672	142	175
	DU	–	–	–	–

Table 4.13 Results for the *Satellite* domain with strategy combinations-1.

Problem	Mod. Sel.	Plan Sel.	PANDA		HLM	
			Space	CPU-Time	Space	CPU-Time
1Obs-1Sat-1Mod	LCF+HZ	FMH+FMF	38	41	37	42
	LCF+EMS	FMH+FMF	46	51	46	53
	HZ+LCF	FMH+FMF	44	53	35	38
	LCF+DU	FHZ+FMF	67	72	67	72
	HZ+LCF	FHZ+LCP+FMF	58	62	53	60
	EMS+LCF	SOCA	55	61	53	59
	EMS+LCF	FMH+FMF	31	36	29	33
	HZ+LCF	PASOCA	63	68	35	37
1Obs-2Sat-1Mod	LCF+HZ	FMH+FMF	142	159	77	95
	LCF+EMS	FMH+FMF	38	42	28	39
	HZ+LCF	FMH+FMF	82	90	70	82
	LCF+DU	FHZ+FMF	126	137	117	132
	HZ+LCF	FHZ+LCP+FMF	118	131	116	128
	EMS+LCF	SOCA	46	52	42	48
	EMS+LCF	FMH+FMF	46	52	45	50
	HZ+LCF	PASOCA	120	131	112	120
2Obs-1Sat-1Mod	LCF+HZ	FMH+FMF	602	788	539	708
	LCF+EMS	FMH+FMF	964	1631	903	1428
	HZ+LCF	FMH+FMF	394	480	321	374
	LCF+DU	FHZ+FMF	1135	1319	901	1030
	HZ+LCF	FHZ+LCP+FMF	1468	1699	1216	1474
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	645	1215	642	1211
	HZ+LCF	PASOCA	–	–	–	–
2Obs-1Sat-2Mod	LCF+HZ	FMH+FMF	–	–	–	–
	LCF+EMS	FMH+FMF	–	–	–	–
	HZ+LCF	FMH+FMF	–	–	–	–
	LCF+DU	FHZ+FMF	–	–	–	–
	HZ+LCF	FHZ+LCP+FMF	–	–	–	–
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	–	–	2615	3848
	HZ+LCF	PASOCA	–	–	–	–

Table 4.14 Results for the *Satellite* domain with strategy combinations-2.

Problem	Mod. Sel.	Plan Sel.	PANDA		HLM	
			Space	CPU-Time	Space	CPU-Time
2Obs-2Sat-1Mod	LCF+HZ	FMH+FMF	–	–	–	–
	LCF+EMS	FMH+FMF	–	–	–	–
	HZ+LCF	FMH+FMF	–	–	–	–
	LCF+DU	FHZ+FMF	–	–	4252	5294
	HZ+LCF	FHZ+LCP+FMF	–	–	–	–
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	1482	2957	1470	2893
	HZ+LCF	PASOCA	–	–	–	–
2Obs-2Sat-2Mod	LCF+HZ	FMH+FMF	–	–	–	–
	LCF+EMS	FMH+FMF	–	–	489	646
	HZ+LCF	FMH+FMF	–	–	–	–
	LCF+DU	FHZ+FMF	–	–	–	–
	HZ+LCF	FHZ+LCP+FMF	–	–	–	–
	EMS+LCF	SOCA	–	–	–	–
	EMS+LCF	FMH+FMF	1230	1839	992	1625
	HZ+LCF	PASOCA	–	–	–	–

Chapter 5

Landmark Planning Strategies

An AI planning system searches a space of partially-developed plans in order to find a solution plan that solves a given planning problem. In general, for a given planning problem, a planner considers a goal as a skeletal plan and successively refines it *(i.e. by adding/deleting plan steps or constraints in/from the current plan)* until a plan *(solution)* that satisfies the given planning problem is created. Since the search space might be infinite, it is important to search for a solution plan in an efficient way. The efficiency of a planning system depends on the planner's search strategy *(refinement strategy)*.

In order to understand how refinement strategies can improve the efficiency of hierarchical planners, we will address plan refinement strategies for hierarchical planning in this chapter. After that, we will introduce novel refinement strategies for hierarchical planning, inspired by the idea of the hierarchical landmarks.

5.1 Refinement Strategy

A lot of AI planning systems rely on the idea of *refinement search*. This means that the planner steadily refines partially-developed plans into more detailed plans, until it finds a final solution plan

that achieves the problem requirments. To this end, the refinement search strategy depends on detecting flaws *(i.e., violations of the solution criteria)* and proposing possible ways or modifications *(i.e., change in the current plan data structure)* that can solve them. In general, planning systems explore only a fraction of the search space which is theoretically reachable (See Figure 5.1). One always wants to keep this fraction as small as possible, since the runtime of a planning system correlates to the number of explored search nodes.

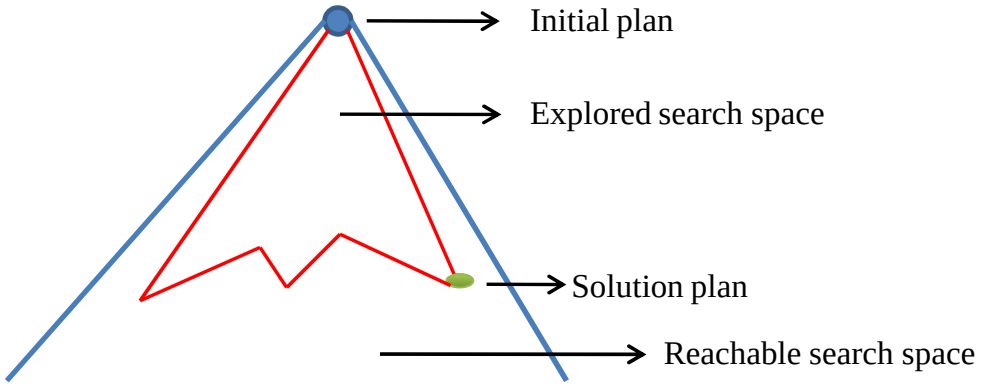


Figure 5.1 Graphical representation of search process

For example, in order to transform the current partial plan $plan\_i$ (See Figure 5.2) into a more specific plan *(plan refinement)*, a planner refines $plan\_i$ by decomposing task t_1 or task t_2 by applying the appropriate method, by binding variable constraints X, by adding an ordering constraints between tasks, or by solving a causal link thread. Suppose the planner chooses to refine the variable constraints X first. In this case, there are three different partial plans in the refinement search space. In each plan, the variable X has different values. As shown in figure 5.3, these plans are $plan\_i_a$, $plan\_i_b$ and $plan\_i_c$.

Suppose the planner concludes, by examining the resulting partial plans that the partial plan $Plan\_i_c$ can not lead to a consistent plan. The planner can therefore, prune or omit the partial plan $Plan\_i_c$. Now, we assume that the planner carries on with the partial plan $Plan\_i_a$ and chooses

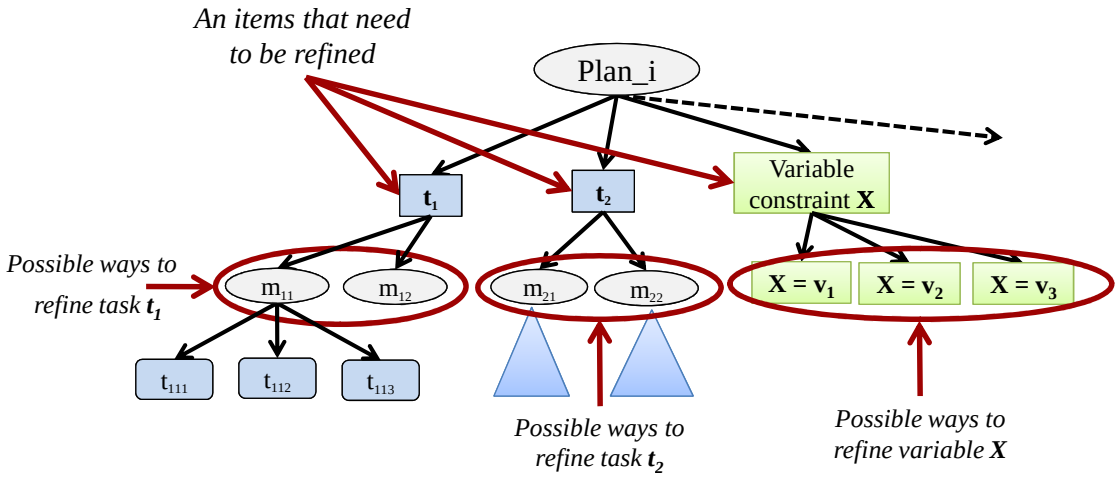


Figure 5.2 Possible ways to refine partial plan $plan\_i$

to refine task t_1 by applying methods m_{11} and m_{12}. This refinement produces two different partial plans $Plan\_i_{am11}$ and $Plan\_i_{am12}$. Therefore, a *refinement search strategy* helps a planner to compare the available plan refinement options in order to focus more efficiently towards a solution plan. In general, refinement search strategies identify how and in what order these refinements are done.

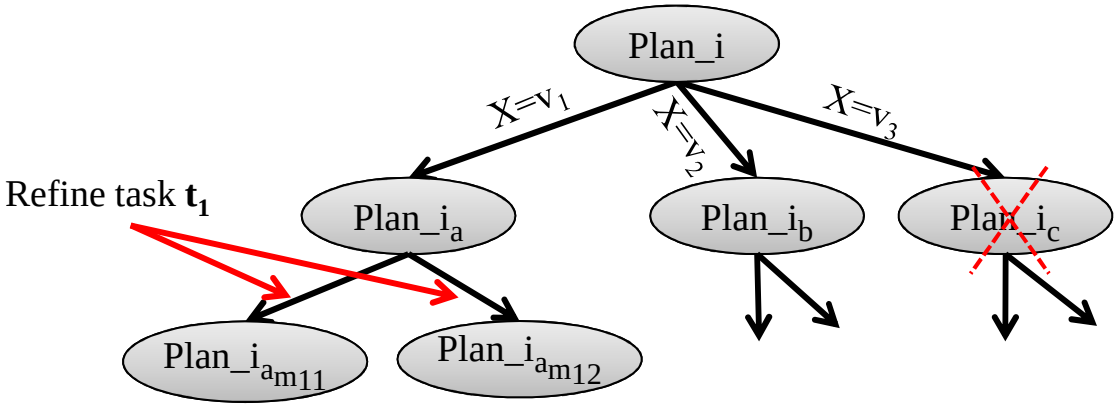


Figure 5.3 Part of a refinement search space of concerning the partial plan $plan\_i$

5.2 Least Commitment Strategies

In general, a lot of refinement search strategies for both action-based planning and hierarchical planning use a *'least commitment strategy'* technique. This is any refinement search strategy that tries to avoid unnecessary branching in the search tree by either postponing specific refinements or performing the necessary changes in the current partial plans. A lot of contributions have been made regarding least commitment techniques. In this section we will illustrate them.

Sacerdoti is considered the be first researcher who introduced the notion of *least commitments* [16]. In his planner NOAH, instead of using a linear sequence of plan steps in partial plans, he used a partially-ordered graph to identify the step ordering between plan steps in a partial plan. His experiments proved that using a partially-ordered graph avoids unnecessary commitments to a particular step ordering when achieving sub-goals. This will cause a reduction of search space and thus an improvement in planning efficiency [6]. In order to keep a plan that has un-ordered plan steps consistent, Chapman [15] introduced the ***M**odel **T**ruth **C**riterion(MTC)* to evaluate conditions in his planner. The evaluation value of a condition c determines if the condition c should be achieved at the moment or be postponed. Chapman's planner terminates and returns a solution plan when all conditions in the plan are satisfied. Another way to evaluate the value of condition c is to use causal links which was introduced by Tate [18]. MTC handles threats by inserting another establishing plan step between the threatening step and the consuming step. The causal links in a plan however, should not be violated by another plan step. If the causal link is violated, the plan is considered inconsistent and pruned from the search space [16].

On the other hand, least commitment techniques can rely on variable binding. In action-based planning such as [10, 18], once the operator has been introduced in a partial plan, the operator variables are instantiated immediately into constant values. In contrast to that, Yang et al. [88], Stefik [89] and Veloso et al. [90] preferred to postpone instantiating variables until it becomes absolutely necessary, Because instantiating operator variables directly after introducing operator

in the plan will produce a huge search space *(i.e., if a variable has 20 possible values, then this will create 20 branches in the search space)*. Therefore, the planner can delay a variable binding decision until the variables are more constrained.

On the other hand, SNLP uses least commitment technique as follows: at every refinement cycle, all threats are resolved before satisfying the next open condition in the current partial plan. In general, threats are solved by three different methods: the *Demotion* process which moves the threatening step before the step that produces the protected literal in the causal link. The *Promotion* process that moves the threatening step behind the step whose pre-condition the causal link protects. Finally, *Separation* binds variables so that the effects of the threatening step and the corresponding condition of the causal link will not unify.

After that, Joslin et al. [78] proposed a new technique,LCFR[1]. The LCFR strategy is considered a generalization of the ***D****elay-****Unf****orced threats (DUnf)* strategy [91]. DUnf prefers to solve threat flaws before openCondition flaws. LCFR strategy has a uniform mechanism to handle all kinds of flaws. LCFR strategy prefers a flaw that has the lowest possible repair cost at a given node. The repair cost is defined as the number of generated nodes which represent all the possible ways to solve the corresponding flaw. Peat et al. [92] found that delaying threat removal is better than solving it immediately because performing other necessary planning operations may cause a threatening situation to go away. Their results have been picked up by Schubert et al. [80] who proposed simple but effective plan metrics for an A^* heuristic.

On the other hand, planning strategies of the hierarchical paradigm differ in the ways they select appropriate methods and interleave the decomposition of tasks with measures to resolve causal interactions between tasks.

In general, there are some considerations that help to choose a commitment strategy for hierarchical planning.

[1]LCFR $\Longrightarrow$ **L**east-**C**ost **F**law **R**epair

1. **Refine abstract tasks first or refine constraints first?** In this least commitment strategy, the process of constraint refinements is postponed until the planner gets a primitive plan. Since some variable constraints and ordering constraints might not be fully instantiated while the plan still has abstract tasks. Therefore, earlier constraint refinement helps the planner to eliminate the redundancy of working on the same constraints several times and of course to prune the search space such as RVBS[2] which does least commitment to variable bindings.
   On the other hand, expanding an abstract task can be delayed as much as possible. This is done by EVIS[3] which delays the expansion of an abstract task until all variable constraints are committed. Finally, the DVCS [4] chooses between RVBS and EVIS according to which one looks best for the task at hand *(i.e., it minimizes the branching factor in the search space by selecting the strategy which returns the smallest value)* [93, 94].

2. **Which abstract tasks to refine?** This is analogous to the problem of goal selection in classical state-based planning. Any systematic expansion can be used. One can prefer to use a depth first expansion or a breadth first expansion.

Tsuneto et al. [95] described the fewest alternatives first heuristic for selecting task expansions in the UMCP system as well as identifying and handling some kinds of *external conditions* automatically [9]. The external conditions strategy is a task selection strategy that specifies the order in which a planner will prefer to expand abstract tasks. McCluskey [87] introduced the *Expand-then-Make-Sound* method in which the expansion of an abstract task is followed by repairing the plan's causal structure. Finally, some planning systems of the SHOP-family, like SHOP2, expand tasks in the order in which they are to be executed and consider causality only on primitive levels [39]. The process of deciding which commitment strategy is best is a hard problem because this decision relies on the kind of application domain as well as on the planning problem being solved

[2] RVBS $\Longrightarrow$ **R**eluctant **V**ariable **B**indings **S**trategy

[3] EVIS $\Longrightarrow$ **E**ager **V**ariable **I**nstantiation **S**trategy

[4] RVBS $\Longrightarrow$ **D**ynamic **V**ariable **C**ommitment **S**trategy

in that domain.

In general, all planning systems have a refinement strategy implicitly defined and indirectly controlled via parameters. In contrast to that, the modularity of the component of our refinement planning framework [7] allows us to completely separate the computation of flaws (Definition 6) from the computation of plan modifications (Definition 7) and in turn both computations can be separated from control search strategy. This separation allows us to define not only a single planning strategy function, but also define a combination of planning strategy functions.

The use of landmarks in hierarchical planning is quite new. In classical state-based planning the concept of landmarks enabled the development of strong heuristics [70,96]. LAMA, the currently best performing classical planner uses such a landmark heuristic [66]. Therefore, the information about landmarks which are extracted by applying our hierarchical landmark algorithm can be exploited in two ways: first, the reduction of domain models or, more precisely, the transformation of a universal domain model into one that includes problem-specific pruning information. Second, deducing heuristic guidance from knowledge about which tasks have to be decomposed on refinement paths that lead towards a solution.

In the next section, we will focus on the latter by presenting novel domain-independent strategies *(modification selection strategies)* that exploit landmark information to speed up the hierarchical planning process.

5.3 Landmark-aware Strategies

Making landmark information operational is generally based on the idea that properly identifying or anticipating the landmarks along the refinement paths perfectly guides the search process because the landmarks are known to be in some sense "inevitable" elements on the way to any solution. The mandatory sets in our landmark table do not contribute directly to the identification of a

solution path in this way because our approach over-approximates landmarks in primarily dealing with local landmarks.

They do, however, allow to estimate upper and lower bounds for the number of expansion refinements that an abstract task implicitly requires before a solution is found. Given a landmark table entry $\langle t(\bar{\tau}), M(t(\bar{\tau})), O(t(\bar{\tau}))\rangle$, the situation can be characterized as follows: If the planning system decomposes the task $t(\overline{\tau})$, all tasks in the mandatory set $M(t(\bar{\tau}))$ are introduced into the refinement plan, no matter which method is used. The optional tasks now make the difference and with the information at hand we can infer that in the most optimistic case, a solution can be developed straight from the implementation of the method with the "smallest" remains according to $O(t(\bar{\tau}))$. Following a similar argument, the upper bound for the "expansion effort" can be obtained by adding the efforts for all implementations that are stored in the optional set.

From the above considerations, two essential properties of our landmark-aware strategies emerge: First, since the landmark exploitation will be defined in terms of measuring expansion alternatives, the resulting strategy will be a modification ordering function and not a plan ordering one. Second, the landmark table entries can be normalized in some sense by focusing on the modification preference on the optional sets in the landmark table entries, we implement an abstract view on the method definition that realizes the least-commitment principle.

Concerning the first two strategies below, we interpret the term "expansion effort" literally and therefore define "smallest" method to be the one with the fewest abstract tasks in its implementing plan. This kind of heuristic is obviously a rough over-estimation for the search effort because the landmark table typically contains a number of tasks that turn out to be un-achievable in the given problem; this is due to the relaxation of the reachability analysis. The strategy also does not take into account the refinement effort that it takes to make an implementation operationable on the primitive level by establishing causal links, resolving causal threats, and grounding tasks. For the time being, we assume that all methods deviate more or less to the same amount in terms of both

factors. We will see that this simplification already yields a heuristic with good performance. To this end, we define the cardinality of a set of tasks in terms of the number of corresponding entries that a given landmark table does contain.

Definition 15 (Landmark Cardinality)**.** *Given a landmark table LT, we define the landmark cardinality of a set of tasks $O = \{t_1(\bar{\tau_1}), \ldots, t_n(\bar{\tau_n})\}$ to be*

$$|O|_{LT} := |\{t(\bar{\tau}) \in O | \langle t(\bar{\tau}), M(t(\bar{\tau})), O(t(\bar{\tau}))\rangle \in LT\}|$$

5.3.1 Landmark-aware Strategy(lm_1)

Based on Landmark Cardinality definition(Definition 15), the 'LT Options ordering lm_1' strategy is defined as follows:

Definition 16 (Landmark-aware strategy lm_1)**.** *For a given plan $P = \langle TE, C\rangle$ where $C = \langle \prec, VC, CL\rangle$, let $t_i(\bar{\tau_i})$ and $t_j(\bar{\tau_j})$ be ground instances of two abstract tasks in TE that are compatible with the (in-) equations in VC and that are referenced by two abstract task flaws f_i and f_j, respectively, that are found in plan P. Let a given landmark table LT contain the corresponding entries $\langle t_i(\bar{\tau_i}), M(t_i(\bar{\tau_i})), O(t_i(\bar{\tau_i}))\rangle$ and $\langle t_j(\bar{\tau_j}), M(t_j(\bar{\tau_j})), O(t_j(\bar{\tau_j}))\rangle$.*
The modification ordering function lm_1 then orders a plan modification ω_i before ω_j ($\omega_i \prec \omega_j$) if and only if ω_i addresses flaw f_i, ω_j addresses flaw f_j, and

$$\sum_{o \in O(t(\bar{\tau})i)} |o|_{LT} < \sum_{o \in O(t(\bar{\tau})j)} |o|_{LT}$$

As has been introduced above, this strategy implements a rationale that is similar to the least commitment principle, because it favors those plan refinements that impose less successor plans, that means, it reduces the effective branching factor of the search space (cf. *fewest alternatives first* heuristic in HTN planning [95]). The proper choice of the grounded task instances $t_i(\bar{\tau_i})$ and $t_j(\bar{\tau_j})$ in the above definition is crucial for the actual performance, however, because the plan modifications typically operate on the lifted abstract tasks and method definitions. For our experiments,

we implemented a random choice on the compatible grounded landmark table entries, future work will however focus on a better informed candidate selection.

$Landmark$	$Mandatory(M)$	$Options(O)$
T_1	.....	$\{\{T_{11},\ T_{12}\}_{m\_1}, \{T_{21}\}_{m\_2}, \{t_{31},\ T_{32}\}_{m\_3}\}$
T_{11}	.....	$\{\{t_{111},\ t_{112}\}_{m\_111}, \{T_{113}\}_{m\_112}, \{t_{114},\ t_{115}\}_{m\_113}\}$
T_{12}	.....	$\{\{t_{121},\ t_{122}\}_{m\_121}\}$
T_{21}	.....	$\{\{t_{211},\ t_{212}\}_{m\_211}, \{T_{213}\}_{m\_212}\}$
T_{32}	.....	$\{\{t_{321},\ t_{322}\}_{m\_321}\}$
T_{113}	.....	$\{\{T_{1131},\ t_{1132}\}_{m\_1131}\}$
T_{213}	.....	$\{\{t_{2131},\ t_{2132}\}_{m\_2131}\}$
T_{1131}	.....	$\{\{t_{11311},\ t_{11312}\}_{m\_11311}\}$
T_2	.....	.....
.....	.....	.....
.....	.....	.....

Figure 5.4 Part of landmark table

For example, as depicted in figure 5.4, suppose a current partial plan includes an abstract task flaw T_1 which has three option sets $\{\{T_{11}, T_{12}\}_{m\_1}, \{T_{21}\}_{m\_2}, \{t_{31}, T_{32}\}_{m\_3}\}$ in LT and suppose an abstract task flaw T_2 has two option sets in LT. By applying our search strategy lm_1 we found that the landmark cardinality of abstract task flaw T_1 is $|O(T_1)|_{LT} = |T_{11}, T_{12}| + |T_{21}| + |T_{32}|$. Then, the cardinality value of task T_1 is 4. Assume the cardinality value of task T_2 is 5. Then, the lm_1 strategy will prefer to expand abstract task T_1 before decomposing abstract task T_2 because $|O(T_1)|_{LT} < |O(T_2)|_{LT}$.

While the above heuristic focuses on the very next level of refinement, a strategy should also take into account estimates for subsequent refinement levels, thus minimizing the number of refinement choices until no more decompositions are necessary. To this end, we will introduce the landmark-aware strategies lm_1^*.

5.3.2 Landmark-aware Strategy(lm_1^*)

In order to compute the cardinality value of an abstract task flaw according to subsequent refinement levels, we assume that the set $O^*(t(\bar{\tau}))$ be the "transitive closure of the optional set" on a recursive traversal of the table entries, beginning in $t(\bar{\tau})$. More formally:

Definition 17 (Closure of the Optional Set)**.** *The closure of the optional set for a given ground task $t(\bar{\tau})$ and landmark table LT is defined as*

$$O^*(t(\overline{\tau})) = \begin{cases} \emptyset & if\ t(\bar{\tau})\ is\ a\ primitive\ task \\ O(t(\overline{\tau})) \cup \bigcup\limits_{o \in O(t(\overline{\tau}))} \Big(\bigcup\limits_{t'(\overline{\tau}') \in o} O^*(t'(\overline{\tau}')) \Big) & Otherwise \end{cases}$$

$$with\ \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau})) \rangle \in LT$$

We would like to point out that $O^*(t(\bar{\tau}))$ is always finite due to the finiteness of the landmark table, even for cyclic method definitions. In the following definition, the landmark-aware strategy lm_1^* traces the entries in the optional task sets into the related entries of the landmark table.

Definition 18 (Landmark-aware strategy lm_1^*)**.** *Given a plan $P = \langle TE, C \rangle$ where $C = \langle \prec, VC, CL \rangle$, let $t_i(\bar{\tau}_i)$ and $t_j(\bar{\tau}_j)$ be ground instances of two abstract tasks in TE that are compatible with the (in-) equations in VC and that are referenced by two abstract task flaws f_i and f_j, respectively, that are found in plan P. Let a given landmark table LT contain the corresponding entries $\langle t_i(\bar{\tau}_i), M(t_i(\bar{\tau}_i)), O(t_i(\bar{\tau}_i)) \rangle$ and $\langle t_j(\bar{\tau}_j), M(t_j(\bar{\tau}_j)), O(t_j(\bar{\tau}_j)) \rangle$.*
The modification ordering function lm_1^* then orders a plan modification ω_i before ω_j ($\omega_i \prec \omega_j$) if and only if ω_i addresses flow f_i, ω_j addresses flaw f_j, and

$$\sum_{o \in O^*(t(\overline{\tau})i)} |o|_{LT} < \sum_{o \in O^*(t(\overline{\tau})j)} |o|_{LT}$$

For example, by applying our search strategy lm_1^* in figure 5.4, we found that the landmark cardinality of abstract task flaw T_1 is $|O(T_1)|_{LT} = |T_{11}, T_{12}| + |T_{21}| + |T_{32}| + |T_{113}| + |T_{213}| + |T_{1131}|$. Then, the cardinality value of abstract task T_1 is 7. Assume the cardinality value of abstract task T_2

is 6. Then, the lm_1^* strategy will prefer to decompose abstract task T_2 before decomposing abstract task T_1 because $|O(T_2)|_{LT} < |O(T_1)|_{LT}$.

So far, the "expansion effort" has been measured in terms of decomposition refinements that have to be applied to the current plan until a solution plan is obtained.

The following two strategies take into account that primitive task in a decomposition also contributes to the costs for developing the current plan into a solution. The cost measure is thereby a unified one, that means, solving the flaws concerning a primitive task are regarded as expensive as the expansion of an abstract task.

5.3.3 Landmark-aware Strategy(lm_2)

Definition 19 (Landmark-aware strategy lm_2)**.** *For a given plan $P = \langle TE, C\rangle$ where $C = \langle \prec, VC, CL\rangle$, let $t_i(\bar{\tau}_i)$ and $t_j(\bar{\tau}_j)$ be ground instances of two abstract tasks in TE that are compatible with the (in-) equations in VC and that are referenced by two abstract task flaws f_i and f_j, respectively, that are found in plan P. Let a given landmark table LT contain the corresponding entries $\langle t_i(\bar{\tau}_i), M(t_i(\bar{\tau}_i)), O(t_i(\bar{\tau}_i))\rangle$ and $\langle t_j(\bar{\tau}_j), M(t_j(\bar{\tau}_j)), O(t_j(\bar{\tau}_j))\rangle$.*

The modification ordering function lm_2 then orders a plan modification ω_i before ω_j ($\omega_i \prec \omega_j$) if and only if ω_i addresses flaw f_i, ω_j addresses flaw f_j, and

$$\sum_{o \in O(t(\bar{\tau})i)} |o| < \sum_{o \in O(t(\bar{\tau})j)} |o|$$

By applying our search strategy lm_2 in the running example (see Figure 5.4), we found that the landmark cardinality of abstract task flaw T_1 is $|O(T_1)|_{LT} = |T_{11}, T_{12}| + |T_{21}| + |t_{31}, T_{32}|$. Then, the cardinality value of task T_1 is 5. Assume the cardinality value of abstract task T_2 is 3. Then, the lm_2 strategy will prefer to expand abstract task T_2 before decomposing abstract task T_1 because $|O(T_2)| < |O(T_1)|$.

As we did for the landmark-aware strategies lm_1 above, we now define a variant for the heuristic

landmark-aware strategy lm_2 that examines the transitive closure of optional sets.

5.3.4 Landmark-aware Strategy(lm_2^*)

Definition 20 (Landmark-aware strategy lm_2^*)**.** *Given a plan $P = \langle TE, C\rangle$ where $C = \langle \prec, VC, CL\rangle$, let $t_i(\bar{\tau_i})$ and $t_j(\bar{\tau_j})$ be ground instances of two abstract tasks in TE that are compatible with the (in-) equations in VC and that are referenced by two abstract task flaws f_i and f_j, respectively, that are found in plan P. Let a given landmark table LT contain the corresponding entries $\langle t_i(\bar{\tau_i}), M(t_i(\bar{\tau_i})), O(t_i(\bar{\tau_i}))\rangle$ and $\langle t_j(\bar{\tau_j}), M(t_j(\bar{\tau_j})), O(t_j(\bar{\tau_j}))\rangle$.*

The modification ordering function lm_2^* then orders a plan modification ω_i before ω_j ($\omega_i \prec \omega_j$) if and only if ω_i addresses flaw f_i, ω_j addresses flaw f_j, and

$$\sum_{o \in O^*(t(\bar{\tau})i)} |o| < \sum_{o \in O^*(t(\bar{\tau})j)} |o|$$

For example, by applying our search strategy lm_2^* in figure 5.4 we found that the landmark cardinality of abstract task flaw T_1 is $|O(T_1)|_{LT} = |T_{11}, T_{12}| + |T_{21}| + |t_{31}, T_{32}| + |T_{111}, T_{112}| + |T_{113}| + |t_{114}, t_{115}| + \cdots + |t_{11311}, t_{11312}|$. Then, the cardinality value of task T_1 is 23. Assume the cardinality value of abstract task T_2 is 26. Then, the lm_2^* strategy will prefer to expand abstract task T_1 before decomposing abstract task T_2 because $|O(T_1)| < |O(T_2)|$.

Then, the lm_2^* strategy will prefer to expand abstract task T_1 before decomposing abstract task T_2 because $|O(T_1)| < |O(T_2)|$.

Since the landmark information can be extracted from any input domain model and problem in an automated pre-processing step, the above strategies are conceptually domain- and problem-independent heuristics. In addition, our landmark-aware strategies are intended to operate on their own local view on the planning domain and problem, that means, the remaining parts and modules of the hierarchical planning system do not necessarily have to adopt the landmark-centered view and to incorporate an accordingly reduced domain model. As a consequence, the presented

strategies can be easily combined with any kind of hierarchical or hybrid search schemata as, e.g., shown by [8] and they are completely compatible with other pre-processing techniques. It is furthermore worth noting that this also implies that our strategy principles can be translated into search controllers for any hybrid planning system.

5.4 Experimental and Empirical Analysis

Since our strategies merely specify the order in which a planner will prefer to decompose tasks, they have no affect on the planner's soundness and completeness. However, they do affect the planner's efficiency. In order to quantify the practical performance gained by the landmark-aware strategies, we conducted a series of experiments on two distinguished hierarchical planning domains: the *UM-Translog* domain and the *Satellite* domain.

As mentioned before, we ran our experiments on the set of planning problems with a lot of planning strategies from literature in order to measure the run time behavior of our planner in terms of the size of the average search space*(Space)* and CPU-Time*(Time)*. The SHOP and UMCP strategies denote strategy function combinations that simulate the respective search procedures, all other strategy implementations use FMF as the plan selection strategy.

5.4.1 Evaluation Results in the UM-Translog Domain

The evaluation results are shown in Table 5.1: lm_1, lm_1^*, lm_2, and lm_2^* do outperform the other strategies on practically all problems in the UM-Translog domain (See Table 5.1) in terms of both size of the explored search space and computation time. This is quite surprising because the landmark table does not reveal any information about causal dependencies on the primitive task level and the strategies hence cannot provide a focused guidance. An adequate selection of the decomposition refinements obviously pays off well enough to compensate for random choice on the

causality issues. Another interesting facet is that the strategies lm_1^* / lm_2^* being the better informed heuristic while repeatedly performing worse than lm_1 / lm_2. Furthermore, the same anomaly occurs when comparing lm_2 / lm_2^* with the more abstract but also more successful lm_1 / lm_1^*. We suppose these phenomena result from two sources: First, the random choice of ground candidates for the lifted task instances is relatively unreliable and this effect gets amplified by traversing along the landmark closures and into the primitive task level. Second, the most important choice points are on the early decomposition levels, i.e. once a method has been chosen for implementing the transport, this refinement puts more constraints on the remaining decisions than the strategy can infer from the feasibility analysis underlying the landmark table.

5.4.2 Evaluation Results in the Satellite Domain

On the *Satellite* domain our landmark-aware strategies show only poor performance (See Table 5.2), as there is hardly any landmark information available due to the shallow decomposition hierarchy of this domain and any landmark centered strategy is bound to be uniformed given limited landmark information.

In general, the experimental results indicate that the novel strategies outperform their conventional counterparts on practically all problems if the decomposition hierarchy of the underlying domain is of non-trivial depth.

Table 5.1 This table shows the impact of the deployed modification selection strategies on the planning problems of the *UM-Translog* domain. The best result for a given problem is emphasized bold, the second best bold and italic.

Modification	Hopper Truck		Auto Truck		Airplane		Reg. Truck-2_Region		Reg. Truck_1		Reg. Truck_2		Flatbed Truck	
f^{ModSel}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
DA	144	352	644	2077	172	620	114	257	148	352	723	2560	99	237
DU	101	224	459	1304	643	2134	160	460	117	258	–	–	1047	2601
HZ	55	121	197	527	345	1323	55	117	55	137	–	–	159	399
lm_1	***52***	***111***	**133**	**329**	142	441	62	135	***53***	**122**	***291***	1172	63	155
lm_1^*	**51**	**109**	***135***	462	189	676	**52**	**112**	65	142	**266**	***1162***	**61**	**144**
lm_2	62	162	***135***	464	***104***	***320***	***53***	123	55	151	339	**1128**	109	315
lm_2^*	124	340	146	489	114	436	57	148	**51**	**122**	305	1318	110	308
LCF	55	118	155	470	247	798	78	173	127	222	327	1278	***62***	179
EMS	147	295	405	976	784	2517	127	262	114	235	–	–	1571	3797
SHOP	89	212	164	***433***	150	450	106	241	83	190	926	4005	98	257
UMCP	58	122	156	474	**70**	**215**	55	***113***	57	***127***	308	1263	63	***149***

Modification	Armored R-Truck		Reg. Truck-3_Location		Auto Traincar		Auto Traincar-bis		Mail Traincar		Refr. Reg. Traincar	
f^{ModSel}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
DA	120	359	239	562	–	–	184	705	641	2031	588	1958
DU	75	201	1508	4079	–	–	1390	4018	424	1090	307	775
HZ	122	355	191	473	–	–	701	1616	81	224	***76***	***196***
lm_1	***71***	***177***	145	**374**	**158**	**596**	183	608	**75**	**184**	**72**	**180**
lm_1^*	**61**	**155**	154	430	304	1473	***158***	***543***	***78***	***205***	89	212
lm_2	73	199	***141***	469	420	1519	211	888	84	248	91	256
lm_2^*	81	228	**137**	413	367	1446	**142**	**511**	87	238	84	226
LCF	86	198	162	463	–	–	227	926	79	209	90	225
EMS	113	269	211	507	–	–	2558	6447	879	1806	500	1048
SHOP	95	227	146	***406***	–	–	247	963	121	274	173	353
UMCP	75	172	177	506	***220***	***739***	161	546	92	229	90	244

Table 5.2 This table shows the impact of the deployed modification selection strategies on the planning problems of the *Satellite* domain. The best result for a given problem is emphasized bold, the second best bold and italic.

Modification	1Obs-1Sat-1Mod		1Obs-2Sat-1Mod		2Obs-1Sat-1Mod		2Obs-1Sat-2Mod		2Obs-2Sat-1Mod		2Obs-2Sat-2Mod	
f^{ModSel}	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**	**Space**	**Time**
DA	**56**	**60**	68	78	782	1131	832	1301	2186	6841	**142**	***175***
DU	100	107	139	150	–	–	–	–	–	–	–	–
HZ	***61***	**60**	57	62	1281	4764	–	–	1094	1338	871	1114
lm_1	73	80	194	208	***560***	***652***	***352***	***400***	693	***785***	295	362
lm_1^*	78	85	**34**	**37**	847	969	1803	2569	739	813	619	1228
lm_2	78	86	128	140	4890	5804	**200**	**251**	–	–	483	965
lm_2^*	73	80	91	99	–	–	1905	2553	–	–	***146***	**161**
LCF	86	93	71	77	1120	1338	3022	4069	***407***	**701**	–	–
EMS	65	64	47	53	1586	2608	–	–	1219	1579	–	–
SHOP	62	66	105	111	**138**	**155**	–	–	1406	1780	–	–
UMCP	83	91	***36***	***41***	883	1035	1558	1894	**278**	1097	1062	1270

Chapter 6

Hybrid Multi-agent Planning

All planners that have were been introduced in chapter 2 study the problem of constructing plans for a single agent. They define an agent as an entity that is able to make changes in its environment. Today, the complexity of real world domains led AI researchers to establish algorithms more closely matching realistic planning environments in which planning activities are often distributed and both plan generation and plan execution happen concurrently. Therefore, several researchers from different fields such as cognitive science, economics, etc, discussed different issues in multi-agent based planning, which are considered to be major and important issues in the field of distributed AI. Therefore, we will describe in this chapter what multi-agent planning is as well as illustrate common multi-agent planning approaches. Then we will introduce a novel hybrid approach that integrates a multi-agent based planning approach with our landmark pre-processing technique in the context of hierarchical planning [97].

6.1 Multi-Agent Planning

Multi-agent planning has been used to solve large planning problems. It works by splitting the given planning problem into sub-problems. Each sub-problem is solved individually in order to

produce a solution, the so-called *sub-plan*. Then, these sub-plans have to be combined in order to construct a final solution plan for the original planning problem [33]. It is important to note, that the multi-agent planning is not only reasoning about actions but also involves some fundamental questions about the role of the agents in the planning society. Therefore, in the next section, we will discuss the characteristics that distinguish multi-agent planning from single-agent planning.

6.1.1 Characterization of Multi-agent Planning

Multi-agent planning *(MAP)* systems have several characteristics: First, they consist of a set of agents, where each agent has various skills and communication which may be limited. Second, the set of agents has a set of tasks. These tasks need to be decomposed into a set of sub-tasks and the sub-tasks to have to be assigned appropriate agents. Finally, each agent has partial knowledge and has limited resources to produce a local plan. To this end, a MAP system should perform the following interleaved phases in order to solve a planning problem in a distributed paradigm:

1. **Decomposition phase :** One of the most important phases in MAP systems is the decomposition phase. It discusses how the original planning problem $\Pi = \langle I, O, G \rangle$ or global tasks can be decomposed into sub-problems $\Pi_i = \langle I_i, O_i, G_i \rangle$, where $O_i \subseteq O$, $I_i \subseteq I$, $G_i \subseteq G$, and $\bigcup_{i=1}^{n} \Pi_i = \Pi$ and n is the number of sub-problems. The symbol O represents the set of actions that can be applied to the world state in order to transform the current world state to a new world state. I and G are sets of facts which represent initial and goal states respectively. The decomposition phase is considered the crucial issue in MAP, because it determines the efficiency of the resolution process and complexity of the plan combination technique. The complexity of plan combination is determined by identifying the interactions that may appear between plan steps in the sub-plans of different agents. In general, there are two different types of interactions: **negative interactions** which occur between two actions when the post-condition of one action violates *(negates)* the post-condition of another action or when one

action deletes the pre-condition of another action, and **positive interactions** which occur between two actions when both need the same pre-condition and at least one of them does not remove it or one action can be used to support the pre-condition of another action. Intuitively, the negative interactions among the different sub-plans cause great complexity in the process of combining sub-plans. We have to be careful when designing the decomposition, because it may cause that solving sub-problems *(i.e., merging sub-plans)* is more costly than solving the original planning problem itself. Due to the difficulty of finding good automatic decompositions, the most MAP systems use manual decomposition of the problem or decompose it automatically by splitting a compound goal into sub-goals. One of the common techniques for decomposing planning problems into sub-problems with limited interactions is the interaction graph technique. It can be built automatically in low order polynomial time [98].

2. **Task-allocation phase :** The problem of allocating sub-problems to an appropriate agent are handled during the task-allocation phase. There are several techniques that discuss the task assignment as a distributed way in order to give agents a higher degree of autonomy. For example, in the auction technique, each agent make one bid and then sub-goals are assigned to the highest bidder [99]. The decomposition and task allocation phases are often merged into one phase due to the existing relationships between the agent's capabilities and task decomposition.

3. **Coordination phase :** Reducing redundant in planning tasks, increasing the system performance, and eliminating conflicts that prevent agents from achieving their goals are the objectives of the coordination phase. Therefore, the coordination phase is the process of managing dependencies between plan steps or sub-plans in different agents. Note that the conflict resolution is the essential and necessary part to achieve the coordination process. Of course, there is a difference between the conflict resolution and coordination process.

The coordination process is a continuous process in the system, while conflict resolution is triggered once a conflict is detected. Most researchers dealing with MAP have focused on coordination, communication, cooperation, and negotiation processes among various agents.

4. **Individual planning phase :** In this phase, the problem of accomplishing sub-problems individually with different agents is discussed. Generally, in muli-agent planning any planning technique can be used by different agents in order to produce a solution plan.

   Attention should be paid to the fact that the coordination process and the individual planning phases are tightly interleaved. The coordination process requires both an adequate plan representation and efficient interaction methods between agents. The process of solving interactions among agents is based on exchanging information between agents and each agent updates its own plan according to the exchanged information. In general, the coordination process has two advantages: it eliminates negative interactions and takes advantage of positive interactions such as handling redundant actions. Consequently, the coordination phase is done before, after or during individual planning phase as follows:

   - **Coordination phase before planning phase**

     In this case, all the dependencies between agents are discovered and resolved before any planning takes place. One of the most well-known techniques in this case is a social law technique [100]. The social law is a mechanism for resolving conflicts over shared resources among a group of agents. It propagates general rules that each agent has to follow in order to resolve conflicts among agents. If these rules have a large enough scope to prevent such conflicts occurring among agents, each agent can achieve its goals by executing its plan according to these laws without coordination with other agents. It is difficult to define social laws to achieve all efficient alternative actions for each agent. It is also difficult to define social laws to avoid cases where agents are unnecessarily coordinating in case no conflicts exist. However, social laws reduce

the costs of communication, planning, and coordination time because no exchange of information between agents is needed.

- **Coordination phase after planning phase :** In this approach, the coordination process is constructed after the individual planning process has been completed. It assumes that agents work independently on their sub-problems to achieve solution plans for these sub-problems. Therefore, the conflicts between these sub-plans are resolved after the planning process by exchanging and revising parts of the sub-plans.
- **Coordination phase during planning phase :** Coordination during planning is different from other coordination methods *(either coordination before or after planning)* where planning and coordination steps are interleaved [101]. In this method, agents continuously work by exchanging planning information in order to reach to a consistent joint solution plan. The main difference to the coordination process before planning phase is that interferences between individual partial plans are resolved before each agent produces its own partial plan. While in the coordination process during planning phase, all agents are cooperative in the sense that they have the ability to exchange planning information with other agents and change their current plans if necessary.

5. **Synthesis or joint plan execution phase :** This discusses the problem of plan synthesis and execution. It involves the re-planning due to a failure and implies to backtrack to phases 1, 2 and 3. It is considered slightly different topic.

In general, as we show in figure 6.1, the standard multi-agent planning approaches can be categorized into centralized multi-agent planning and distributed multi-agent planning. In the next section, we will briefly discuss these different categories.

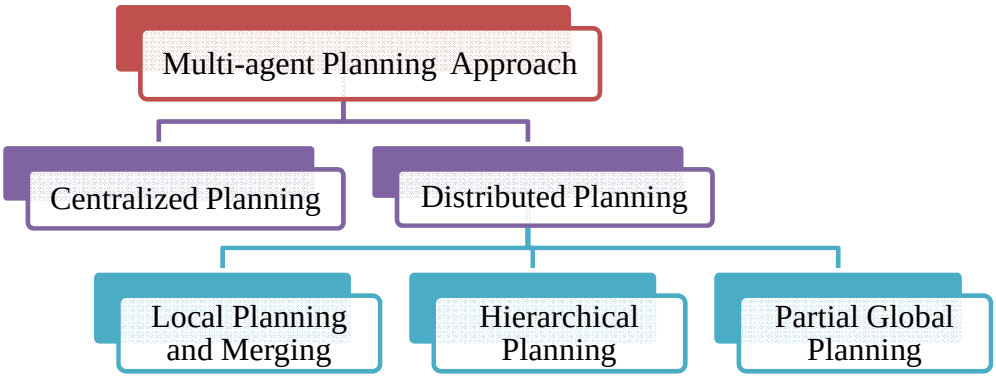


Figure 6.1 Multi-agent planning taxonomy.

6.1.2 Centralized Multi-agent Planning

In general, any of the techniques in classical state-based planning can be applied to the centralized multi-agent planning paradigm. For simplicity we will say centralized planning instead of centralized multi-agent planning. As shown in figure 6.2, the process of centralized planning consists of three steps:

1. **Pooling goals** – The agents' goals are collected into a single set of joint goals and then passed to the planning system. In case the set of goals are independent *i.e. non conflicting*, the collecting process is very simple. It is done by unifying the different goals. Otherwise agents may have to select a subset of goals that are achievable. Rosenschein et al. [102,103] introduced bidding and negotiation protocols that can be applied to such problems. Their emphasis is on efficiency, simplicity, and fairness. Domain dependent aspects of negotiation, such as agent preferences for plans that achieve a maximum number of goals or that are robust with respect to environmental events, are typically encoded as measures of cost or worth of goals and plans, which are fed into the protocols in a domain independent way. Note that agents might not be able to determine goal conflicts until some planning has been performed *i.e. collecting goals has to be revisited after planning has started* [104].

2. **Planning system** – In order to generate a global plan for the collected goals in the previous step, any single agent planning technique can be used.

3. **Synchronized Plans** – The global plan should be split into a set of coordinated individual plans. Individual plans are distributed among relevant agent executives [105, 106]. Therefore, this technique is sometimes called *centralized planning for distributed plans*.

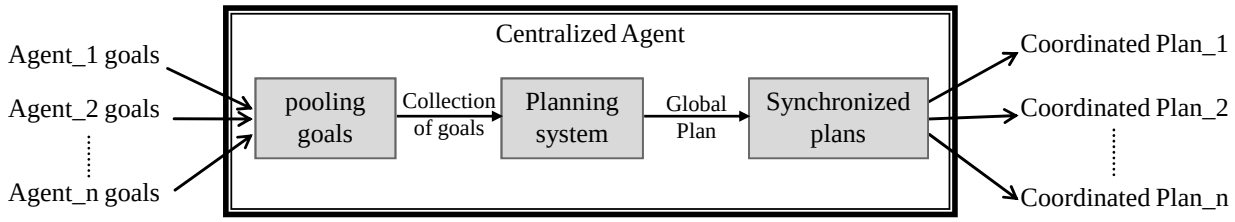


Figure 6.2 The centralized multi-agent planning process.

Georgeff [107] is the first researcher who proposed synchronization among individual plans. He assumes that the set of original agents' goals are collected and the global plan which solves the set of goals is generated. Then, the problem is how to synchronize among individual plans. To this end, he inserts primitives synchronization into individual plans in order to avoid unsafe *(i.e., interaction)* situation. His method collects individual plans and analyzes them to identify potential interactions, such as conflicts between the agents over limited resources. For example, in case of actions a_i and b_j are the next actions to be executed by agents A and B respectively. These actions can be executed in parallel if the pre-conditions and post-coditions of each are consistent at the same time, then these actions can *commute* and are essentially independent. Otherwise, all unsafe situations are collected in order to create *critical regions*. Finally, communication commands such as *Semaphore* are inserted into individual plans to solve synchronization between agents.

Cammarata et al. [108] have also introduced a centralized planning system. Their system is developed for an *Air-traffic* control domain. It depends on selecting one agent from a set of agents

and considering it as a coordinator or centralized agent. Afterwards, the centralized agent collects different goals from different airplanes in order to constructs a multi-agent plan that achieves all of the requirements of the airplanes' goals, as well as including those actions that specify coordination between airplanes to avoid collisions.

In general, centralized planning approaches have some advantage such as: (1) planning techniques like SHOP [72] or PANDA [46] can be used without any modification because the planning process itself is located in a single agent, (2) There is no additional communication cost in the planning process because the planning process is executed in a single agent. Nevertheless, centralized planning approaches have drawbacks such as: (1) All planning process steps are executed sequentially, none of them are performed in parallel, so more time for execution is needed, (2) All knowledge is located in one agent *i.e. central point of failure*.

6.1.3 Distributed Multi-agent Planning

In contrast to the centralized planning paradigm, the distributed multi-agent planning paradigm does not have a single node with oversight over network activities. The planning process itself is distributed among a set of agents. Therefore, the process of detecting and resolving interactions between agents is much more difficult. As depicted in figure 6.1, there are three different directions in the distributed planning paradigm: Local Planning and Merging, Distributed Hierarchical Planning and Partial Global Planning.

Local Planning and Merging

Local planning and merging is a popular approach in the context of distributed planning. The idea of this approach is quite simple. As shown in figure 6.3, individual plans are created by different agents each having just a local view of the overall problem. Therefore, the conflicts among these individual plans have been discovered and resolved by adding ordering constraints, adding variable

bindings and eliminating redundant actions in order to generate a single coordinated global plan using a merging agent. In local planning and merging, several individual plans can be merged into a single global plan which is considered a general solution for the original planning problem or the global plan can be split into a set of coordinated individual plans as we have seen with the centralized approach. Each individual plan involves a set of tasks for achieving the goals of a single agent, while the global plan is a combination of individual plans of several agents. In the plan separation phase, the global plan is separated into a set of coordinated individual plans and passed to the relevant executive agents. Note that synchronization information has to be inserted into the individual plans to ensure successful execution.

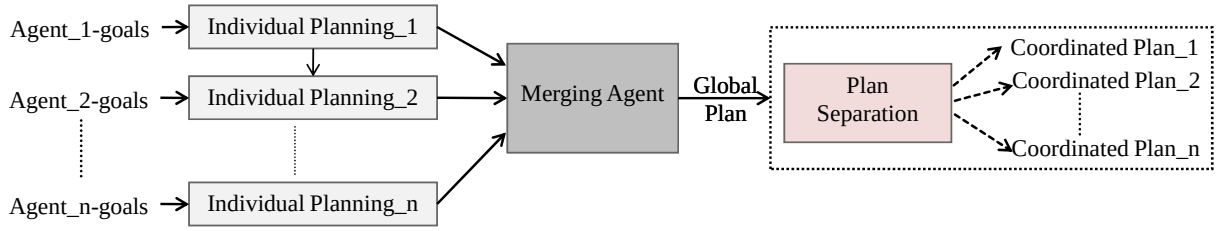


Figure 6.3 The local planning and merging structure

In general, merging plans has several meanings in the multi-agent planning field:

- Merging redundant tasks that achieve the same effect in separate sub-goals into a single plan step.
- Incorporating individual plans for new goals into a plan that achieves current goals.
- Distributing the goals of the planning problem to agents that cooperatively solve them.
- Resolving conflicts among the individual plans of separate agents.

Merging plans in a single agent has a rich history. Many researchers have developed methods for preventing or exploiting redundancies in the plans of a single agent either during the actual planning process or when integrating the separated sub-plans.

Ephrati et al., proposed a technique to merge distributed individual plans [109]. The idea of their technique depends on a dynamic generation of alternative plans that identify the global plan. In each step, every agent adds information about its own current plan to the set of global plans, and hence the current set of global plans is refined according to this information in order to produce a new set of global plans. This process is repeated recursively until the solution global plan is found.

Later, Yang discussed the problem of merging several independent plans that were established by different agents in order to generate a single plan [110]. Yang introduced a formal definition of plan merging states, where the set of plan steps Σ can be merged with a plan step Ω *(meaning* Ω *can replace* Σ *in the plan)*, if the union of pre-conditions of Ω subsume those of Σ, the post-conditions of Ω subsume the useful effects of Σ, and the cost of Ω is less than the cost of Σ. **Useful effects** are defined as the post-conditions that establish conditions that are pre-conditions of other plan steps in the overall plan. Yang's definition is flexible, it allows for any single plan step in a partial order plan to merge with any possible subset. However, Yang's merging criterion suffers from the fact that there are not always intuitive groupings of plan steps in an arbitrary partial-order plan and that the number of possible groupings is quite large. His merging algorithm is complex and incomplete.

Tsamardinos et al. [111] presented a new plan merging algorithm for partial order plans that can handle simple conditional branching, quantitative time, and actions with temporal duration. Their algorithm merges new goals into existing single agent plans in three phases. First, it uses a new data structure, the so-called **C**onditional **S**imple **T**emporal **N**etwork *(CSTN)*, which was extended from simple temporal networks [112] in order to identify conflicts between plan steps. CSTN consists of a set of nodes that represent the start and end points of the plan steps and a set of arcs that represent the temporal constraints. Second, their algorithm uses the Yang algorithm [110] to

solve conflicts between plan steps. Finally, CSTN has been built again in order to check whether the proposed resolutions achieve all temporal constraints and ensure that there are no more conflicts in the resulting plans.

Iwen et al, [113] developed GRAPHPLAN planner [27] further into a distributed planning system by decomposing the initial and goal states into sub-goals. The decomposition process depends on a new representation, the so-called *interaction graph* [98].

In the planning problem, the interaction graph is an undirected and simple bipartite graph. It consists of a set of vertices which represents the propositions that are true in the initial state and each proposition that needs to be true in the goal state. Hence, two vertices are connected by an edge if the corresponding propositions have a common object. This graph is decomposed into sub-graphs by detecting the disconnected parts. After that, each sub-problem is allocated to an agent, then each agent uses GRAPHPLAN to solve its own sub-problem and generate individual solutions for its own sub-problems. Once all sub-problems have been solved the global plan is obtained by taking a union of sets of actions at respective steps in individual plans. In this technique, conflict resolutions are resolved by applying a progression search technique, where the action selection is restricted by the actions from the corresponding action levels in the individual planning graph.

Adriann tel Mors et al, [114] proposed a new method that guarantees that: (1) the set of agents work independently, (2) the individually constructed plans can be joined into a general or global plan without refinement in any individual plan *i.e., there is no need for additional re-planning for any agent*. Their method occurs at the task level and is independent from the planning process and allows reusing the existing single-agent planning software in multi-agent planning. They solved the coordination problem by adding new constraints (Δ_i) to each agent i to ensure that all feasible individual plans constructed by agents can always be joined in order to produce a feasible global solution plan for the given planning problem. Although this method handles coordination and planning separately, it leads to the addition of more precedence restrictions on the set of tasks.

Therefore, the feasibility of finding cheaper plans is reduced.

In general, local-planning and merging techniques suitable for both competitive and cooperative agents to improve plans by removing incompatibilities and exploiting positive interactions. The quality of the resulting plan is dependent upon the cooperativeness of the participating agents and the quality of the individual plans. However, local-planning and merging has two major drawbacks: (1) The success of the plan merging process is dependent on the process of merging the individual plans into a coordinated joint plan. Sometimes, this process is impossible due to lack of time and memory capacity or unsolvable conflicts between plan steps in different individual plans. (2) The isolation between agents during the planning process prevents agents to be benevolent.

Hierarchical Planning

Hierarchical planning is an alternative approach for cooperative distributed planning that exploit the hierarchical structure of the plan space to perform distributed hierarchical planning. The main advantage of this approach is that the conflicts may be detected at more abstract levels which lead to pruning big portions of the more detailed search space.

The general approach in distributed hierarchical planning relies on attaching to each agent model of other agents plans. Corkill [19] has studied interleaved planning and merging in a distributed version of NOAH[1] planner. In the NOAH system, each agent has a picture of other agent's so-called *MODEL node* and message passing was used to synchronize plan execution. Each agent generates his plan level by level such that each agent builds local plans at one level of detail, and then constructs suitable models of each other's by exchanging the shared resources needed for their goals. The agents proceed to the next level when the conflicts in their current plans have been resolved.

Clement et al. [115, 116] have developed a new method to detect and resolve conflicts not only at

[1] NOAH $\Longrightarrow$ **N**ets **O**f **A**ction **H**ierarchies

the primitive task levels, but also at the abstract levels. This is done by computing and reasoning about summary information. Summary information is a process of computing abstract task resources which are derived from decomposition methods and sub-tasks to detect possible conflicts between abstract tasks in non-recursive propositional HTN domains. Therefore, summary information is defined as knowledge about pre-conditions, in-conditions, and post-conditions for a plan p. Summary pre-conditions of plan p are a set of all pre-conditions of sub-plans that must be met to execute plan p successfully. Summary post-conditions of a plan p are the set of all effects that are generated by sub-plans which must be performed before executing plan p. While, summary in-conditions of a plan p are any proposition that must hold within the interval of the execution time of plan p. Clement used summary information as a heuristic guide in order to avoid branches of search space that lead to inconsistent or sub-optimal plans. Their planning mechanism called ***C**oncurrent **H**ierarchical **P**lans (CHIPs)*.

Afterwards, Desjardins et al. used SIPE-2 in order to produce a new system, the so-called *DSIPE-2* [117]. This development was done by focusing on efficient communication among planners and creating a common partial view of sub-plans, where the coordinator agent distributes sub-goals among the planning agent in the system. Each planning agent expands its own sub-plans separately, while relevant information and constraints are shared between them. At the end, the coordinator agent merges the generated sub-plans together to produce a general plan for the given planning problem.

A novel approach that suggests synergy between hierarchical planning agents is introduced by Jeffery et al. [118]. They define **synergy** as a process of identifying the overlapping or subsuming effects between hierarchical planning agents when trying to achieve their goals. The reasoning of synergy concentrates on abstract plan steps instead of reasoning between primitive plan steps. They used Clement's algorithm that propagates such conditional information from the bottom up the plan hierarchy in order to identify relationships between abstract plan steps of different agents

[119]. Later, Jeffery et al, [120] proposed a general framework to extend the partial order causal link plan representation to multi-agent planning and deal with the coordination problem as a form of iterative repair of plan flaws that cross agents.

Recently, Hisashi [121] proposed an architecture that arranges the set of agents in a stratified form as parent and children planning agents. The parent planning agent and its child planning agents work together to achieve the goal. This is done by producing a rough plan for a goal by the parent planning agent and producing detailed plans for sub-goals by child planning agents. Stratified multi-agent planning reduces the search space of HTN planning because once the parent planning agent makes plans and selects a plan from them, the other alternative plans of the parent planning agent are not taken into consideration. The child planning agents produce a detailed plan for the selected plan which is constructed by the parent agent.

Partial Global Planning

In partial global planning, agents communicate parts of their individual plans in order to construct plans that are partially global. These partially global plans specify the relations between actions and can be used by agents to adapt their individual plans to other agent's actions. This direction allows exchanging crucial information about individual plans between agents in order to prevent conflicts.

Mathijs et al. proposed an incomplete algorithm to study the interleaving between planning process and coordination process [122]. Their algorithm provides a distributed way for a set of agents to construct coordinated plans for a large set of autonomous multi-agent planning problems through exchanging a resource during the planning process by propagating two artificial actions, the so-called **get** and **put** actions. These two actions organize the way to transfer resources from agent a_i to agent a_j, where *get action* represents receiving resources from another agent and *put action* represents sending resources to another agent. In the final solution plan these actions only come

in pairs. Consequently, the resources that are exchanged are deleted from the state of the agent a_i and added to the state of the agent a_j.

Krogt et al. introduced a new method to coordinate plans in different agents without exchanging explicit information or constructing a global set of constraints, but by combining a propositional plan repair algorithm for each agent using a blackboard that auctions sub-goals on behalf of the agents [123]. In contrast to a batch system which pushes all goals at the same time, this method channels the goals *one-by-one* through the plan repair system that observes change in the world state and updates its state by re-planning according to the current information. With the *one-by-one-goal* strategy, failure of adding a goal to the plan is discovered easily and put up for auction. The strategy *One-by-one-goal* accelerates the process of determining the goal which should be announced for auction, but positive interactions between goals cannot be exploited easily. It takes more time than constructing a one shot plan for all goals.

Different aspects of coordination and cooperation problems of multiple agents that have individual goals and operate in the same environment have been studied [124]. In the coordination process, each agent has the ability to generate and execute its plan independently and coordination is needed to solve harmful interactions between individual plans in different agents. The set of agents are coordinated by exchanging messages that contain the candidate sub-plan of a joint plan. The receiving agent checks the consistency between its own plan that achieves the local goal and the incoming plan. After that, the received agent responds with "fail", if the candidate plan cannot be part of a joint plan, or its sends a message indicating that the candidate plan can be a part of joint plan. The coordination algorithm terminates, when the agent has received replies to all its own sub-plans. It does not have more proposed plans and does not expect any other sub-plans from other agents.

In the cooperation process, each agent asks some other agent to construct pre-conditions of actions that appear in its plan. The cooperation process between agents is very similar to the coordination

process with the main differences being in form and meaning of the exchanged messages. These messages consists of the current plan of the sender agent and the request to achieve some resources. Consequently, the receiver agent looks for a plan to achieve its own goal and achieves the requested resources as well as checks the consistency between local individual plan and the received plan.

6.2 Hybrid Multi-agent Planning

Although several approaches have been constructed for multi-agent planning, solving large planning problems is still quite difficult. In this section, we present a novel approach called *Hybrid Multi-agent Planning* which integrates the landmark pre-processing technique in the context of hierarchical planning approaches with a multi-agent planning approach.

Figure 6.4 depicts the components of our hybrid multi-agent planning architecture. It consists of the pre-processing agent, the planning agents, and the shared constraints set. The pre-processing agent analyzes a given hierarchical planning problem by systematically analyzing the ways in which relevant abstract tasks can be decomposed. The planning agents construct a set of hierarchical planning problems *(Clusters)*. Each agent generates a solution plan *(individual plan)* individually. After that, the set of individual plans are merged in order to generate a final solution plan. All planning agents have complete knowledge about the initial state of the planning problem. The **S**hared **C**onstraints set (SC) includes a set of constraints between the set of abstract tasks in different clusters. The shared constraints set (SC) provides a great service for the process of merging individual plans. In the following section we will illustrate the key features of our agents.

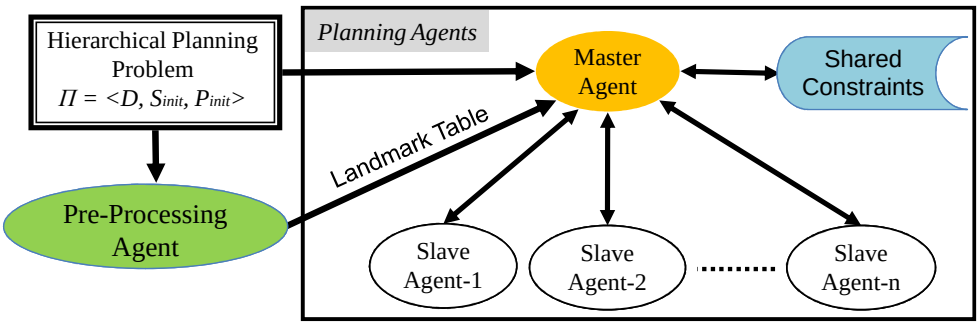


Figure 6.4 Hybrid multi-agent based planning architecture

6.2.1 Pre-processing Agent

A pre-processing technique will be used to perform some pruning of the search space before the actual search is performed in order to reduce the planning effort. Very recently, we introduced a landmark technique which restricts the domain and problem description of a Hierarchical Task Network (HTN) problem to a smaller subset, because some parts of the domain description might be irrelevant for the given planning problem [71, 125, 126].

Since the landmark extraction algorithm (Algorithm 3) has already been described in detail before, we will only briefly sketch the outcome of landmark extraction algorithm, the so-called *landmark table*, which is the central means for exploiting landmark information.

Let $\mathcal{T} = (V_M, V_T, E)$ be the TDT after methods that contain infeasible tasks have been pruned. V_M is the set of method vertices, V_T is the set of task vertices, and E is the set of edges connecting the methods with its tasks. Then, for each abstract task $t(\overline{\tau}) \in V_T$, the landmark table LT contains an entry $\langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle$, where mandatory sets $M(t(\overline{\tau}))$ and optional sets $O(t(\overline{\tau}))$ are

calculated as follows:

$$M(t(\overline{\tau})) = \{t'(\overline{\tau}') \in V_T \mid \text{f.a. } \langle t(\overline{\tau}), \langle S, \prec, V, CL\rangle\rangle \in V_M \text{ with } t'(\overline{\tau}') \in Ground(TE, V)\}$$

$$O(t(\overline{\tau})) = \{Ground(TE, V) \setminus M(t(\overline{\tau})) \mid \langle t(\overline{\tau}), \langle S, \prec, V, CL\rangle\rangle \in V_M\}$$

How a landmark table entry for an abstract task looks like, in particular, how many tasks are contained in the mandatory set M, clearly depends on the pruning that has been performed before the calculation of M and O.

Once the landmark table LT has been constructed, the pre-processing agent terminates itself after sending landmark table LT to the master agent. The information about landmarks can be exploited in two ways: first by reducing the domain model by ignoring infeasible method decompositions or, more precisely, by transforming a universal domain model into one that includes problem-specific pruning information [71]. 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 [125,126]. In our hybrid multi-agent planning we will focus on the former way in order to construct individual solution plans with each agent.

6.2.2 Planning Agents

Our planning scenario includes a set of planning agents. The planning agents integrate in order to cooperate a general solution plan for the given hierarchical planning problem. To this end, the set of planning agents are divided into two different types: a master agent and a set of non-cooperative agents, the so-called slave agents.

Master Agent

The master agent takes a hierarchical planning problem Π and a landmark table LT as input and generates the final solution plan (Algorithm 7). To this end, in line 4, a given planning problem Π is decomposed into a set of clusters *(i.e., sub-problems)*. We will come back to the decomposition process later in this section. In lines 6 and 7, the master agent initiates a number of slave agents based on the number of clusters. Then, it distributes these clusters among slave agents in order to construct individual solution plans for these clusters. Afterwards, in lines 8 to 11, the master agent receives a `Termination-Message`[2] from the slave agent. The individual plan in the `Termination-Message` is preserved in a set, the so-called *IndPlan*. Then, the communication channel between master and slave agents is closed when the value of *Terminator* counter, which counts the number of `Termination-Messages`, equals to the number of the created clusters. After that, the master agent checks the components of the *IndPlan*. If there exists at least one empty plan in the set *IndPlan*, the master agent returns failure (lines 12 to 14). This means that one of the set of slave agents fails to find a solution for its sub-problem . Otherwise, in line 15, a merged plan *MPlan* is constructed by calling the `Merging` algorithm which combines the set of individual plans in *MPlan*. Finally, the master agent terminates successfully with the merged plan *MPlan* as a solution, if *MPlan* satisfies the solution criteria. Otherwise, it terminates with failure (lines 16 to 19).

The master agent breaks up a given planning problem Π into a set of clusters according to two different algorithms: `Dependent` and `Independent` algorithms.

In each iteration of the `Dependent` algorithm (Algorithm 8), a set of tasks that are not preceded by any other tasks are separated in one cluster. The set of constraints between tasks in different clusters is inserted in the shared constraints set (SC).

[2]Termination-Message includes the individual plan which is produced by slave agent.

Algorithm 7: *Master Agent Algorithm*

Input : $\Pi = \langle D, s_{init}, p_{init} \rangle$: Planning problem,

LT: LandmarkTable

Output: *Solution Plan* or *Failure*

1 *Terminator* $\longleftarrow 0$

2 Create an empty set $\Gamma = \langle \Pi_\gamma, SC \rangle$

3 Create an empty set, called *IndPlan*, to preserve the individual plans

4 $\Gamma \longleftarrow$ Dependent(Π, LT, Γ)

5 *NSA* $\longleftarrow$ Number of clusters

6 Initiates a set of slave agents

7 Distributes the set of clusters among the set of slave agents

8 **while** *(*Terminator $\neq$ NSA *)* **do**

9 **if** *(Termination-Message from slave agent is received)* **then**

10 *IndPlan* $\longleftarrow$ individual plan

11 *Terminator* $\longleftarrow$ *Terminator* + 1

12 **foreach** *(*$p \in$IndPlan*)* **do**

13 **if** *(p is empty)* **then**

14 **return *Failure***

15 *MPlan* $\longleftarrow$ *Merge(*IndPlan,SC*)*

16 **if** *(*MPlan *is a solution)* **then**

17 **return *MPlan***

18 **else**

19 **return *Failure***

Algorithm 8: `Dependent` *Algorithm*

input : $\Pi = \langle D, s_{init}, p_{init} \rangle$: Planning problem,

LT: LandmarkTable, $\Gamma = \langle \Pi_\gamma, SC \rangle$

output: Γ

$\Gamma = \langle \Pi_\gamma, SC \rangle \longleftarrow\ null$

if *(TE == $\emptyset$)* **then return** Γ

Select tasks te_i that are pre-request-free from p_{init}.

Create new cluster $\Pi_{\gamma_i} = \langle D, s_{init}, p_{\gamma_{init}}, LT_{\gamma_i} \rangle$.

Add these tasks te_i to partial plan $P_{\gamma_{init}}$ in cluster Π_{γ_i}.

Let $TE \longleftarrow (TE - te_i)$

foreach *(task $t \in te_i$)* **do**

  Attach the relevant information of the task t in the LT to the LT_{γ_i} in cluster Π_{γ_i}

  Add all constraints $(\prec_t, VC_t, CL_t)$ that relate task t with the other tasks in te_i to the partial plan $P_{\gamma_{init}}$ in cluster Π_{γ_i}.

  Let $VC \longleftarrow (VC - VC_t)$

  Let $\prec \longleftarrow (\prec - \prec_t)$

  Let $CL \longleftarrow (CL - CL_t)$

  Insert all constraints $VC_{\bar{t}}$, $\prec_{\bar{t}}$ and $CL_{\bar{t}}$ that relate task t with other tasks $\bar{t}$, where $\bar{t} \notin te_i$ to the shared constraints set SC.

  Let $VC \longleftarrow (VC - VC_{\bar{t}})$

  Let $\prec \longleftarrow (\prec - \prec_{\bar{t}})$

  Let $CL \longleftarrow (CL - CL_{\bar{t}})$

$\Gamma = \langle \Pi_\gamma, SC \rangle \longleftarrow\ \langle (\Pi_\gamma \cup \Pi_{\gamma_i}), SC \rangle$

return `Dependent`(Π, LT, Γ)

As shown in algorithm 8, the `Dependent` algorithm takes the planning problem $\Pi = \langle D, s_{init}, p_{init} \rangle$,

a landmark Table $LT = \langle t(\overline{\tau}), M(t(\overline{\tau})), O(t(\overline{\tau}))\rangle$ *i.e., the* LT *is computed by pre-processing agent*, and a set $\Gamma = \langle \Pi_\gamma,\ SC\rangle$ as input and computes the final set of clusters Γ. Note that a set Γ consists of a 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. SC is a shared memory that includes a set of constraints between tasks in different clusters Π_{γ_i}. In order to identify different clusters, `Dependent` algorithm runs recursively through all tasks of sub-tasks in the initial plan p_{init} until all tasks have been traversed. Each cluster $\Pi_{\gamma_i} = \langle D, s_{init}, p_{\gamma_{init}}, LT_{\gamma_i}\rangle$ includes a domain model D, an initial state s_{init}, a partial plan $p_{\gamma_{init}}$ which represents initial partial plan for cluster Π_{γ_i}, and LT_{γ_i} which represents relative pre-processing information of the set of tasks in partial plan $P_{\gamma_{init}}$.

Now we will have a look at how the set Γ is constructed by `Dependent` algorithm. First, the set Γ is initialized (line 1). Afterwards, the set of tasks te_i which are pre-request-free are collected (line 3). In lines 4 to 6 a new cluster Π_{γ_i} is created, and the set of tasks in te_i are added to the task network of $P_{\gamma_{init}}$ in cluster Π_{γ_i}. Then, the tasks te_i are removed from the task network TE of plan Π. For the current set of tasks te_i at hand, lines 7 to 16 illustrate iterative loops to extend the created cluster Π_{γ_i} by adding different constraints and then updating SC by inserting new constraints. For each task t in the set of tasks te_i, the pre-processing information in LT which is relevant to the current task t is attached to the current cluster Π_{γ_i}. This information in LT_{γ_i} will help the planner *(slave)* agent to reduce the planning effort in order to find the individual solution plan for its own cluster (line 8). In lines 9 to 12 the partial plan $p_{\gamma_{init}}$ in the current cluster Π_{γ_i} is extended by adding all variable VC, ordering $\prec$ and CL constraints that point to the relation between the current task t and other tasks in the set of tasks te_i. After that, these constraints are removed from the current plan P_{init} in Π. The shared constraints set SC and the current partial plan p_{init} are updated by inserting and removing all constraints that relate the task t to other tasks outside the set of tasks te_i respectively (lines 13 to 16). Finally, the current set Γ is updated by adding the current new cluster Π_{γ_i} to the set of clusters Π_γ as well as updating the shared constraints set SC (line 17). The

`Dependent` algorithm is called recursively with the modified plan and the updated Γ in order to inspect a new cluster (line 18).

On the other hand, the idea of the `Independent` algorithm depends on separating the set of tasks that are ordered together in the initial plan in one cluster, thus the shared constraints set (SC) becomes empty. Therefore, in order to split the planning problem Π into a set of clusters according to `Independent` algorithm, we will replace line 3 in algorithm 8 for 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.

As mentioned before, once the set of clusters has been generated either by `Dependent` or `Independent` algorithm, the master agent initiates a set 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 collects the individual solution plans which are generated by slave agents and starts to perform the merging process in order to construct a general solution plan.

Slave Agents

The slave planning agents are a set of identical agents working concurrently in order to solve the set of clusters which passed from the master agent. There is no communication among slave agents. As shown in the algorithm 9, each slave agent considers a sub-problem $\Pi_\gamma = \langle D, s_{init}, p_{\gamma_{init}}, LT_\gamma \rangle$ as input and returns *`Termination-Message`*. The *`Termination-Message`* encapsulates the individual solution plan p_γ or an empty plan which represents failure in its context.

Algorithm 9: *Slave Agent Algorithm*

Input : $\Pi_{\gamma_i} = \langle D, s_{init}, p_{\gamma_{init}}, LT_{\gamma_i} \rangle$: Cluster (sub-problems)

Output: `Termination-Message(`p_γ`)` : Message

1 $p_\gamma \longleftarrow$ `Refinement-Planning(`Π_{γ_i}`)`

2 **if** *(p_γ is a solution)* **then**

3 Send `Termination-Message(`p_γ`)` to the master agent

4 Deactivate itself

5 **else**

6 $p_\gamma \longleftarrow$ slave agent acts as master agent

7 Send `Termination-Message(`p_γ`)` to the master agent

8 Deactivate itself

Each slave agent implements the refinement planning algorithm (Algorithm 2) in order to generate its own individual plan(line 1). Our refinement planning 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. Each slave agent sends a `Termination-Message` which includes the individual solution plan (p_γ) to the master agent and terminates itself(lines 2 to 4).

Special attention should be paid to the fact that, if the refinement planning algorithm returns failure, the slave agent works as a master agent in order to split its own cluster into a set of new clusters and then initiates a set of new slave agents in order to find an individual solution for the new clusters (lines 5 to 8).

6.3 Merging Methodology

Of course, dividing a large planning problem into a number of smaller individual problems or sub-problems, solving them concurrently and then joining their individual solutions in order to generate a general solution plan for the original planning problem, can sometimes achieve high performance. The time taken to create individual plans can be significantly reduced because the joint problem is broken down into smaller components. An overall speed gain is achieved if the reduction in individual planning time is more significant than the time taken to merge the individual plans into a joint plan. Otherwise, if the individual plans have many redundancy and conflicting actions, the plan merging technique may be prohibitively expensive. The approaches which depend on the merging technique strongly depend on two factors: the ability of planning agents to produce individual plans without communication and the ability of the merging process to construct a general plan from the set of individual plans.

In general, the merging algorithms are limited because they can not add tasks to the joint plan. If the planning slave agents generate individual plans with unresolvable conflicts, the merging process could not be able to find a solution even if there is an alternative set of non-conflicting tasks that could be chosen. If the problem can not be solved by the merging process because of the conflicts that arise during the merging process, it is said to be a non-mergeable problem [127, 128].

Our merging methodology depends on the notion of `Fragments` which encapsulate individual plans with in their context. Our merging methodology proceeds in two processes (See Algorithm 10): First, dividing the set of individual plans *(i.e., which are produced by slave agents)* into a set of `Fragments` (line 3). Second, merging the individual plans in each fragment in order to produce a plan, the so-called *Merge-Fragment-Plan(MFP)* (line 5), and then merges all MFPs in order to construct a general solution plan(line 7).

Algorithm 10: *Merge Algorithm*

Input : $P_\Gamma = \{p_{\gamma_1}, p_{\gamma_2}, \cdots, p_{\gamma_n}\}$: Individual Plans,

SC: Shared Memory

Output: `Plan`

1 Create an empty set *MFPlans* to preserve the merged plans

2 MFP $\longleftarrow$ empty plan

3 Fragments $\longleftarrow$ `Divide(`P_Γ`, SC)`

4 **forall** *(*$f \in$ *Fragments)* **do**

5 MFP $\longleftarrow$ `Combines(`$P_1, \cdots, P_n$`)` | $P_i \in f$ and $1 \leq i \leq n$

6 *MFPlans* $\longleftarrow$ *MFPlans* $\cup$ MFP

7 `Final_Plan` $\longleftarrow$ `Combines(`$MFP_1, \cdots, MFP_k$`)` | $MFP_j \in$ ***MFPlans*** and $1 \leq j \leq k$

8 **return `Final_Plan`**

In order to construct the `Fragments`, the set of individual plans $P_\Gamma = \{p_{\gamma_1}, p_{\gamma_2}, \cdots, p_{\gamma_n}\}$ is divided into sets according to the ordering constraints in SC. Accordingly, each set is called `Fragment`. Furthermore, each `Fragment` is represented by a tuple $F = \langle P_\gamma, O_\gamma \rangle$, where P_γ is a set of individual plans ($P_\gamma \subseteq P_\Gamma$) and O_γ is a set of order constraints on P_γ. For example, suppose the ordering constraints in SC are $\{T_1 \prec T_2, T_2 \prec T_3, T_4 \prec T_5\}$. Suppose a set of individual plans $P_\Gamma = \{p_1, p_2, \cdots, p_7\}$ are respectively a solution plan for the abstract plan steps $T_1, T_2, \cdots, T_7$ as shown in figure 6.5(a). Then, according to the ordering constraints in the shared constraints set SC, these individual plans in P_Γ constitute three different fragments $F_1 = \langle \{p_1, p_2, p_3\}, \{p_1 \prec p_2, p_2 \prec p_3\} \rangle$, $F_2 = \langle \{p_4, p_5\}, \{p_4 \prec p_5\} \rangle$, and $F_0 = \langle \{p_6, p_7\}, \{\emptyset\} \rangle$ (See Figure 6.5(b),(c),(d)). In general, there are two types of fragment plans: **Related-Fragment** which 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.

Second, the set of individual plans in each fragment are merged in two phases: Detecting de-

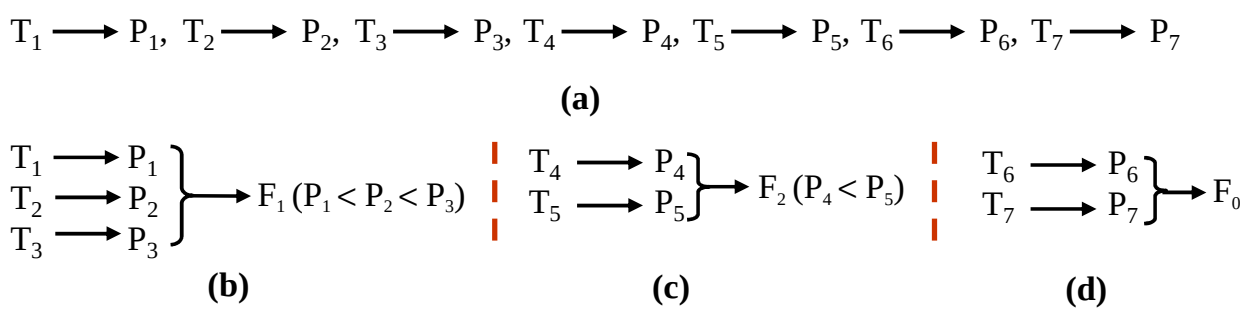


Figure 6.5 Fragment plans

pendency and detecting redundancy.

Phase 1: Detecting Dependency: Through this phase, the implicit dependency between individual plans in Zero-Fragment is determined. In our approach, the plan dependency is determined by matching pre-conditions and effects of the tasks in individual plans. This means that certain post-condition of plan p_{γ_i} tasks are required as pre-conditions for plan p_{γ_j} tasks.

There are two reasons for determining the dependency between individual plans: canceling the negative interactions *(i.e. one task deletes an effect or pre-condition of another task)*, and benefit from the positive interactions *(i.e. two different tasks need the same pre-condition and at least one of them does not remove it or one task generates pre-conditions of another task)*. Intuitively, if we have a set of independent plans that can be executed concurrently, they then are performed directly by integrating them into one large plan.

If the implicit dependency between individual plans in Zero-Fragment is detected, the master agent updates this fragment by adding new order constraints between these plans.

Despite the order dependency between plans, some tasks in these plans can be performed concurrently. Tasks can not take place concurrently if the pre- or post-conditions of the tasks in the successor plan are inconsistent with the post-conditions of the tasks in the predecessor plan [120]. Therefore, in order to determine the concurrent tasks, we will establish a comparison between pre- and post-conditions of tasks in the successor plan and the post-conditions of the tasks in the prede-

cessor plans. The comparison process will be started by checking pre- and post-conditions of the first task in the successor plan p_{γ_j} with post-conditions 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 *(i.e. deleted)*, this task will be executed sequentially in the plan p_{γ_i}, and the procedure in case-1 will be performed. Otherwise, the first task in p_{γ_j} will be executed concurrently to the predecessor plan p_{γ_i} and the procedure in case-2 will be performed.

Case-1:

1. Create a new order constraint <last task in p_i, current task in p_j >
2. Remove the order constraint <last task in p_i, goal() task in p_i >
3. Remove the goal() task from the predecessor plan p_i
4. Remove the order constraint <initial() task in p_j, current task in p_j >
5. Remove the initial() task from the successor plan p_j
6. Stop the comparison process

Case-2:

1. Remove the order constraint <initial() task in p_j, current task in p_j >
2. Remove the initial() task in the successor plan p_j
3. Create a new order constraint < initial() task in p_i, current task in p_j >
4. Continue the comparison process to the next task in the successor plan p_j.

Note that 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.

For example, as depicted in figure 6.6-(a) – suppose we have two individual plans p_i and p_j where plan p_i is the predecessor plan and plan p_j is the successor plan. Suppose the post-conditions of the first task te_{11} in the successor plan p_j is violated by another task in the predecessor plan p_i. Then

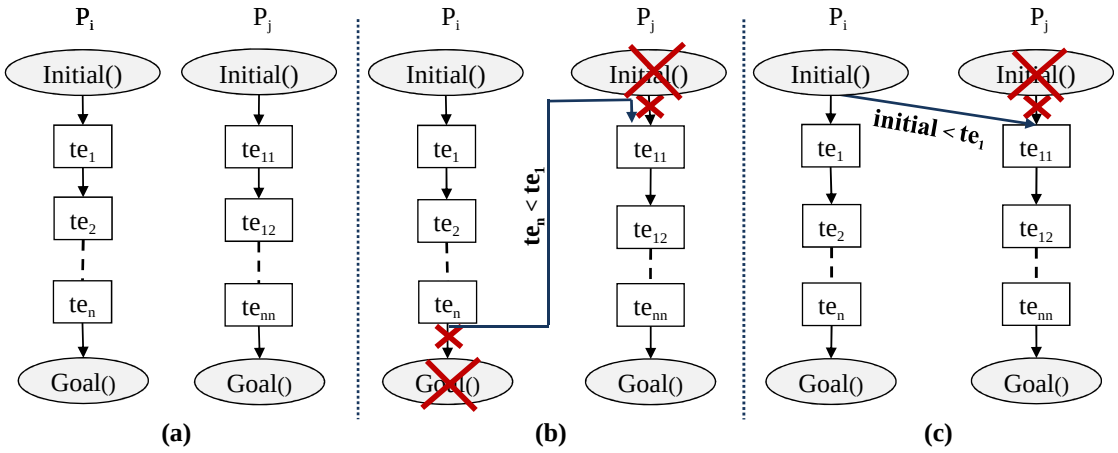


Figure 6.6 Dependency propagation

the steps in case-1 will be performed to generate figure 6.6-(b). Otherwise, the steps in case-2 will be performed to generate figure 6.6-(c).

Phase 2: Detecting Redundancy: In this phase, we detect the possible plan steps that can be merged and we update their constraints. A pair of plan steps in different individual plans is merged if their post-conditions are matched and there is no another task that is ordered after them that could violate these post-conditions. More formally,

Definition 21 (**Merging Plan steps** $merge(te_i, te_j)$)**.** $\forall$ *plan steps* $te_i \in P_{\gamma_i}$ *and* $te_j \in P_{\gamma_j}$, $merge(te_i, te_j)\ iff\ (post(te_i) = post(te_j)) \wedge (\neg\exists te_k \in P_{\gamma_j}\ violate\ post(te_j)\ s.t.\ (te_j \prec te_k))$

Once a pair of plan steps has been merged, the related constraints of the removed or joined plan step should be modified. This means, the replacement of the plan step will inherit all constraints of the merged plan steps as well as adding new constraints. As shown in figure 6.7, for all merged plan steps $te_j \in P_{\gamma_j}$ and $te_i \in P_{\gamma_i}$:

1. The plan step te_j is replaced by plan step te_i.

2. $\forall\ te_k, te_l, te_j \in P_{\gamma_j}$ and $\exists \langle te_k, te_j \rangle$, $\langle te_j, te_l \rangle \in \prec$ of plan P_{γ_j} add new order constraints $\langle te_k, te_l \rangle$ to plan P_{γ_j}.

3. Remove the ordering constraints $\langle te_k, te_j \rangle$, $\langle te_j, te_l \rangle$ $from\ plan\ P_{\gamma_j}$.

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.

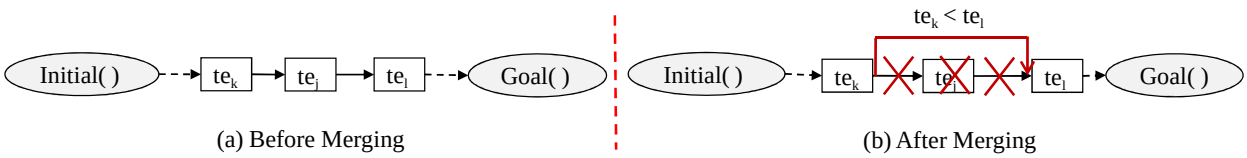


Figure 6.7 Local dependency propagation.

For all merged plan steps $te_j \in P_{\gamma_j}$ and $te_i \in P_{\gamma_i}$: $\forall\ te_k, te_l \in P_{\gamma_j}$

1. $\forall$ causal link $te_j \xrightarrow{\Phi} te_l \in CL$ of plan P_{γ_j} add new causal link $te_i \xrightarrow{\Phi} te_l$ to CL of plan P_{γ_j}.
2. Remove causal link $te_j \xrightarrow{\Phi} te_l\ from\ CL\ of\ plan\ P_{\gamma_j}$.
3. Remove causal link $te_k \xrightarrow{\Phi} te_j\ from\ CL\ of\ plan\ P_{\gamma_j}$.

6.4 Experimental and Empirical Analysis

In order to evaluate the introduced *hybrid multi-agent planning* approach, we consequently performed our experiments on qualitatively different problems by specifying various transportation means and goods. The number of transportation task in these planning problems ranged from one to six tasks. Each problem runs three times on the same configuration. We calculated the average values of three runs. Every run of the planning system was limited to a real time consumption of 18,000 seconds and an exploration of at most 10,000 plans. If a run of the planning problem failed to find a solution within these limits, it was counted as a non-terminating run and excluded from the average computation. A dash indicates that the run over a planning problem exceeded the limitation. *CPU-Time* is computed as the time of dividing the given planning problem plus the time

of pre-processing phase and the time which used to solve the largest cluster. While the *Merg-Time* denotes the running time of the merging process in seconds.

Consequently, in tables 6.1, 6.2 and 6.3 the column *PANDA* refers to the reference system's behavior, the column *HLM* refers to the PANDA version that performs a pre-processing phase and finally the column *HMAP* refers to the version that performs the integration between a MAP with pre-processing techniques. The column *HMAP* considers clustering the planning problem by two different methods `Dependent` and `Independent`. The experimental results which are done in tables 6.1, 6.2 and 6.3 use $f_{LCF}^{ModSel} + f_{DU}^{ModSel}$ as the modification selection strategy and $f_{FHZ}^{PlanSel} + f_{FMF}^{PlanSel}$ as the plan selection strategy.

6.4.1 Hybrid Multi-agent Planning Evaluation in the UM-translog Domain

For planning problems with one transportation task, the difference between planning systems with pre-processing *(HLM)* and hybrid multi-agent planning approach *(HMAP)* is not big. As we have documented in table 6.1, the average performance improvement over all one transportation task problems in the *UM-Translog* domain is about 51% compared to the PANDA planner. The HLM planner achieves an improvement of about 2% compared to HMAP.

Our experiments in the *UM-Translog* domain show poor performance (See Tables 6.2) of the *PANDA* and *HLM* versions, because it is difficult to solve planning problems which have a large number of abstract tasks in the initial plan.

The experiments show that the clusters which are obtained from our decomposition techniques either `Dependent` or `Independent` are easier to solve than the original planning problem. Solving clusters concurrently can save an important amount of time. Consequently, our hybrid multi-agent planning system is able to solve the problems for which the competing systems could not find a solution within the given resource bounds. For example, the average performance improvement of HMAP regarding all *UM-Translog* problems that have more than one abstract task in

Table 6.1 *UM-Translog* domain: The evaluation results of HMAP *(one transportation task)*.

Problem	PANDA	HLM	HMAP					
			`Dependent`			`Independent`		
	CPU-Time	CPU-Time	CPU-Time	Mer-Time	Total	CPU-Time	Mer-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	151	159	0	159	157	0	157
Translog-P4	772	621	630	0	630	625	0	625
Translog-P5	1074	159	160	0	160	155	0	155
Translog-P6	483	318	320	0	320	323	0	323
Translog-P7	216	129	130	0	130	135	0	135
Translog-P8	165	101	105	0	105	105	0	105
Translog-P9	348	245	246	0	246	243	0	243

the initial plan is about 72% in comparison to *PANDA*. HMAP achievies an improvement of about 48% comapred to *HLM* as documented in table 6.2. Not surprisingly, the performance improvements will increase dramatically if the number of abstract tasks in the initial plan is raised. Our experiments proved that when there is an interaction between tasks in the plan, the `Independent` decomposition technique is more efficient than `Dependent` decomposition technique such as in *UM-Translog* domain where the `Independent` technique achieves an average improvement of 21% over the `Dependent` technique as documented in table 6.2.

6.4.2 Hybrid Multi-agent Planning Evaluation in the Satellite Domain

We also performed our experiments on qualitatively different problems in the *satellite* domain by specifying various observations with different properties. The evaluated scenarios are therefore defined as observations on one to six satellites. Each problem runs with the same specification as we mentioned before in section 6.4.

Table 6.2 *UM-Translog* domain: The evaluation results of HMAP *(More than one transportation task)*.

Problem	PANDA	HLM	HMAP					
			Dependent			Independent		
	CPU-Time	CPU-Time	CPU-Time	Mer-Time	Total	CPU-Time	Mer-Time	Total
Translog-P10	170	102	103	0	103	105	0	105
Translog-P11	853	153	159	0	159	157	0	157
Translog-P12	621	229	231	0	231	235	0	235
Translog-P13	1184	639	358	179	537	512	0	512
Translog-P14	-	3437	476	956	1432	1794	964	2758
Translog-P15	-	-	1413	2397	3810	703	1967	2670
Translog-P15	-	-	4562	6094	10656	1587	6731	8318
Translog-P16	-	-	454	148	602	450	0	450
Translog-P17	1284	583	451	941	1392	627	878	1505
Translog-P18	-	3930	2954	2769	5723	750	2343	3093
Translog-P19	-	-	3335	5981	9316	1218	3622	4840
Translog-P20	-	-	4370	6327	10697	1463	7250	8713
Translog-P21	-	-	770	223	993	673	351	1024
Translog-P22	-	-	1705	1440	3145	2109	1345	3454
Translog-P23	-	-	547	418	965	4785	578	5363
Translog-P24	-	-	3366	5921	9287	1328	4364	5692
Translog-P25	3268	1287	1079	0	1079	698	392	1090
Translog-P26	-	4184	3417	1002	4419	489	835	1324
Translog-P27	-	-	3692	2015	5707	1123	1910	3033
Translog-P28	-	-	4007	3842	7849	1379	4808	6187
Translog-P29	-	-	4705	5841	10546	1777	6370	8147
Translog-P30	5238	1211	832	0	832	383	376	759
Translog-P31	-	10006	3227	1045	4272	537	833	1370
Translog-P32	-	-	3445	2614	6059	686	1939	2625
Translog-P33	-	-	3874	5637	9511	1040	5481	6521
Translog-P34	-	-	4739	6627	11366	1521	5904	7425
Translog-P35	-	2623	1047	0	1047	2045	753	2798
Translog-P36	-	-	6008	697	6705	5069	3471	8540
Translog-P37	-	-	3237	940	4177	540	1014	1554

Table 6.3 *Satellite* domain: This table shows the evaluation results of HMAP.

Problem	PANDA	HLM	HMAP					
			Dependent			Independent		
	CPU-Time	CPU-Time	CPU-Time	Merg-Time	Total	CPU-Time	Merg-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

Although the *satellite* domain does not benefit significantly from the landmark pre-processing technique, as there is hardly any landmark information available due to the shallow decomposition hierarchy of this domain, it achieves good performances with clustering techniques. As documented in table 6.3, solving the observation planning problems using the cluster technique `Independent` achieves an average improvement of about 29% over solving the same observation problems with the cluster technique `Dependent`.

Chapter 7

Conclusion

7.1 Research Contributions

The goal of this dissertation is developing methods to systematically reduce the search effort and to improve the efficiency of planning systems. We have developed a novel pre-processing technique to extract knowledge from a hierarchically structured planning domain and a current problem description which is used to significantly improve planning performance. Our pre-processing technique enables us to prune parts of the search space by identifying tasks that are not achievable from a certain initial situation. Furthermore, it is used to guide the hierarchical planning processes more efficiently towards a solution of a given planning problem. Finally, our approach supports the combination of a multi-agent based planning approach with the pre-processing technique in the context of hierarchical planning. In each chapter the technical details have been collected by an *Experimental and Empirical Analysis* in which we have tested our approach with established planning strategies from literature.

The following sections will therefore briefly discuss our approach. This chapter will conclude by suggesting future developments that may build on our work.

7.1.1 Extracting Hierarchical Landmarks

In recent years, the exploitation of knowledge gained by pre-processing a planning domain and/or problem description has proven to be an effective means to reduce planning effort. A lot of different pre-processing procedures have been proposed for classical state-based planning, where they serve to compute strong search heuristics. As opposed to this, pruning the search space of a hierarchical planner by pre-processing the underlying HTN-based domain description has not been considered so far. Therefore, We have presented an effective landmark technique for hierarchical planning. Hierarchical landmarks are those tasks that occur in the decomposition refinements on every plan development path. It analyzes the planning problem by pre-processing the underlying domain and prunes those regions of the search space where a solution cannot be found.

Our experiments on a number of representative hierarchical planning domains and problems give reliable evidence for the practical relevance of our approach. The performance gain went up to about 60% for problems with a deep hierarchy of tasks. Our technique is domain- and strategy-independent and can help any hierarchical planner to improve its performance.

7.1.2 Search Strategies

The problem which addressed in this contribution is how to systematically construct strategic guidance in order to improve the search efficiency of hierarchical planning systems in terms of explored search nodes. For a given planning problem description, an initial search node is successively expanded until a search node is created that describes a solution to the planning problem *(a solution plan)*. Planning systems explore only a fraction of the search space which is theoretically reachable. One always wants to keep this fraction as small as possible, since the runtime of a planning system correlates to the number of explored search nodes. In order to guide the search *(i.e., to decide which fraction to explore)*, planning systems use heuristics, so-called *search-strategies*. Therefore, we used the knowledge extracted from the pre-processing phase to introduce four search

strategies for hierarchical planning. Through these strategies, we compute the expansion effort of the problem – a heuristic to guide the selection of methods with which we can reduce the effective branching factor of the search space.

We have made a number of experiments to compete these search-strategies with a set of representative search procedures from literature. The results showed that the new strategies outperformed the established ones on all relevant problems.

7.1.3 Hybrid Multi-agent Planning

The knowledge which we obtained during the pre-processing process could be used in multiple ways. One would be to prune parts of the search space and to identify tasks that are not achievable from a certain situation. Another way would be to build a hybrid system that integrates the pre-processing technique with a multi-agent based planning approach. Our approach introduces the ability to break up the given planning problem into a set of clusters *(i.e. sub-problems)* using two different techniques: `Dependent` and `Independent`. Our hybrid approach guarantees that (1) the set of agents work independently in order to solve the set of clusters, (2) the individually constructed solution plans are merged in order to generate a global plan without additional refinement in any individual plan, and (3) the problems are solved in shorter time.

In hybrid multi-agent planning, we have performed a number of experiments on qualitatively different problems in the *UM-Translog* and *Satellite* domains. The number of tasks in each planning problem ranges between one and six tasks. Through out this experiment, our hybrid approach competed with non-pruning and pruning planners. The results give evidence for the practical relevance of our approach.

7.2 Future Work

The techniques discussed in this dissertation directly apply to hierarchical planning approaches. However, there are various extensions possible that apply to the hybrid planning approach:

- One of the main differences between those two approaches is that in hybrid planning, not only primitive tasks show preconditions and effects, but also abstract tasks. This allows to test even the abstract tasks for reachability. Another difference is that hybrid planning problems also specify a goal state that has to be accomplished. Using this goal state, one can use techniques from classical planning in order to generate classical landmarks which can be used in the hybrid setting.

- Some complex planning domains such as *crisis management* domain often cause intractable efforts in modeling the domain as well as a huge search space to be explored by the system. A way to overcome these problems is to impose a structure not only according to tasks but also according to relationships between properties of the objects involved, thereby using so-called *decomposition axioms*. One of our future work is modifying our landmark extraction algorithm to support decomposition axioms.

- Our empirical evaluation has also shown the success of the introduced landmark-aware search strategies which use the calculated landmarks in order to guide the search process. Future work will be devoted to the construction and evaluation of other types of landmark-aware strategies and to the investigation of those domain model and problem features that suggest their deployment.

- Plan coordination in our hybrid multi-agent planning depends on the utilized plan merging technique. However, the plan merging technique has two major drawbacks: (1) The success of the whole process is dependent on the ability of the merger process to join individual

solution plans into a general solution plan. This may be impossible because of the conflicts between individual plans that the merger process can not resolve. If the mering process fails, time spent planning will be wasted. (2) slave agents can not give each other assistance during the planning process because they plan individually. This means that for example an agent with exclusive control over a resource can not assist other agents in their planning of related tasks. Therefore, future work will focus on additional cooperation between slave agents during the planning process.

- In future work, the merging process can be modified by performing it inside the planner environment. To this end, all individual solution plans which are produced by slave agents are combined in a large plan. After that, the master agent submits the created plan to the planner environment as a new planning problem. Thus, the planner can solve conflicts in the new planning problem inside the planning environment by removing tasks or inserting new tasks.

Bibliography

[1] B. Bonet and H. Geffner, "Planning as Heuristic Search: New Results," In *Proceedings of the 5th European Conference on Planning (ECP-99)*, pp. 360–372 (Springer, 1999).

[2] P. Haslum, B. Bonet, and H. Geffner, "New Admissible Heuristics for Domain-Independent Planning," In *Proceedings of the International Conference On Artificial Intelligence*, pp. 1163–1168 (AAAI Press / MIT Press, 2005).

[3] J. Porteous, L. Sebastia, and J. Hoffmann, "On the Extraction, Ordering, and Usage of Landmarks in Planning," In *Proceedings of the 6th European Conference on Planning (ECP-01)*, A. Cesta and D. Borrajo, eds., pp. 37–48 (2001).

[4] A. Barrett, D. S. Weld, O. Etzioni, S. Hanks, J. Hendler, C. Knoblock, and R. Kambhampati, "Partial-Order Planning: Evaluating Possible Efficiency Gains," Journal of Artificial Intelligence **67,** 71–112 (1993).

[5] C. A. Knoblock, *Generating Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning* (Kluwer Academic Publishers, 1993).

[6] S. Minton, J. Bresina, and M. Drummond, "Total-Order and Partial-Order Planning: A Comparative Analysis," Journal of Artificial Intelligence Research **2,** 227–262 (1994).

[7] B. Schattenberg, A. Weigl, and S. Biundo, "Hybrid Planning Using Flexible Strategies," In *Advances in Artificial Intelligence, Proceedings of the 28th German Conference on Artificial Intelligence (KI-05)*, pp. 249–263 (Springer-Verlag Berlin Heidelberg, 2005).

[8] B. Schattenberg, J. Bidot, and S. Biundo, "On the Construction and Evaluation of Flexible Plan-Refinement Strategies," In *Advances in Artificial Intelligence, Proc. of the 30th German Conference on Artificial Intelligence (KI-07)*,

[9] R. Tsuneto, J. Hendler, and D. Nau, "Analyzing External Conditions to Improve the Efficiency of HTN Planning," In *Proceedings of the IN SIXTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pp. 913–920 (1998).

[10] R. Fikes and N. Nilsson., "STRIPS: A new approach to the application of theorem proving to problem solving.," Journal of Artificial Intelligence **2(3-4),** 189–208 (1971).

[11] B. Nebel., "On the compilability and expressive power of propositional planning formalisms.," Journal of Artificial Intelligence Research **12,** 271–315 (2000).

[12] M. Ghallab, C. Isi, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "PDDL, the Planning Domain Definition Language.," Technical Report No. TR-98-003/DCS TR-1165, University of Washington, Yale Center for Computational Vision and Control (1998) .

[13] A. Tate, "INTERPLAN: a plan generation system which can deal with interactions between goals.," Memorandum MIP-R-109, Machine Intelligence Research Unit, University of Edinburgh (1974).

[14] A. Tate, "Interacting goals and their use," In *Proceedings of the 4th international joint conference on Artificial intelligence (IJCAI-75)*, **1,** 215–218 (1975).

[15] D. Chapman, "Planning for conjunctive goals," Journal of Artificial Intelligence archive **32(3),** 333–377 (1987).

[16] D. Mcallester and D. Rosenblitt, "Systematic Nonlinear Planning," In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 634–639 (1991).

[17] S. Penberthy and D. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL," In *Proceedings of the Third International Conference on the principles of knowledge representation*, pp. 103–114 (Morgan Kaufmann, 1992).

[18] A. Tate, "Generating project networks," In *Proceedings of the 5th international joint conference on Artificial intelligence(IJCAI-77)*, 2 (1977).

[19] D. Corkill, "Hierarchical planning in a distributed environments.," In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence.*, p. 168 175 (1979).

[20] E. D. Sacerdoti, "The nonlinear nature of plans," In *Proceedings of the 4th international joint conference on Artificial intelligence (IJCAI-75)*, 1 (1975).

[21] D. Mcallester and D. Rosenblitt, "Systematic Nonlinear Planning," In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pp. 634–639 (1991).

[22] E. Pednault, "Generalizing Nonlinear Planning to Handle Complex Goals and Actions with Context-Dependent Effects," In *Proceedings of the 12th international joint conference on Artificial intelligence*, (1991).

[23] E. Pednault, "Formulating multi-agent dynamic-world problems in the classical planning framework," In *M.P. Georgeff and A.L. Lansky, Editors, Reasoning about Actions and Plans: Proceedings of the 1986 Workshop. San Mateo, CA (1987), pp. 4782 Also published ; In: J. Allen, J. Hendler and A. Tate, Editors, Readings in Planning, Morgan Kaufmann, San Mateo, CA (90)*, (1990).

[24] E. Pednault, "ADL: exploring the middle ground between STRIPS and the situation calculus," In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, (1989).

[25] H. L. S. Younes and R. G. Simmons, "VHPOP: Versatile heuristic partial order planner," JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH **20,** 405–430 (2003).

[26] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," Journal of Artificial Intelligence Research **20,** 61–124 (2004).

[27] A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis.," Journal of Artificial Intelligence. **90,** 1636–1642 (1995).

[28] E. D. Sacerdoti, "Planning in a hierarchy of abstraction spaces," In *Proceedings of the 3rd international joint conference on Artificial intelligence (IJCAI-73)*, (1973).

[29] C. Knoblock, "Automatically Generating Abstractions for Planning," Journal of Artificial Intelligence **68,** 243–302 (1994).

[30] K. Currie, A. Tate, and S. Bridge, "O-Plan: the Open Planning Architecture," Journal of Artificial Intelligence **52,** 49–86 (1991).

[31] A. Tate, B. Drabble, J. Dalton, and S. Bridge, "The Use of Condition Types to Restrict Search in an AI Planner," In *Proceedings of the Twelth National Conference on Artificial Intelligence*, pp. 1129–1134 (1994).

[32] D. Wilkins, "Practical planning: extending the classical AI planning paradigm," In *Morgan Kaufmann,San Francisco, CA, USA, ISBN:0-934613-94-X.*, (1988).

[33] D. Wilkins and K. Myers, "A Multiagent Planning Architecture," In *Proceedings of the Proceedings of AIPS-98*, pp. 154–162 (1998).

[34] D. Wilkins and R. V. Desimone, "Applying an AI Planner to Military Operations Planning," Journal of Intelligent Scheduling pp. 685–709 (1992).

[35] D. Wilkins, S. Smith, L. Kramer, T. Lee, and T. Rauenbusch, "Execution monitoring and re-planning with incremental and collaborative scheduling," In *International Conference on Automated Planning and Scheduling, Workshop on Multi-agent Planning and Scheduling, (ICAPS-05)*, (2005).

[36] K. Erol, "Hierarchical Task Network Planning:Formalization, Analysis, and Implementation," PhD thesis, Department of Computer Science, The University of Maryland, 1995 (1995).

[37] K. Erol, J. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning," In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pp. 249–254 (1994).

[38] D. Nau, Y. Cao, A. Lotem, and H. Muoz-Avila, "SHOP: Simple Hierarchical Ordered Planner," In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 968–975 (1999).

[39] D. Nau, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," Journal of Artificial Intelligence Research **20,** 379–404 (2003).

[40] S. Biundo and B. Schattenberg, "From Abstract Crisis to Concrete Relief (A Preliminary Report on Combining State Abstraction and HTN Planning)," In *Proceedings of the 6th European Conference on Planning (ECP 2001)*, pp. 157–168 (Springer Verlag, 2001).

[41] L. Castillo, J. Fdez-Olivares, and A. González, "On the Adequacy of Hierarchical Planning Characteristics for Real-World Problem Solving," In *Proceedings of the 6th European Conference on Planning (ECP-01)*, pp. 169–180 (2001).

[42] S. Andrews, B. Kettler, K. Erol, and J. A. Hendler, "UM Translog: A planning domain for the development and benchmarking of planning systems," Technical Report No. CS-TR-3487, University of Maryland (1995) .

[43] M. Young, M. Pollack, and J. Moore, "Decomposition and Causality in Partial-Order Planning," In *Proceedings of the Second International Conference on Artificial Intelligence and Planning Systems*, pp. 188–193 (1994).

[44] S. Kambhampati, A. Mali, and B. Srivastava, "Hybrid Planning for Partially Hierarchical Domains," In *Proceedings of the 15th International Conference on Artificial Intelligence*, pp. 882–888 (AAAI, 1998).

[45] S. Kambhampati, "Refinement Search as a Unifying Framework for analyzing Planning Algorithms," In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KRR-94)*, (1994).

[46] B. Schattenberg, "Hybrid Planning and Scheduling," PhD thesis, The University of Ulm, Institute of Artificial Intelligence (2009).

[47] A. E. Tara, A. C. Steve, and W. Xuemei, "An Argument for a Hybrid HTN/Operator-Based Approach to Planning," In *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, pp. 182–194 (1997).

[48] W. Stephan and S. Biundo, "Deduction-based Refinement Planning," In *Proceedings of the 3rd International Conference On Artificial Intelligence Planning Systems (AIPS-96)*, pp. 213–220 (AAAI Press, 1996).

[49] M. Fox and D. Long, "The Detection and Exploitation of Symmetry in Planning Problems," In *Proceedings of the Sixteenth international joint conference on Artificial intelligence (IJCAI-99)*, pp. 956–961 (1999).

[50] M. Fox and D. Long, "The automatic inference of state invariants in TIM," Journal of Artificial Intelligence Research **9,** 367–421 (1998).

[51] D. Long and M. Fox, "Efficient Implementation of the Plan Graph in STAN," Journal of Artificial Intelligence Research **10,** 87–115 (1999).

[52] B. Nebel, Y. Dimopoulos, and J. Koehler, "Ignoring Irrelevant Facts and Operators in Plan Generation," In *Proceedings of the 4th European Conference on Planning (ECP-97)*, pp. 338–350 (1997).

[53] P. Haslum and P. Jonsson, "Planning with Reduced Operator Sets," In *Proceedings of the AIPS*,

[54] J. Koehler, "Solving Complex Planning Tasks Through Extraction of Subproblems," In *Proceedings of the 4th AIPS*,

[55] J. Koehler and J. Hoffmann, "On Reasonable and Forced Goal Orderings and their Use in an Agenda-Driven Planning Algorithm," Journal of Artificial Intelligence Research **12,** 338–386 (2000).

[56] J. Hoffmann, J. Porteous, and L. Sebastia, "Ordered Landmarks in Planning," Journal of Artificial Intelligence Research **22,** 215–278 (2004).

[57] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," Journal of Artificial Intelligence Research **14,** 253–302 (2001).

[58] J. Porteous and S. Cresswell, "Extending Landmarks Analysis to Reason about Resources and Repetition," In , pp. 45–54 (2002).

[59] L. Zhu and R. Givan, "Landmark Extraction via Planning Graph Propagation," In , pp. 156–160 (2003).

[60] P. Gregory, S. Cresswell, D. Long, and J. Porteous, "On the extraction of disjunctive landmarks from planning problems via symmetry reduction," In , pp. 34–41 (2004).

[61] L. Sebastia, E. Onaindia, and E. Marzal, "Decomposition of Planning Problems," journal of AI Communications **19,** 49–81 (2006).

[62] S. Richter, M. Helmert, and M. Westphal, "Landmarks revisited," In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pp. 975–982 (AAAI Press, 2008).

[63] C. Bckstrm and B. Nebel, "Complexity Results for SAS+ Planning," Journal of Computational Intelligence **11,** 625–655 (1993).

[64] M. Helmert, "Concise finite-domain representations for PDDL planning tasks," Journal of Artificial Intelligence **173,** 503–535 (2009).

[65] M. Helmert, "The Fast Downward Planning System," Journal of Artificial Intelligence Research **26,** 191–246 (2006).

[66] S. Richter and M. Westphal., "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks.," Journal of Artificial Intelligence Research (JAIR) **39,** 127–177 (2010).

[67] E. Keyder and H. Geffner, "Heuristics for planning with action costs revisited," In *Proceedings of the 18th European Conference on Artificial Intelligence ECAI-08*, (2008).

[68] E. Karpas and C. Domshlak, "Cost-optimal planning with landmarks," (2009).

[69] V. Vidal and H. Geffner, "Branching and Pruning: An Optimal Temporal POCL Planner Based On Constraint Programming," Journal of Artificial Intelligence **170,** 298–335 (2006).

[70] M. Helmert and C. Domshlak, "Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?," (2009).

[71] M. Elkawkagy, B. Schattenberg, and S. Biundo, "Landmarks in Hierarchical Planning," In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, pp. 229–234 (2010).

[72] D. Nau, Y. Cao, A. Lotem, and H. Munoz-Avila, "SHOP: Simple Hierarchical Ordered Planner," In *Proceedings of the Sixteenth international joint conference on Artificial intelligence (IJCAI-99)*, pp. 968–975 (1999).

[73] K. Erol, J. Hendler, and D. S. Nau, "HTN Planning: Complexity and Expressivity," In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, pp. 1123–1128 (AAAI Press, 1994).

[74] K. Erol, J. Hendler, and D. Nau, "Complexity Results for HTN Planning," Journal of Annals of Mathematics and Artificial Intelligence pp. 69–93 (1995).

[75] K. Erol, J. Hendler, D. Nau, and R. Tsuneto, "A critical look at critics in HTN planning," In *Proceedings of the 14th international joint conference on Artificial intelligence (IJCAI-95)*, (1995).

[76] S. Ghosh, J. Hendler, S. Kambhampati, and B. Kettler, "UM Nonlin– A common Lisp Implementation of Nonlin: User Manual.," In *University of Maryland at College park, Department of Computer Science 1.2.2 edition, February-92*, (1992).

[77] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis," Journal of Artificial Intelligence Research **20,** 1–59 (2003).

[78] D. Joslin and M. E. Pollack, "Least-Cost Flaw Repair: A Plan Refinement Strategy for Partial-Order Planning,", 1994.

[79] M. E. Pollack, D. Joslin, and M. Paolucci, "Flaw Selection Strategies for Partial-Order Planning," Journal of Artificial Intelligence Research **6,** 223–262 (1997).

[80] L. Schubert and A. Gerevini, "Accelerating Partial Order Planners by Improving Plan and Goal Choices," In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, pp. 442–450 (IEEE Computer Society Press, 1995).

[81] A. Gerevini and L. Schubert, "Accelerating Partial-Order Planners: Some Techniques for Effective Search Control and Pruning," Journal of Artificial Intelligence Research **5,** 95–137 (1996).

[82] R. Tsuneto, "Efficient Refinement Strategies for HTN Planning," PhD thesis, Department of Computer Science, The University of Maryland, 1999 (1999).

[83] T. L. Mccluskey, D. Liu, and R. M. Simpson, "Using Knowledge Engineering and State Space Planning Techniques to Optimise an HTN Planner," In *Proceedings of the PlanSig-02*, pp. 1–11 (Delft University of Technology, The Netherlands, 2002. ISBN 1-3685-70-8, 2002).

[84] R. Tsumeto, J. Hendler, D. Nau, and L. Nunes de Barros, "Matching problem features with task selection for better performance in HTN planning," In , Proceedings of the AIPS-98 Workshop on Knowledge Engineering and Acquisition for Planning (1998).

[85] P. Laborie and M. Ghallab, "IxTeT: An Integrated Approach for Plan Generation and Scheduling," In *Proceedings of the 4th IEEE-INRIA Symposium on Emerging Technologies and Factory Automation (ETFA-95)*, pp. 485–495 (1995).

[86] E. Fink and M. Veloso, "Formalizing the PRODIGY Planning Algorithm," In *Proceedings of the European Workshop on Planning*, pp. 261–272 (1996).

[87] T. L. Mccluskey, "Object Transition Sequences: A New Form of Abstraction for HTN Planners," In *Proceedings of the fifth international conference on artificial intelligence planning systems (AIPS-2000)*, pp. 216–225 (2000).

[88] Q. Yang and A. Y. M. Chan, "Delaying Variable Binding Commitments in Planning," In *Proceedings In The 2nd International Conference on AI Planning Systems (AIPS)*, pp. 182–187 (1994).

[89] M. J. Stefik., "Planning with constraints.," Journal of Artificial Intelligence **16,** 111–140 (1981).

[90] M. Veloso and P. Stone, "FLECS: Planning with a flexible commitment strategy," Journal of Artificial Intelligence Research **3,** 25–52 (1995).

[91] M. A. Peot and D. E. Smith, "Threat-removal strategies for partial-order planning," In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pp. 492–499 (AAAI, 1993).

[92] M. A. Peot and D. E. Smith, "Delaying Variable Binding Commitments in Planning," In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pp. 492–499 (1993).

[93] S. Minton, J. Bresina, and M. Drummond, "Commitment Strategies in Planning: A Comparative Analysis," In *Proceedings of the 12th international joint conference on Artificial intelligence IJCAI-91*, pp. 259–265 (1991).

[94] R. Tsuneto, K. Erol, J. A. Hendler, and D. S. Nau, "Commitment Strategies in Hierarchical Task Network Planning," In *AAAI/IAAI, Vol. 1*, pp. 536–542 (1996).

[95] R. Tsuneto, D. Nau, and J. Hendler, "Plan-Refinement Strategies and Search-Space Size," In *Proceedings of the European conference on planning*, pp. 414–426 (1997).

[96] B. Bonet and M. Helmert, "Strengthening Landmark Heuristics via Hitting Sets," In *Proc. of ECAI 2010*, pp. 329–334 (2010).

[97] M. Elkawkagy and S. Biundo, "Hybrid Multi-agent Planning," In *Proceedings of the Ninth German Conference on Multi-Agent System Technologies (MATES-11)*, (2011).

[98] M. Iwen and A. D. Mali, "Automatic Problem Decomposition for distributed planning.," In *Proceedings of the International Conference on Artificial Intelligence (ICAI).*, p. 411 417 (2002).

[99] M. Wellman, W. Walsh, P. Wurman, and J. MacKie-Mason, "Auction protocols for Decentralized scheduling.," Journal of Games and Economic Behavior **35,** 271 303 (1998).

[100] Q. Yang, D. S. Nau, and J. Hendler, "Merging Separately generated plans with restricted interactions.," Journal of Computational Intelligence **8(4),** 648 676 (1992).

[101] E. Durfee, "Organizations, Plans and schedules: An interdiscplinary perspective on coordinating AI agents.," Journal of Intelligent Systems. Special Issue on the Social Context of Intelligent Systems 3 (1993).

[102] J. Rosenschein and G. Zlotkin, "Consenting Agents: Designing Conventions for Automated Negotiation," Journal of Artificial Intelligence Magazine **15,** 29–46 (1994).

[103] G. Zlotkin and J. Rosenschein, "Mechanism Design for Automated Negotiation, and its Application to Task Oriented Domains," Journal of Artificial Intelligence **86,** 195–244 (1996).

[104] L. Beaudoin, "Goal processing in autonomous agents," (1994).

[105] K. Biggers and T. Ioerger, "Automatic Generation of Communication and Teamwork within Multi-Agent Teams," Journal of Applied Artificial Intelligence **15,** 875–916 (2001).

[106] J. Rosenschein, "Synchronization of Multi-Agent Plans," In *AAAI*, pp. 115–119 (1982).

[107] M. Georgeff, "Communication and Interaction in Multi-agent Planning.," In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI-83).*, pp. 125–129 (1983).

[108] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of cooperation in distributed problem solving," In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.*, pp. 767–770 (1983).

[109] E. Ephrati and J. Rosenschein, "Multi-ahent planning as the process of merging distributed sub-plans.," In *Proceedings of the Twelfth International Workshop on distributed Artificial Intelligence (DAI-93)*, p. 115 129 (1993).

[110] Q. Yang, "Intelligent Planning: A Decomposition and Abstraction Based Approach.," New York: Springer (1997).

[111] I. Tsamardinos, M. Pollack, and J. Horty, "Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches.," In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling.*, pp. 264–272 (2000).

[112] I. Meiri, "Temporal Reasoning: A Constraint-Based Approach," (1992).

[113] M. Iwen and A. D. Mali, "Distributed Graphplan.," In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI).*,

[114] A. ter Mors, J. Valk, and C. Witteveen., "Task Coordination and Decomposition in Multi-Actor Planning Systems.," In *Proceedings of the Workshop on Software-Agents in Information Systems and Industrial Applications (SAISIA).*, pp. 83 – 94 (2006).

[115] B. Clement and E. Durfee, "Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information.," In *Proceedings of the Sixteenth National Conference on Artificial Intelligence.*, pp. 495 – 502 (1999).

[116] B. Clement and E. Durfee, "Scheduling High-Level Tasks among Cooperative Agents," In *Proceedings of the Third International Conference on Multiagent Systems, ICMAS 1998, 3-7 July 1998, Paris, France*, pp. 96–103 (1998).

[117] M. desJardins and M. Wolverton, "Coordinating a distributed planning system.," Journal of Artificial Intelligence. **20(4),** 45 53 (1999).

[118] J. Cox and E. Durfee, "Discovering and Exploiting Synergy between Hierarchical Planning Agents.," In *Proceedings of the second international conference on autonomous agents and multi-agent systems (AAMAS).*, pp. 281–288 (ACM Press, 2003).

[119] B. Clement and E. Durfee, "Top-down Search for Coordinating the Hierarchical plans of Multiple Agents.," In *Proceedings of the third international conference on autonomous agents.*, pp. 252 – 259 (ACM Press, 1999).

[120] J. Cox and E. Durfee, "An Efficient algorithm for multi-agent plan coordination.," In *Proceedings of the fifth international conference on autonomous agents and multi-agent systems (AAMAS-05).*, pp. 828–835 (2005).

[121] H. Hayashi, "Stratified Multi-agent HTN planning in Dynamic Environments.," In *Proceedings of the First KES International Symposium, Agent and Multi-Agent Systems: Technologies and Applications(KES-AMSTA).*, pp. 189–198 (2007).

[122] M. de weerdt and R. van der Krogt, "A method to Integrate Planning and Coordination.," In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS-02).*, pp. 83–88 (AAAI Press, 2002).

[123] R. V. der Krogt, , and M. D. Weerdt, "Coordination Through Plan Repair.," In *Proceedings of the MICAI-05: Advances in Artificial Intelligence.*, pp. 264–274 (2005).

[124] Y. Dimopoulos and P. Moraitis, "Multi-Agent Coordination and Cooperation through Classical Planning.," In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (AIT-06)*, pp. 398–402 (2006).

[125] M. Elkawkagy, P. Bercher, B. Schattenberg, and S. Biundo, "Exploiting Landmarks for Hybrid Planning.," In *Proceedings of the 25th PuK Workshop Planen, Scheduling und Konfigurieren, Entwerfen*, (2010).

[126] M. Elkawkagy, P. Bercher, B. Schattenberg, and S. Biundo, "Landmark-Aware Strategies for Hierarchical Planning.," In *Proceedings of the Workshop on Heuristics for Domain-independent Planning (HDIP-2011) at ICAPS*, (2011).

[127] S. Kambhampati, L. Ihrig, and B. Srivastava, "A Candidate Set based analysis of Subgoal Interactions in conjunctive goal planning," In *Proceedings of the 3rd International Conference on AI Planning Systems (AIPS-96)*, pp. 125–133 (1996).

[128] R. Korf, "Planning as search: a quantitative approach.," Journal of Artificial Intelligence **33,** 6588 (1987).