ABSTRACT

Title of dissertation: HIERARCHICAL GOAL NETWORKS: FORMALISMS AND ALGORITHMS FOR PLANNING AND ACTING

Vikas Shivashankar, Doctor of Philosophy, 2015

Dissertation directed by: Professor Dana S Nau
Department of Computer Science

In real-world applications of AI and automation such as in robotics, computer game playing and web-services, agents need to make decisions in unstructured environments that are open-world, dynamic and partially observable. In the AI and Robotics research communities in particular, there is much interest in equipping robots to operate with minimal human intervention in diverse scenarios such as in manufacturing plants, homes, hospitals, etc. Enabling agents to operate in these environments requires advanced planning and acting capabilities, some of which are not well supported by the current state of the art automated planning formalisms and algorithms. To address this problem, in my thesis I propose a new planning formalism that addresses some of the inadequacies in current planning frameworks, and a suite of planning and acting algorithms that operate under this planning framework.

The main contributions of this thesis are:

- **Hierarchical Goal Network (HGN) Planning Formalism**. This planning formalism combines aspects (and therefore harnesses advantages) of Classical Planning

and Hierarchical Task Network (HTN) Planning, two of the most prominent planning formalisms currently in use. In particular, HGN planning algorithms, while retaining the efficiency and scalability advantages of HTNs, also allows incorporation of heuristics and other reasoning techniques from Classical Planning.

- **Planning Algorithms**. **G**oal **D**ecomposition **P**lanner (GDP) and the **Go**al **De**composition with **L**andmarks (GoDeL) planner are two HGN planning algorithms that combines hierarchical decomposition with classical planning heuristics to outperform state-of-the-art HTN planners like SHOP and SHOP2.

- **Integration with Robotics**. The **C**ombined **H**GN **a**nd **M**otion **P**lanning (CHaMP) algorithm integrates GoDeLwith low-level motion and manipulation planning algorithms in Robotics to generate plans directly executable by robots.

Given the need for autonomous agents to operate in open, dynamic and unstructured environments and the obvious need for high-level deliberation capabilities to enable intelligent behavior, the planning-and-acting systems that are developed as part of this thesis may provide unique insights into ways to realize these systems in the real world.

HIERARCHICAL GOAL NETWORKS: FORMALISMS AND ALGORITHMS FOR PLANNING AND ACTING

by

Vikas Shivashankar

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Dedication

To my parents and my brother Vikram, without whose love and support this thesis would not have been possible.

Acknowledgments

First and foremost, I would like to thank Dana Nau for being a wonderful advisor during my PhD. He created a fantastic environment for me in which I learnt how to do research the right way. He helped me learn from both my successes as well as from my mistakes, and set the right example in terms of how to do good science. I am also extremely grateful for his hands-off approach towards advising, which I strongly believe brought out the best in me. He gave me plenty of time to mull over research questions, giving me a ton of space in which I could thrive. At the same time, he was very approachable and was always available for technical discussions. In short, it was clear that he set a very high bar for himself, both from a research as well as advising standpoint, and at the same time was very understanding and patient with his students and gave them the time to flourish. I am extremely fortunate to have had him as my advisor, and will forever cherish my time here at UMd spent under his tutelage.

I would also like to thank Ugur Kuter, who also advised and guided me during my thesis. Like Dana, he too provided a guiding hand throughout my PhD and his advice during the course of my thesis research was invaluable. His guidance and technical expertise was instrumental in making this thesis happen.

I would also like to acknowledge the help and guidance provided by S K Gupta and Krishnanand Kaipa. I am extremely grateful for their guidance and support in our work on applying planning techniques to automated manufacturing problems. I would especially like to thank Krishna for his time and patience to be able to regularly meet with me for technical discussions despite his busy schedule.

I would like to thank Michael Cox and Don Perlis for a number of useful discussions on goal reasoning. These discussions helped us find some interesting research challenges that my thesis work could be applied to, and for that I'm very grateful. I would also like to thank my prelim and thesis committees for useful feedback that helped shape my thesis.

I would also like to acknowledge the contribution of the support staff here at UMd; without their help, my time here would not be nearly as enjoyable as it was.

I would also like to acknowledge research funding, in particular ONR grants N000141310597 and N000141210430, which in part supported the work in this thesis.

I was extremely fortunate to have a very supportive group of friends here at UMd; their presence and emotional support helped me tremendously during my PhD. I would especially like to thank Aishwarya, AJ and Chirranjeevi for making my time here much, much more enjoyable and memorable than it otherwise might have been.

Last but not least, I would like to thank my parents and my brother Vikram for always being there for me, showering me with love, affection and emotional support that has helped me through the darkest of times during the course of my PhD. No matter what the adversity, I could always rely on them to lift my spirits up and keep me going. They were a constant amongst all the craziness that comes with moving to a new country, new culture, and a completely new phase of life. Without their love and support, this thesis would not exist and for that reason, I dedicate this work to them.

Vikas Shivashankar,

April 2015

Table of Contents

List of Figures

List of Abbreviations

Chapter 1: Introduction

In the real world, intelligent agents typically have to operate in environments that are open-world, partially observable and dynamic, such as robotics [1,2], computer game playing [3,4], web-services [5] and others. In the area of robotics in particular, research has slowly begun to shift focus from low-level issues of getting robots to execute *preprogrammed* plans reliably in highly constrained "lab-like" settings to the bigger challenge of operating autonomously in unknown environments and completing the given task with minimal human intervention. Enabling agents to operate as described requires AI research to support advanced task-level planning and acting capabilities.

However, the current state of the art AI planning formalisms and algorithms, the most prominent of which are *Classical Planning* and ***H****ierarchical* ***T****ask* ***N****etwork* (HTN) planning, do not adequately support some of these requirements. The following paragraphs (and Table 1.1) compares these two formalisms.

Classical planning algorithms take as input only the *descriptive models* of actions that can be executed in the domain. They leverage sophisticated domain-independent heuristics and other reasoning techniques to decide how to choose and organize these actions suitably in order to achieve the given goal. However, due to their inability to model and exploit any additional domain-specific strategies, they often do not scale well

Criterion	**Classical Planning**	**HTN Planning**
Structure	flat	hierarchical
Amount of Domain Engineering	low (only Action Models needed)	high (Action Models, HTN Methods needed)
Expressivity	low (finite-state machine)	high (Turing-complete)
Heuristic guidance	Yes; sophisticated heuristic search guidance available	No; relies almost exclusively on HTN methods to guide search
Scalability	low	high, when given high-quality HTN methods

Table 1.1: Comparison of Classical and HTN Planning

in real-world planning domains.

HTN planning algorithms take, in addition to the classical planning action models, additional domain-specific advice as input; this knowledge consists of HTN *methods*, which are descriptions of how to decompose complex problems into simpler subproblems. Due to this, HTN planners are much more scalable than classical planners in practice. Moreover, the hierarchical nature of HTNs mirrors the hierarchical structure of many real-world planning applications, making HTNs a natural choice in practice.

However, HTN planning has its own inadequacies, some of which are:

- **Huge Domain Engineering Effort in writing HTN methods**. This is because the HTN formalism requires users to provide methods to cover for every possible scenario that the agent could encounter. If the HTN planner finds itself in a situation the user hadn't anticipated, it would just fail without returning a solution.
- **Brittleness in Open, Dynamic Environments**. The previous problem is exacerbated in open, dynamic environments; since events outside of the agent's control can happen leading to novel situations not anticipated by the user, HTN planners are not well suited to work in open, dynamic environments.
- **Difficulty in designing domain-independent HTN planning heuristics**. Heuristics are critical in guiding the algorithm quickly towards high-quality solutions; due to the lack of such heuristics, HTN planners are often completely reliant on the user-provided knowledge in providing the necessary guidance, thus further increasing the burden on the user.

There have been attempts to alleviate these issues by combining HTN and Classical planning approaches, the idea being that planners could leverage HTNs whenever available, and fall back to Classical Planning otherwise. This would help increase robustness since the completeness of HTN planners no longer depends on the coverage of the given HTN methods. Similarly, one could imagine constructing HTN planning heuristics by adapting Classical Planning heuristics to work in the HTN setting. However, due to incompatibilities between the two formalisms, these efforts have met only with limited success.

The aim of this thesis is therefore to design a new hierarchical task-level planning formalism that better combines elements of HTN and Classical Planning, and exploit this synergy to:

- **design better task planning algorithms** that can leverage classical planning heuristics and other techniques both to reduce the domain-engineering burden and to generate higher-quality plans;
- **extend their scope to Robotics** by developing algorithms that can refine the generated task-level plans into a more detailed plan that can directly executed by the robots.

1.1 Contributions of This Thesis

The contributions of this thesis include a hierarchical planning formalism that combines advantages of Classical and HTN Planning formalisms into a single theory, as well as a suite of algorithms developed under this formalism that (1) generate plans for the given planning problem, and (2) integrate these algorithms with off-the-shelf *motion planners* from Robotics to generate plans that are directly executable by robotic platforms.

Our formalism allows these algorithms to incorporate classical planning heuristics and other reasoning techniques into a hierarchical planning setting, enabling them to outperform current state of the art planning systems.

1. **Planning Formalism**. As indicated above, classical planning and HTN planning have different (and in fact, complementary) advantages and disadvantages. To combine the advantages of both approaches, we have developed **H**ierarchical **G**oal **N**etwork (HGN) planning [6], a hybrid planning formalism that is hierarchical, but is based on classical goals rather than tasks. HGN methods, the counterpart of HTN methods in this formalism, thus decompose goals into networks of subgoals. This has several advantages over HTNs:

    - Unlike in HTNs where one has to commit to a task name in the methods, in HGN methods we simply specify in a declarative manner what should be achieved, instead of how to achieve it. The planner can then decide which methods and actions are relevant and applicable.

    - By moving from task hierarchies to goal hierarchies, leveraging reasoning techniques from Classical Planning such as state-space heuristics, propositional landmark inference, etc. now becomes feasible. Analogous efforts in HTNs have not not been very successful. This is mainly due to incompatibilities between Classical and HTN Planning; HTN planning is centered around tasks that need to be accomplished while Classical Planning is centered around goals that need to be achieved.

    - When given only partial control knowledge, goal decomposition using HGN methods can be easily augmented with Classical Planning techniques to fill in the gaps. Analogous approaches in HTN planning have been difficult to realize for the same reason as in the previous point; it has been hard to combine aspects of HTN and Classical Planning.

2. **Planning Algorithms**. We propose two HGN planning algorithms that help realize the advantages of the HGN planning formalism.

    - The **G**oal **D**ecomposition **P**lanner (GDP) [6] is a sound and complete HGN planning algorithm. It is similar in several ways to SHOP/SHOP2 (two popular HTN planning algorithms) and like them, requires a complete set of methods in order to function correctly. However, one of the novel aspects of GDP is its heuristic function; it is enhanced with a variant of the *relaxed planning graph* heuristic [7]. Typically HTN domain models manually specify the order in which the methods and operators should be tried when more than one of them is applicable; the heuristic function in GDP makes this easier by enabling the planner to deduce the best order on its own.

      Experiments comparing GDP to SHOP2 showed that GDP performed comparably to SHOP2 when given strong control knowledge, and outperformed SHOP2 when given weak control knowledge due to its heuristic function. A comparison of HGN and HTN domain model sizes showed that the HGN formalism allowed for much simpler and succinct encodings, mainly because the declarative semantics eliminated the need for a plethora of auxiliary methods and bookkeeping operations which SHOP2 domain models required.

- The **Go**al **De**composition using **L**andmarks (GoDeL) algorithm [8] extends GDP using subgoal inference via landmarks and action chaining techniques to plan with partial sets of HGN methods, guaranteeing soundness and completeness irrespective of the completeness of the given set of methods. Experiments showed that GoDeL could generate solutions with any amount of knowledge given to it, its performance improved as more knowledge was given, and that it performed at par with GDP when given complete knowledge.

3. **Combined Task and Motion Planning Algorithms for Robotics**. HGN planning algorithms, like most other task-level planning algorithms, operate on symbolic abstractions of the target applications, and therefore generate abstract plans that are not directly executable in the target domain. Instead, low-level domain-specific planning techniques are needed to translate the abstract plans into a concrete plan that is executable.

   This thesis proposes the **C**ombined **H**GN **a**nd **M**otion **P**lanning (CHaMP) algorithm, which attempts to address this problem in the context of planning for autonomous robots. CHaMP integrates HGN planning with motion and manipulation planning techniques from Robotics that handle the low-level planning problems of navigation, object grasping, etc. It tightly integrates the two levels of planning by leveraging certain useful properties of a class of motion planners called *heuristic search motion planners* that allow estimates of how well the search process is going on at the lower levels to be sent up to the HGN planner, which can then evaluate the relative merits of continuing that search process versus trying another option at the upper levels.

The rest of this document is structured as follows. Chapter 2 describes the related literature in more detail and put our work in perspective. Chapter 3 then formalizes HGNs. Chapter 4 describes GDP, the first HGN planning algorithm we developed. Chapter 5 describes GoDeL, our second HGN planning algorithm that extends GDP to work with incomplete sets of HGN methods. Chapter 6 then describes CHaMP, the integrated planning algorithms for robotics applications. Chapter 7 discusses possible extensions and open problems within the scope of this thesis. Finally, Chapter 8 wraps up.

Chapter 2: Related Work

In this section, we shall review the literature related on prominent task-level planning formalisms like Classical and HTN Planning. The literature related to integrating task and motion planning techniques in Robotics will be reviewed in Chapter 6.

2.1 Classical Planning

As mentioned in Chapter 1, classical planning algorithms take as input a domain description D consisting of models of all the actions executable in the domain. Each action model consists of *preconditions* that need to be true in the system state in order to execute the action, as well as *effects* that the action generates after execution. The action *preconditions* and *effects* are both modeled as formulas in first-order logic. Example 1 describes such a model for the `load-truck` action in the Logistics domain.

Example 1. *A model of the* `load-truck` *action, which loads object* `?o` *onto truck* `?t` *at location* `?l`*, in the Logistics domain will look as follows:*

```
(:action load-truck
:parameters   (?o ?t ?l)
:precondition (and (obj-at ?o ?l)
                   (truck-at ?t ?l))
```

```
    :effect        (and (not (obj-at ?o ?l))
                        (in-truck ?o ?t)))
```

The preconditions of this action require both object `?o` *and truck* `?t` *to to be at location* `?l`*. After the application of this action, the effects of this action model the fact that* `?o` *is now in the truck, and no longer at* `?l`*.*

A problem in classical planning is modeled as a $(D, \mathsf{initial\text{-}state}, \mathsf{goal})$ tuple. The initial-state and goal, along with the domain D, induces a search space; nodes are states and edges denote actions that can take the system from one state to the next. The problem can now be formulated as searching for a path in this search space from initial-state to a state in which goal is satisfied.

The main thrust in AI Planning research has been in the development of domain-independent heuristics and other reasoning techniques that can guide the underlying search algorithm (which is usually depth-first search or A*) towards a good solution. The Fast-Forward planner [7] was the first really successful example of such an approach; the authors developed the *relaxed planning graph* (RPG) heuristic which efficiently solves a relaxed version of the original problem and uses that solution to guide the search. Since then, there have been a number of planners [9–12] that have proposed various heuristic techniques that help scale up classical planning.

In particular, the LAMA planner [12], which won the 2008 and 2011 International Planning Competitions, repurposes the RPG technique to infer *propositional landmarks*: facts that need to be made true at some point in every plan that solves the given problem.

It then uses a novel landmark-based heuristic to guide the planner towards a goal state via the inferred landmarks.

There has been a number of research attempts in extending these heuristics and other techniques to do optimal planning [13, 14], temporal reasoning [15, 16], reason about metric effects [17, 18] among others. However, despite the development of highly sophisticated reasoning techniques in these works, the impact of classical planning in practical applications has been limited by the impoverished planning model, which does not allow for specification of higher-level planning strategies, even if available. Next, we shall review HTN planning, which does provide for specification of richer planning knowledge.

2.2 Hierarchical Task Network (HTN) Planning

HTN Planning is a hierarchical planning framework that allows encoding, in addition to the base actions, higher-level planning advice on how to decompose complex tasks into sets of simpler subtasks. This allows the planners to decompose the given planning problem recursively into smaller and smaller subtasks using the given planning knowledge until they arrive at an executable sequence of *primitive tasks* (which correspond directly to the base actions).

Example 2. *An example HTN method that moves a package between locations in the same city using trucks.*

```
(:method          (in-city-delivery ?truck ?obj
                                      ?loc-from ?loc-to)
:preconditions   ((in-city ?loc-from ?city)
                  (in-city ?loc-to ?city)
                  (truck ?truck ?city))
:subtasks        ((:task truck-at ?truck ?loc-from)
                  (:task !load-truck ?obj ?truck
                                        ?loc-from)
                  (:task truck-at ?truck ?loc-to)
                  (:task !unload-truck ?obj ?truck
                                          ?loc-to)))
```

In this example, `?obj` *needs to be moved from* `?loc-from` *to* `?loc-to` *using* `?truck`*. The preconditions that need to be true in the current state for this method to be applicable are* `?loc-from` *and* `?loc-to` *should be in the same city and* `?truck` *should also belong to that city. If the preconditions are satisfied, the method decomposes the top-task into the subtasks of (1) getting* `?truck` *to* `?loc-from`*, (2) loading* `?truck` *with* `?obj`*, (3) getting* `?truck` *to* `?loc-to`*, and finally (4) unloading* `?obj` *and depositing it at* `?loc-to`*.*

As one can imagine, this ability to exploit extra domain-specific knowledge allows HTN planners to scale much better than Classical planners, thus enabling more widespread use in practical planning applications. In fact, even before HTN planning was properly formalized and analyzed, there existed several HTN-like planning systems used in practice [19–21].

Erol et al [22] provided the first formalization and theoretical analysis of HTN planning and later implemented the Universal Method-Composition Planner (UMCP) [23], a sound and complete HTN planning procedure with respect to their formalization. Nau et al then introduced SHOP and SHOP2 [24, 25], two HTN planners that were vastly more efficient than UMCP and are considered state of the art even today.

However, while HTNs allow encoding of sophisticated domain-specific knowledge, HTN planners require *complete* models of such knowledge; informally, what this means is that users ought to provide HTN methods for every eventuality that the planner faces. This is intractable even in moderately complex domains; there have been attempts to combine HTN planning with Classical Planning in order to provide some fallback planning mechanism in case of incomplete HTN models [26–28], but the incompatibilities between HTNs (which is based on tasks) and Classical (based on goals) planning necessitated *ad-hoc* modifications and restrictions, leading to unsatisfactory solutions.

Additionally, there is a lack of an 'optimization' element in HTNs; for instance, there is a conspicuous lack of optimal and anytime HTN planning algorithms that use admissible heuristic estimates to search for a cost-efficient plan; the closest that comes to this is a version of SHOP2 that uses depth-first branch and bound techniques to iteratively prune search for the next solution based on the best solution found so far [25]. However, this can potentially require searching an exponential amount more than usual, thus negating all the efficiency advantages of HTN planning. An alternative approach would be to use admissible heuristic estimates to guide the search. However, this has not been possible in HTNs due to the lack of HTN planning heuristics, which is also due to the incompatibilities between tasks and goals; heuristics from classical planning cannot be easily adapted to work in the HTN setting. Therefore, end-users of HTN planning systems often need to encode very complex domain models to direct the planner in exactly the right directions that will lead to good solutions.

Chapter 3: Hierarchical Goal Networks

As Chapter 1 indicates, domain-independent planning and HTN planning formalisms have different (and in fact, complementary) advantages and disadvantages. Therefore, we attempted to come up with a hybrid formalism that is hierarchical, but is based on goals rather than tasks. This idea is not new; SIPE [21] and PRS [29, 30] both use goal decomposition techniques in order to build planning systems. Instead, our contributions lie in (1) building a formal theory and analyzing properties of a hierarchical goal-based planning framework, and (2) exploiting connections between HGNs and techniques in domain-independent planning such as state-space heuristics and landmark reasoning.

Differences between tasks and goals. Before we get to the HGN formalism, a note on some fundamental differences between tasks and goals might benefit the reader. Ever since Sacerdoti's seminal work on NOAH [31], problem decomposition have been recognized as a useful planning technique. However, an important design decision made in these early works was to use *task symbols* as the *de facto* problem representation; this was motivated by practical considerations in applications like mission planning, where its natural to view the decision-making process as a set of tasks (or missions) that need to be accomplished. So, practical HTN planning systems often used task symbols to represent problems, and used domain-specific decomposition knowledge to decompose these tasks into smaller and smaller tasks until it reached tasks that had an associated operational model that could then be executed.

However, in formalizations of HTN planning [22, 23, 32], tasks got decomposed into *primitive* tasks that had an associated descriptive model, much like actions in Classical Planning. This was there to be able to easily check whether a plan is executable. Therefore, tasks were treated as abstract symbols with no intrinsic meaning; they derived their semantics from the methods that decomposed them.

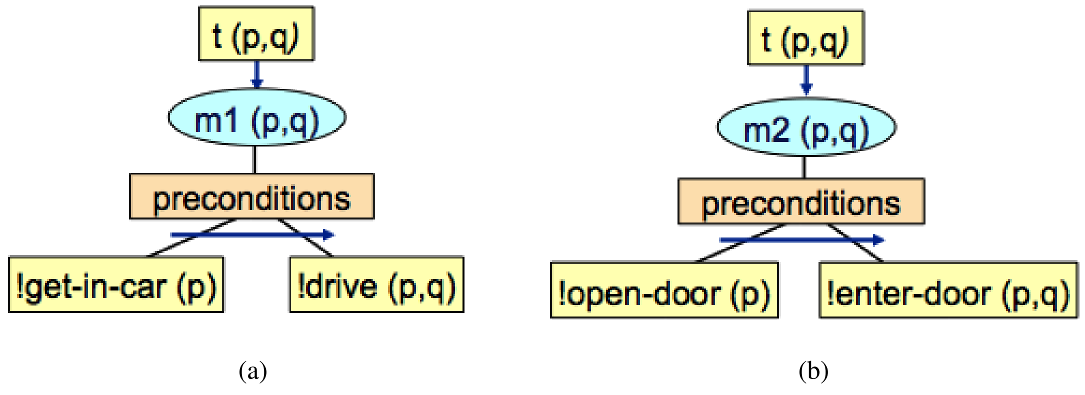


Figure 3.1: Decompositions of a task symbol `t`. Figure (a) shows the result of decomposing `t` using method $m1$, while figure (b) shows the result if method $m2$ was used. The resulting plans are formed by the leaves of the tree.

For instance, in Figure 3.1 we see the result of decomposing the task symbol `t` by two methods $m1$ and $m2$. Thus, the same task symbol `t` can yield completely different plans based on the method that decomposes it; this is because its simply a syntactic construct and its semantics is determined by the methods that decompose it.

This aspect of HTNs makes it hard to incorporate sophisticated reasoning techniques into HTNs. For instance, this is the reason why its hard to incorporate planning heuristics into HTNs; given a set of tasks (that might have been obtained by decomposing the top-level task) that need to be planned for, its hard to estimate how costly it is to accomplish these tasks since we don't know what the task symbols mean. Similarly, if we want to repair HTN plans during execution, its hard to localize the faulty section of the plan since we don't know the semantics of the tasks in the plan. For this reason, goals are at times preferable to tasks since the semantics of goals are much clearer. For instance, the task in Figure 3.1 (a) can be replaced by the goal its supposed to represent, which would be $\texttt{at}\ (q)$. Similarly, the task in Figure 3.1 (b) can be replaced with $\texttt{in}\ (q)$. Below we'll crystallize this notion of planning with a hierarchy of goals.

3.1 HGN Planning

Before we formalize HGN planning, we shall first formalize Classical Planning since we reuse many concepts from the Classical Planning formalism in HGN Planning.

Classical planning. Following Ghallab, Nau and Traverso (2004), we define a classical planning domain D as a finite state-transition system in which each state s is a finite set of ground atoms of a first-order language L, and each action a is a ground instance of a planning operator o. A planning operator is a triple $o = (\text{head}(o), \text{precond}(o), \text{effects}(o))$, where $\text{precond}(o)$ and $\text{effects}(o)$ are sets of literals called o's *preconditions* and *effects*, and $\text{head}(o)$ includes o's *name* and *argument list* (a list of the variables in $\text{precond}(o)$ and $\text{effects}(o)$).

An action a is executable in a state s if $s \models \text{precond}(a)$, in which case the resulting state is $\gamma(a) = (s - \text{effects}^{-}(a)) \cup \text{effects}^{+}(a)$, where $\text{effects}^{+}(a)$ and $\text{effects}^{-}(a)$ are the atoms and negated atoms, respectively, in effects(a). A plan $\pi = \langle a_1, \ldots, a_n \rangle$ is executable in s if each a_i is executable in the state produced by a_{i-1}; and in this case we let $\gamma(s, \pi)$ be the state produced by executing the entire plan.

A *classical planning problem* is a triple $P = (D, s_0, g)$, where D is a classical planning domain, s_0 is the initial state, and g (the *goal formula*) is a set of ground literals. A plan π is a solution for P if π is executable in s_0 and $\gamma(s_0, \pi) \models g$.

HGN planning. An *HGN method* m has a head head(m) and preconditions precond(m) like those of a planning operator, and a sequence of subgoals subgoals$(m) = \langle g_1, \ldots, g_k \rangle$, where each g_i is a goal formula (a set of literals). We define the *postcondition* of m to be post$(m) = g_k$ if subgoals(m) is nonempty; otherwise post$(m) =$ precond(m). See Figure 3.2 for an example set of HGN methods for the Logistics domain.

An action a (or grounded method m) is *relevant* for a goal formula g if effects(a) (or post(m), respectively) entails at least one literal in g and does not entail the negation of any literal in g.

Some notation: if $\pi_1, \ldots, \pi_n$ are plans or actions, then $\pi_1 \circ \ldots \circ \pi_n$ denotes the plan formed by concatenating them.

An *HGN planning domain* is a pair $D = (D', M)$, where D' is a classical planning domain and M is a set of methods.

A *goal network* is a way to represent the objective of satisfying a partially ordered sequence of goals. Formally, it is a pair $gn = (T, \prec)$ such that:

- T is a finite nonempty set of nodes;

- each node $t \in T$ contains a *goal* g_t that is a DNF (disjunctive normal form) formula over ground literals;

- $\prec$ is a partial order over T.

An *HGN planning problem* is a triple $P = (D, s_0, gn)$, where D is a planning domain, s_0 is the initial state, and $gn = (T, \prec)$ is a goal network.

Definition 1. *The set of* solutions *for P is defined as follows:*

Case 1. *If T is empty, the empty plan is a solution for P.*

Case 2. *Let t be a node in T that has no predecessors. If $s_0 \models g_t$, then any solution for $P' = (D, s_0, (T', \prec'))$ is also a solution for P, where $T' = T - \{t\}$, and $\prec'$ is the restriction of $\prec$ to T'.*

Case 3. *Let a be any action that is relevant for g_t and executable in s_0. Let π be any solution to the HGN planning problem $(D, \gamma(s_0, a), (T', \prec'))$. Then $a \circ \pi$ is a solution to P.*

Case 4. *Let m be a method instance that is applicable to s_0 and relevant for g_t and has subgoals $g_1, \ldots, g_k$. Let π_1 be any solution for (D, s_0, g_1); let π_i be any solution for $(D, \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_{i-1})), g_i)$, $i = 2, \ldots, k$; and let π be any solution for $(D, \gamma(s_0, (\pi_1 \circ \ldots \circ \pi_k)), (T', \prec'))$. Then $\pi_1 \circ \pi_2 \circ \ldots \circ \pi_k \circ \pi$ is a solution to P.*

In the above definition, the relevance requirements in Cases 2 and 3 prevent classical-style action chaining unless each action is relevant for either the ultimate goal g or a subgoal of one of the methods. This requirement is analogous to (but less restrictive than) the HTN planning requirement that actions cannot appear in a plan unless they are mentioned explicitly in one of the methods. As in HTN planning, it gives an HGN planning problem a smaller search space than the corresponding classical planning problem.

The next theorem shows that HGN planning is *sound*: any HGN solution is also a solution to the corresponding classical problem.

Theorem 1 (HGN soundness)**.** *Let $D = (D', M)$ be an HGN planning domain. For every (s_0, g), the set of solutions to the HGN planning problem $P = (D, s_0, g)$ is a subset of the set of solutions to the classical planning problem $P' = (D', s_0, g)$.*

Proof. Let $\pi = \langle a_1, \ldots, a_n \rangle$ be any solution for P. From the definition of a solution, it follows that in the HGN domain D, π is executable in s_0 and $\gamma(s_0, \pi) \models g$. But $D = (D', M)$, so any action that is executable in D is also executable in the classical domain D' and produces the same effects. Thus it follows that in D', π is executable in s_0 and $\gamma(s_0, \pi) \models g$. □

Theorem 2 (HGN completeness)**.** *For every classical planning domain D, there is a set of HGN methods M such that the classical planning problem $P = (D, s_0, g)$ and the HGN planning problem $P' = ((D, M), s_0, g)$ have the same set of solutions.*

Proof.

Let X be the set of all *simple* paths in D. For each path x in X, suppose M contains methods that will specify goals for each state on x as subgoals. Thus, each subgoal will be achieved by a single action such that when the sequence of actions applied from the start of x, and the result will be the end state. Then the theorem follows. □

The following two theorems show that the HGN formalism provides expressive power equal to that of SHOP's HTN formalism:

Theorem 3 (HTN expressivity)**.** *For any HGN problem* (D, s_0, g_0)*, there exists a totally-ordered HTN problem* (D', s_0, t_{g_0}) *such that* (D, s_0, g_0) *is solvable if and only if* (D', s_0, t_{g_0}) *is solvable.*

Proof Sketch. We proceed to translate an HGN planning problem (D, s_0, g_0) into an HTN planning problem as follows: each goal formula g in D is represented by a task symbol t_g; g_0 is represented by the task symbol t_{g_0}. For each HGN method $\langle pre, \langle g_1, \ldots, g_k \rangle\rangle$, we create a new HTN method accomplishing task t_{g_k} with preconditions pre and subtasks $\langle t_{g_1}, \ldots, t_{g_k}\rangle$. Then for each t_g, we create an HTN method having a precondition of g and no subtasks. Also for every method or operator u relevant to g, we have a method accomplishing t_g having a precondition of $\neg g$ and subtasks $\langle t_u, t_g\rangle$, t_u being the task symbol corresponding to u. We then return the HTN problem $(D' \cup O, s_0, t_{g_0})$ where D' is the set of translated HTN methods and O is the set of planning operators.

It is now easy to show that any HGN decomposition trace can be mapped to a corresponding trace of the HTN problem thus constructed and vice-versa. Thus the theorem follows. □

Theorem 4 (HGN expressivity)**.** *For any totally-ordered HTN planning problem* (D, s_0, t_0)*, there is an HGN planning problem* (D', s_0, g_{t_0}) *such that* (D, s_0, t_0) *is solvable if and only if* (D', s_0, g_{t_0}) *is solvable.*

Proof Sketch. To translate an HTN planning problem[1] (D, s_0, t_0), we create predicates $\mathit{fin}_t(.)$ for each task $t(.)$ to represent task completion. We add an extra predi-

[1]We assume a single task t_0 in the initial task network; this is without loss of generality as we can replace a totally-ordered initial task network with an artificial toptask and add an extra method decomposing the toptask to the initial task network.

cate *lead* that is asserted by an artificial operator with no preconditions. We have artificial operators *assert-fin-t(.)* for each task symbol t that has precondition $\langle \textit{lead} \rangle$ and effect $\langle \neg \textit{lead}, \textit{fin}_t(.) \rangle$ Each HTN method for task t with subtasks $\langle t_1, t_2, \ldots, t_n \rangle$ is now converted to an HGN method with the same preconditions and a sequence of subgoals $\langle \textit{fin}_{t_1}, \neg \textit{fin}_{t_1}, \textit{fin}_{t_2}, \neg \textit{fin}_{t_2}, \ldots \textit{fin}_{t_n}, \neg \textit{fin}_{t_n}, \textit{lead}, \textit{fin}_t \rangle$. The $\neg fin(.)$ subgoals are used to cleanup the state for future decompositions. The HGN planning problem is $(D' \cup O, s_0, \textit{fin}_{t_0})$, where D' is the set of translated HGN methods and O is the set of classical planning operators and additional artificial operators described above. It is now easy to show that every HTN decomposition trace can be mapped to a corresponding trace of the HGN planning problem thus constructed and vice-versa. The theorem follows. □

The above theorems provide procedures to translate HGN planning problems to HTN problems and vice-versa in low-order polynomial time. This proves that HGN planning has the same expressive power as totally-ordered HTN planning.

Let HGN-PLAN-EXISTENCE be the following problem: *Given an HGN planning problem P, is there a plan that solves P?*

Theorem 5. HGN-PLAN-EXISTENCE *is decidable.*

Proof Sketch. Erol et al. [33] prove that the plan existence problem for totally-ordered HTN planning is decidable. From this and Theorem 3, the result immediately follows. □

3.2 Summary

The HGN planning formalism provides a decomposition semantics like HTN planning, but which readily corresponds to the goal semantics of classical planning and gives stronger soundness guarantees when applied to classical planning domains.

Our proof that HGN planning is as expressive as totally-ordered HTN planning means that it is capable of encoding complicated control knowledge, one of the main strengths of HTN planning. This suggests that HGN planning has the potential to be very useful both for research purposes and in practical applications.

In the next chapter, we shall develop GDP, a provably sound and complete algorithm that solves HGN planning problems.

Method for using truck ?t to move crate ?o from location ?l1 to location ?l2 in city ?c:

Head: (move-within-city ?o ?t ?l1 ?l2 ?l3 ?c)

Pre: ((obj-at ?o ?l1) (in-city ?l1 ?c)
(in-city ?l2 ?c) (truck ?t ?c) (truck-at ?t ?l3))

Sub: ((truck-at ?t ?l1) (in-truck ?o ?t)
(truck-at ?t ?l2) (obj-at ?o ?l2)))

Method for using airplane ?plane to move crate ?o from airport ?a1 to airport ?a2:

Head: (move-between-airports ?o ?plane ?a1 ?a2)

Pre: ((obj-at ?o ?a1) (airport ?a1) (airport ?a2) (airplane ?plane))

Sub: ((airplane-at ?plane ?a1) (in-airplane ?o ?plane)
(airplane-at ?plane ?a2) (obj-at ?o ?a2)))

Method for moving ?o from location ?l1 in city ?c1
to location ?l2 in city ?c2, via airports ?a1 and ?a2:

Head: (move-between-cities ?o ?l1 ?c1 ?l2 ?c2 ?a1 ?a2)

Pre: ((obj-at ?o ?l1) (in-city ?l1 ?c1) (in-city ?l2 ?c2) (different ?c1 ?c2)
(airport ?a1) (airport ?a2) (in-city ?a1 ?c1) (in-city ?a2 ?c2))

Sub: ((obj-at ?o ?a1) (obj-at ?o ?a2) (obj-at ?o ?l2)))

Figure 3.2: HGN methods for transporting a package to its goal location in the Logistics domain.

Chapter 4: The **G**oal **D**ecomposition Planner (GDP)

This chapter introduces the **G**oal **D**ecomposition Planner, an HGN planning algorithm that is sound and complete with respect to Definition 1. As a result, it requires a *complete* set of HGN methods: in other words, the user needs to provide methods for handling every possible scenario that the planner could encounter in the target domain. This is similar to restrictions that popular HTN planners like SHOP/SHOP2 [24, 25] impose.

GDP takes as input a HGN domain model $D = (O, M)$ where O is the set of planning operators and M is the set of HGN methods, the initial state s_0 and the goal network gn that needs to be achieved.

4.1 Algorithm

Algorithm 1 is GDP, our HGN planning algorithm. It works as follows:

In Line 2, if gn is empty then the goal has been achieved, so GDP returns π. Otherwise, GDP selects a goal g in gn without any predecessors (Line 3). If g is already satisfied, GDP removes g from gn and calls itself recursively on the resulting goal network.

In Lines 7-8, if no actions or methods are applicable to s and relevant for g, then GDP returns failure. Otherwise, GDP nondeterministically chooses an action/method u

Algorithm 1: A high-level description of GDP. Initially, D is an HGN planning domain, s is the initial state, gn is the goal network, and π is $\langle\rangle$, the empty plan.

```
Procedure GDP($D, s, gn, \pi$)
begin
    if $gn$ is empty then return $\pi$
    $g \leftarrow$ goal formula in $gn$ with no predecessors
    if $s \models g$ then
        remove $g$ from $gn$ and return GDP($D, s, gn, \pi$)
    $U \leftarrow \{$actions and method instances that are relevant for $g$ and applicable to $s\}$
    if $U = \emptyset$ then return failure
    nondeterministically choose $u \in U$
    if $u$ is an action then append $u$ to $\pi$ and set $s \leftarrow \gamma(s, u)$
    else insert subgoals($u$) as predecessors of $g$ in $gn$
    return GDP($D, s, gn, \pi$)
end
```

from U.

If u is an action, then GDP computes the next state $\gamma(s, u)$ and appends u to π. Otherwise u is a method, so GDP inserts u's subgoals at the front of g. Then GDP calls itself recursively on gn.

4.1.1 Formal Properties

The following theorems show that GDP is sound and complete:

Theorem 6 (GDP soundness)**.** *Let $P = (D, s_0, g)$ be an HGN planning problem. If a nondeterministic trace of* $\mathsf{GDP}(D, s_0, \langle g \rangle, \langle \rangle)$ *returns a plan π, then π is a solution for P.*

Proof Sketch. The proof is by induction on n, the length of π. When $n = 0$ (i.e. $\pi = \langle \rangle$), this implies that s_0 entails g. Hence, by Case 1 of the definition of a solution, π is a solution for P. Suppose that if GDP returns a plan π of length $k < n$, then π is a solution for P. At an invocation suppose GDP returns π of length n. The proof proceeds by showing the following. When GDP chooses an action or a method for the current goal at any invocation, then by induction, the plans returned from those calls are solutions to the HGN planning problems in those calls. Hence, by definition of solutions for P, π is a solution for P. □

Theorem 7 (GDP completeness)**.** *Let $P = (D, s_0, g)$ be an HGN planning problem. If π is a solution for P, then a nondeterministic trace of* $\mathsf{GDP}(D, s_0, \langle g \rangle, \langle \rangle)$ *will return π.*

Proof Sketch. The proof is by induction on n, the length of π. When $n = 0$, this implies that the empty plan is a solution for P and that $s_0 \models g$. Hence GDP would return $\langle \rangle$ as a solution. Suppose that if P has a solution of length $k < n$, then GDP will return it. At any invocation, the proof proceeds to show by induction the following. If GDP chooses an action a, then one of the nondeterministic traces of the subsequent call to GDP must return $\pi = a \circ \pi'$ where π' is a solution for the problem $P' = (D, \gamma(s_0, a), g)$. If GDP chooses a method m relevant to g with subgoals $g_1, g_2, \ldots g_l$, then there must exist a sequence of plans $\pi_1, \pi_2, \ldots, \pi_{l+1}$ that constitute π and GDP will return each π_i as a solution each goal g_i from the state $\gamma(s_0, (\pi_1 \circ \pi_2 \circ \cdots \circ \pi_{i-1}))$. Then the theorem follows. □

4.1.2 Domain-Independent Heuristics

GDP can easily be modified to incorporate heuristic functions similar to those used in classical planning. The modified algorithm, which we will call GDP-h (where h is the heuristic function) in the experiments, is like Algorithm 1, except that Lines 9–12 are replaced with the following:

```
sort $U$ with $h(u), \forall u \in U$
foreach $u \in U$ do
    if $u$ is an action then
        append $u$ to $\pi$; remove $g$ from $G$;
        $s \leftarrow \gamma(s, u)$
    else push subgoals($u$) into $G$
    $\pi \leftarrow$ GDP($D, s, G, \pi$)
    if $\pi \neq$ failure then return $\pi$
return failure
```

Intuitively, this replaces the nondeterministic choice in GDP with a deterministic choice dictated by h. GDP-h uses h to order U, then attempts to decompose the current goal g in that order.

As an example, here is how we compute a variation of the Relaxed Graphplan heuristic used by the FF planner [7]. At the start of the planning process, we generate a relaxed planning graph PG from the start state s_0 to its fixpoint. Let $l_{PG}(p)$ be the first propositional level in which p appears in PG. Then $h_{s,G}(u)$, the heuristic value of applying action/method u in a state s to achieve the goals in the list G, is as follows:

$$h_{s,G}(u) = \begin{cases} 1 + \max\limits_{p \in G} l_{PG}(p) - \max\limits_{p \in \gamma(s,u)} l_{PG}(p), & \text{if } u \text{ is an action,} \\ \max\limits_{p \in G \cup sub(u)} l_{PG}(p) - \max\limits_{p \in s} l_{PG}(p), & \text{if } u \text{ is a method.} \end{cases}$$

Intuitively, what h estimates is the distance between the first level in which the literals in G are asserted and the first level in which the current state is asserted. When u is a method, since any plan generated via u has to achieve $sub(u)$ enroute, it considers the set $G \cup sub(u)$ instead as the goal.

Note that this gives weaker heuristic values than the original FF heuristic since we do not generate a relaxed plan and use its length as the heuristic value. However, we use this variant of the heuristic since it is much more efficiently computable without compromising too much on search control. The strength of the heuristic is not as critical here as in classical planning, since the HGN methods themselves constrain what part of the space gets searched.

4.2 Experimental Evaluation

We implemented GDP in Common Lisp, and compared it with SHOP2 and the classical planner FF in five different planning domains:[1] These included the well-known Logistics [34], Blocks-World [35], Depots [36], and Towers of Hanoi [28] domains, and a new *3-City Routing* domain that we wrote in order to provide a domain in which the planners' domain models would not be of much help. The following questions motivated our experiments:

- *How does* GDP*'s performance (plan quality and running time) compare with SHOP2's?* In order to investigate this question, we were careful to use domain models for SHOP2 and GDP that encoded basically the same control information.[2]
- *What is the relative difficulty of writing domain models for* GDP *and SHOP2?* We had no good way to measure this directly;[3] but as a proxy for it, we (i) measured the relative sizes of the SHOP2 and GDP domain models, and (ii) examined the domain models to find out the reasons for the difference in size.

[1]We used SHOP2 instead of SHOP for two reasons: (1) its algorithm is identical to SHOP's when restricted to totally-ordered subtasks, and (2) since its implementation includes many enhancements and optimizations not present in SHOP, it provides a more rigorous test of GDP.

[2]An important aspect of SHOP2's domain models is the use of Horn-clause inference to infer some of the preconditions. So that we could write GDP domain models equivalent to SHOP2's, we included an identical Horn-clause inference engine in GDP.

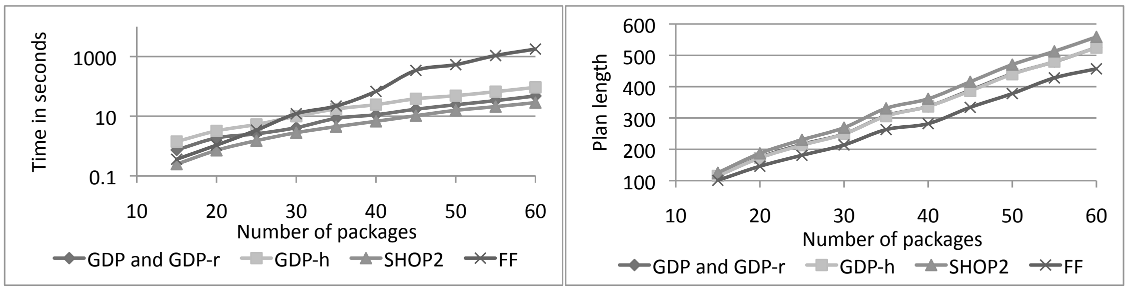
[3]That would have required a controlled experiment on a large number of human subjects, each of whom has equal amounts of training and experience with both GDP and SHOP2. We have no feasible way to perform such an experiment.

- *How useful is* GDP-h*'s heuristic function when the domain model is strong?* For this, we compared GDP-h with GDP on the Logistics, Blocks World, and Depots domains.
- *When the domain model is weak, how much help does* GDP-h*'s heuristic function provide?* For this, we compared GDP-h's performance with GDP's on the 3-City Routing domain.
- *Since* GDP-h*'s heuristic function is loosely based on FF's, how does* GDP-h*'s performance compare to FF's?* For this purpose, we included FF in our experiments.
- *Is* GDP *as sensitive as SHOP2 is to the order in which the methods appear in the domain model?* To investigate this question, we took our domain models for SHOP2 and GDP, and rearranged the methods into a random order. In experimental results that follow, we use the names SHOP2-r and GDP-r to refer to SHOP2 and GDP with those domain models.

The GDP source code, and the HGN and HTN domain models used in our experiments, are available at http://www.cs.umd.edu/projects/planning/data/shivashankar12hierarchical/.

4.2.1 Planning Performance

To compile and execute GDP, GDP-h, and SHOP2, we used Allegro Common Lisp 8.0. For FF, we used the open-source C implementation from the FF web site. All experiments were run on 2GHz dual-core machines with 4GB RAM. We set a time limit of two hours per problem, and data points not solved within the required time limit were discarded.



(a) (b)

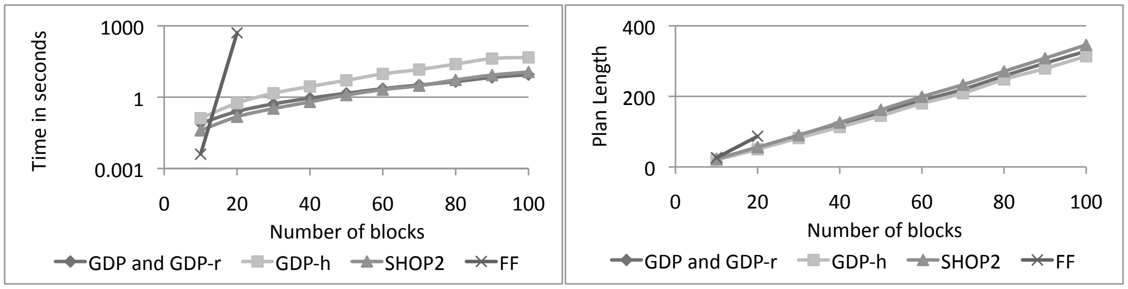
Figure 4.1: Average running times (in logscale) and plan lengths in the Logistics domain, as a function of the number of packages. Each data point is an average of the 10 problems from the SHOP2 distribution. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the methods had mutually exclusive preconditions.

The Logistics Domain. For SHOP2, we used the Logistics domain model in the SHOP2 distribution. For GDP and GDP-h, we wrote the methods in Fig. 3.2 (these methods are easy to prove complete [37]). For the experiments, we used the Logistics Domain problems in the SHOP2 distribution. These included ten n-package problems for each of $n = 15, 20, 25, \ldots, 60$.

Figure 4.1 shows a comparison of running times and plan lengths of the planners in this domain. The running times of GDP, GDP-h and SHOP2 were very similar, showing that even on easy domains with strong domain models, the heuristic does not add much overhead to GDP-h's running time. FF's running times, however, grew much faster: with 60 packages, FF was nearly two orders of magnitude slower than SHOP2.

The plans produced by GDP and GDP-h were of nearly the same length, and the plans produced by SHOP2 were slightly longer. FF produced the shortest plans; this indicates that its heuristic function was slightly stronger than the relaxed version we used in GDP-h.

SHOP2-r did not terminate on any of the instances, while GDP-r performed identically to GDP. In fact, we observed that the same was true across all of the domains in our experimental study. We defer the explanation of this to Section 4.2.3.



(a) (b)

Figure 4.2: Average running times (in logscale) and plan lengths in the Blocks World domain, as a function of the number of blocks. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive. FF was unable to solve problems involving more than 20 blocks.

The Blocks World. For SHOP2, we used the domain model included in SHOP2's distribution. For GDP and GDP-h we used a much more compact domain model consisting of three methods (shown here as pseudocode):

- **To achieve** $\texttt{on}(x, y)$

precond: y is in its final position[4]

subgoals: achieve `clear(`x`)`, `clear(`y`)` and `on(`x, y`)`

- **To achieve** `clear(`x`)`

  precond: `on(`y, x`)`

  subgoals: achieve `clear(`y`)` and then `clear(`x`)`

- **To achieve** `on-table(`x`)`

  precond: None

  subgoals: achieve `clear(`x`)` and then `on-table(`x`)`

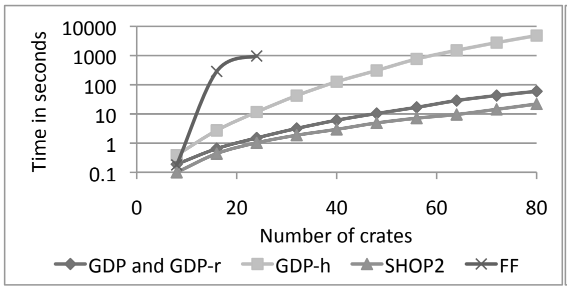
As shown in Figure 5.2, GDP and SHOP2 took nearly identical times to solve the problems, with GDP-h taking slightly longer due to its heuristic computation overhead. FF, which is known to have problems with the Blocks World [35], was unable to solve problems with more than 20 blocks.

As shown in the figure, GDP, GDP-h and SHOP2 produced solution plans of similar length, with GDP-h producing the shortest plans. FF produced significantly longer plans than the other three planners, even for the problems it managed to solve.

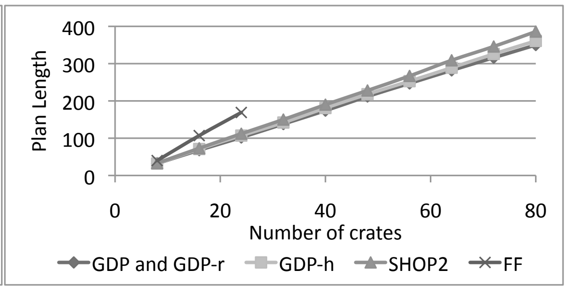
The Depots Domain. For SHOP2, we used the Depots domain model from the SHOP2 distribution. For GDP and GDP-h, we simply stitched together relevant parts of the Logistics and Blocks-World domain models, and adapted them to obtain an HGN Depots domain model that encoded the same control information.

As shown in Figure 5.3, GDP and SHOP2 took similar times to solve the problems. However, GDP-h's running times grew much faster than GDP or SHOP2, indicating that

[4]Inferred using Horn clauses (see footnote 2).



(a)

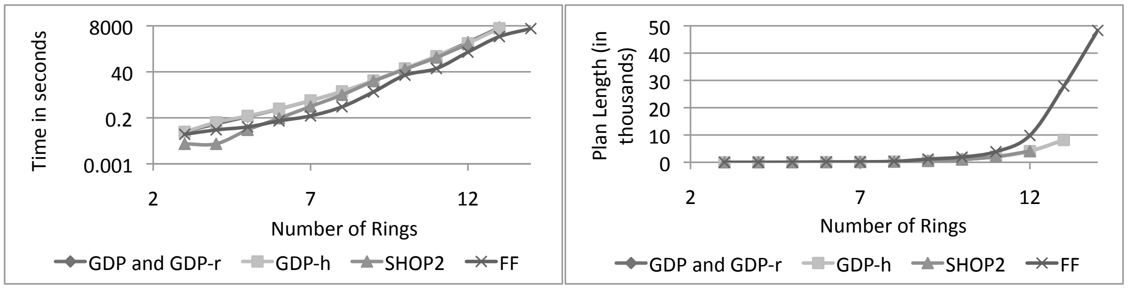


(b)

Figure 4.3: Average running times (in logscale) and plan lengths in the Depots domain, as a function of the number of crates. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive. FF was unable to solve problems involving more than 24 crates.

the overhead of the heuristic can increase with the complexity of the domain. FF was unable to solve any problems of size greater than 24 crates.

With respect to plan lengths, GDP and GDP-h produced almost identical plans, with SHOP2 producing slightly longer plans than GDP. For the problem sizes it could handle, FF produced significantly longer plans than the other three planners.



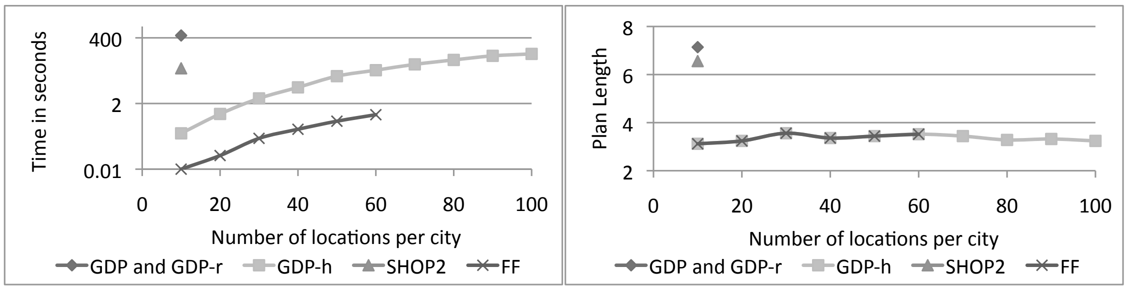
(a) (b)

Figure 4.4: Average running times (in logscale) and plan lengths in the Towers of Hanoi domain, as a function of the number of rings. Each data point is an average of 10 runs. There are no data points for SHOP2-r because it could not solve any of the problems. GDP and GDP-r performed identically because the preconditions of the methods were mutually exclusive.

Towers of Hanoi. We wrote domain models for SHOP2 and GDP that encoded an algorithm to produce optimal solution plans (i.e., length $2^n - 1$ for an n-ring problem).

Figure 4.4 shows the planners' runtimes and plan lengths. As expected, GDP, GDP-h and SHOP2 returned optimal plans whereas FF returned significantly sub-optimal plans.

However, while GDP, GDP-h, SHOP2 and FF had similar runtimes up to problems of size 12, SHOP2 could not solve the larger problems due to a stack overflow, and GDP could not solve the 14-ring problem within the time limit. We believe this is basically an implementation issue: both GDP and SHOP2 had recursion stacks of exponential size, whereas FF (since it never backtracks) did not.



(a) (b)

Figure 4.5: Average running times (in logscale) and plan lengths in the 3-City Routing domain, as a function of the number of locations per city. Each data point is an average of 25 randomly generated problems. There are no data points for SHOP2-r because it couldn't solve any problems. FF couldn't solve problems involving more than 60 locations while GDP and SHOP2 could not solve problems with more than 10 locations. GDP and GDP-r performed identically because there was only one method in the domain model.

3-City Routing. In the four planning domains discussed above, the GDP and SHOP2 domain models pruned the search space enough that GDP-h's heuristic function could not reduce it much further (if at all). In order to examine the performance of the planners in a domain with a weak domain model, we constructed the *3-City Routing* domain. In this domain, there are three cities c_1, c_2 and c_3, each containing n locations internally connected by a network of randomly chosen roads. In addition, there is one road between a randomly chosen location in c_1 and a randomly chosen location in c_2, and similarly another road between locations in c_2 and c_3. The problem is to get from a location in c_1 or c_3 to a goal location in c_2.

We randomly generated 25 planning problems for each value of n, with n varying from 10 to 100. For the road networks, we used near-complete graphs in which 20% of the edges were removed at random. Note that while solutions to such problems are typically very short, the search space has extremely high branching factor, i.e. of the order of n. For GDP and GDP-h, we used a single HGN method, shown here as pseudocode:

- **To achieve** `at(`b`)`

  precond: `at(`a`)`, `adjacent(`c, b`)`

  subgoals: achieve `at(`c`)` and then `at(`b`)`

By applying this method recursively, the planner can do a backward search recursively from the goal location to the start location.

To accomplish the same backward search in SHOP2, we needed to give it three methods, one for each of the following cases: (1) goal location same as the initial location, (2) goal location one step away from the initial location, and (3) arbitrary distance between the goal and initial locations.

As Figure 4.5 shows, GDP and SHOP2 did not solve the randomly generated problems except the ones of size 10, returning very poor solutions and taking large amounts of time in the process. GDP-h, on the other hand solved all the planning problems quickly, returning near-optimal solutions. The reason for the success of GDP-h is that the domain knowledge specified above induce an unguided backward search in the state space and the planner uses the domain-independent heuristic to select its path to the goal.

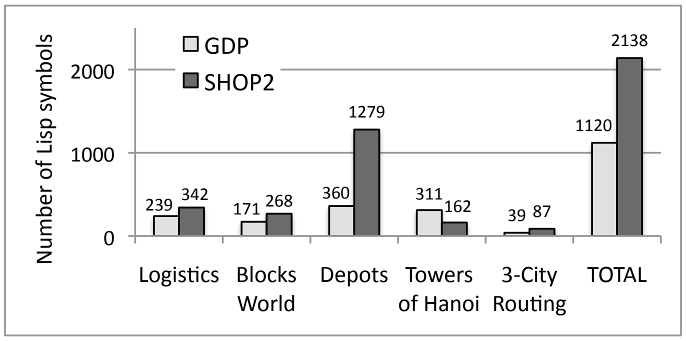


Figure 4.6: Sizes (number of Lisp symbols) of the GDP and SHOP2 domain models.

FF was able to solve all problems up to $n = 60$ locations, after which it could not even complete parsing the problem file. We believe this has to do with FF grounding all the actions right in the beginning, which it could not do for the larger problems.

4.2.2 Domain Authoring

When writing the domain models for our experiments, it seemed to us that writing the GDP domain models was easier than writing the SHOP2 domain models—so we made measurements to try to verify whether this subjective impression was correct.

Figure 4.6 compares the sizes of the HGN and HTN domain descriptions of the planning domains. In almost all of them, the domain models for GDP were much smaller than those for SHOP2. There are three main reasons why:

- To specify how to achieve a logical formula p in the HTN formalism, one must create a new task name t and one or more methods such that (i) the plans generated by these methods will make p true and (ii) the methods have syntactic tags saying that they are relevant for accomplishing t. If there is another method m' that makes p true but does not have such a syntactic tag, the planner will never consider using m' when it is trying to achieve p. In contrast, relevance of a method in HGN planning is similar to relevance of an action in classical planning: if the effects of m' include p, then m' is relevant for p.

- Furthermore, suppose p is a conjunct $p = p_1 \wedge \ldots \wedge p_k$ and there are methods $m_1, \ldots, m_k$ that can achieve $p_1, \ldots, p_k$ piecemeal. In HGN planning, each of these methods is relevant for p if it achieves some part of p and does not negate any other part of p. In contrast, those methods are not relevant for p in HTN planning unless the domain description includes (i) a method that decomposes t into tasks corresponding to subsets of $p_1, \ldots, p_k$, (ii) methods for those tasks, and (iii) an explicit check for deleted-condition interactions.[5] This can cause the number of HTN methods to be much larger (in some cases exponentially larger) than the number of HGN methods.

- In recursive HTN methods, a "base-case method" is needed for the case where nothing needs to be done. In recursive HGN methods, no such method is needed, because the semantics of goal achievement already provide that if a goal is already true, nothing needs to be done.

The Towers of Hanoi domain was the only one where the HGN domain model was larger than the corresponding HTN domain model. In this domain, the HGN domain model needed two extra actions, *enable* and *disable*, to alternately insert and delete a spe-

[5]In the HTN formalism in [22], one way to accomplish (iii) is to specify t as the syntactic form $achieve(p)$, which adds a constraint that p must be true after achieving t. But that approach is inefficient in practice because it can cause lots of backtracking. In the blocks-world implementation in the SHOP2 distribution, (iii) is accomplished without backtracking by using Horn-clause inference to do some elaborate reasoning about stacks of blocks.

cial atom in the state. They were needed in order to control the applicability of the *move* operator to ensure optimality. Thus, it seems like HGNs are a better fit for domains for which domain knowledge can be specified in a declarative manner, while HTNs provide more concise domain models for domains which require modeling procedural/operational models.

is possibly easier than HTNs. However, this comes with a caveat; our study was very rudimentary, and we didn't control against various factors such as familiarity with either HGN/HTN planning, order in which models were written, user intelligence, etc. A more conclusive study would have been to conduct a user study and evaluate ease of domain modeling for a number of different users with different degrees of familiarity with either HGN or HTN planning. We didn't have the resources to conduct such a study; so while our measurements lend support to our claim, the proof isn't conclusive.

4.2.3 Discussion

We have seen from our experimental study that HGN domain models are considerably more succinct than the corresponding HTN models. We also saw that this compactness came at no extra cost; GDP's performance compared favorably to that of SHOP2's across all domains. Runtimes of the heuristic-enhanced planner GDP-h were, for the most part, comparable to those of GDP's and SHOP2's, indicating that our heuristic does not add a significant overhead to the planning time. Lengths of plans returned by GDP-h were nearly always better than GDP's and SHOP2's. This difference was especially amplified in cases where the planners had weak domain models; in such cases, the heuristic provided critical search control to GDP-h, thus helping it terminate quickly with good solutions.

In our experiments, SHOP2-r did not solve any of the problems. The reason for this was SHOP2's heavy reliance on the method order in its domain model, especially the placement of "base-cases" for recursion. For GDP-r, shuffling HGN methods had no effect at all on performance. This was because the methods in our HGN domain models had mutually exclusive preconditions, hence at most one of them was applicable. In domains where more than one method is applicable at once, GDP-r should (like SHOP2-r) perform badly when presented with methods in the wrong order.

4.3 Summary

GDP is a HGN planning algorithm that solves planning problems by leveraging domain-specific advice in the form of HGN methods, which decompose goals into sequences of subgoals. GDP uses these methods to recursively decompose the given problem into smaller and smaller subproblems until a plan can be generated. GDP is provably sound and complete with respect to the definition of solutions in HGN planning (Definition 1).

Since GDP operates on hierarchies of goals, it can leverage heuristics from Classical Planning (which is also a goal-based planning formalism) to help make some of the decisions GDP has to make during the planning process.

The experimental study across a variety of domains demonstrates that GDP not only matches the performance of state-of-the-art HTN planners like SHOP2 when given sophisticated domain-specific advice, but also that in cases when we don't have sophisticated advice to provide, GDP can use its heuristic search capabilities to provide the missing guidance. We also saw that HGN domain models are considerably more succinct than their HTN counterparts, due to the fact that HGN planners automatically infer a number of facts that need to be explicitly provided to HTN planners.

An important disadvantage that GDP inherits from HTN planning, due to similarities in the way problem decomposition is set up, is that it still requires a complete set of HGN methods to guarantee solvability for the target domain. In the next chapter, we shall see how GoDeL, another HGN planning algorithm, extends GDP to address these limitations.

Chapter 5: The GoalDecomposition with Landmarks Planner (GoDeL): HGN Planning with Partial Domain Knowledge

The planning algorithm GDP proposed in Chapter 4 requires a complete set of HGN methods; in other words, the domain author needs to design methods capable of generating plans for every possible planning problem in a given domain. This is hard to guarantee, especially in real-world domains, which can get very complicated. Moreover, the completeness requirement is hard; even if a single method is missing, the planner can fail and not find a solution, even if the action models allow one.

The purpose of this work is to provide a way of overcoming this difficulty. In particular, the main contributions are the following:

- We describe a simple extension to the HGN planning formalism that allows falling back to classical planning techniques whenever there are gaps in the methods provided. Like the strict version of HGN planning described in Chapter 4, this formalism involves hierarchical decomposition, but goals may be achieved regardless of whether there are methods for achieving them. This is described in Section 6.3.

- We present the GoDeL (**Go**al **De**composition with **L**andmarks) planning algorithm which is sound and complete with respect to the abovementioned version of HGN planning. So, GoDeL can generate plans using the given HGN methods, but can also find solutions without them; it accomplishes this by combining standard HGN method decomposition with classical action chaining, domain-independent heuristics and planning landmarks [38]. We present several theorems, including soundness and completeness irrespective of whether GoDeL is given a complete set of methods.

- We present experimental results for an implementation of GoDeL (for landmark generation in GoDeL, we used code from the well-known LAMA planner [12]). The results show that GoDeL works correctly with partial planning knowledge (i.e., an incomplete set of methods), and its performance improves as more planning knowledge is given. When given complete domain knowledge, GoDeL performs as well or better than the heuristic-enhanced planner GDP-h (from Chapter 4).

5.1 Formalism

The change to the HGN formalism needed to work with partial sets of HGN methods is quite simple; we simply remove the *relevance* requirement for applying actions in Definition 1, Case 3. Therefore, given a planning problem $P = (D, s_0, gn)$ and an action a applicable in the state s_0, we can progress to the problem $P' = (D, \gamma(s_0, a), gn)$; thus if π' is a solution to P', then $a \circ \pi'$ is a solution to P. Recall that the original formalism would not allow this, unless a, in addition to being applicable in s_0, was also immediately relevant to some goal g in gn having no predecessors. Therefore, this change now allows decoupling progressing the current state (via action application) from expanding the goal network (via method decomposition).

In fact, by removing the *relevance* requirement, we in effect reduce the definition to one that is equivalent to classical planning. This is because the definition allows for purely chaining actions together until we achieve the goal network, while ignoring the methods provided altogether. Therefore, we can recast the definition as a simple extension of classical planning that solves for goal networks, and treat HGN methods as auxiliary advice provided to the algorithm.

Definition 2. *The set of* solutions *for P is defined as follows:*

Case 1. *If T is empty, the empty plan is a solution for P.*

Case 2. *Let t be a node in T that has no predecessors. If $s_0 \models g_t$, then any solution for $P' = (D, s_0, (T', \prec'))$ is also a solution for P, where $T' = T - \{t\}$, and $\prec'$ is the restriction of $\prec$ to T'.*

Case 3. *If action a is applicable in s_0 and π is a solution for $P'' = (D, \gamma(s_0, a), gn)$, then $a \circ \pi$ is a solution for P.*

In cases where gn contains just a single node, the definitions of P and its solutions reduce to the conventional definitions of a classical planning problem and its solutions. In such cases we will say that P is *classical.*

Notice that unlike conventional definitions of HTN and HGN planning problems (e.g., [39, Chapter 11]), in which methods are essential to the definition of a solution, our definition of a planning problem and its solutions does not involve methods at all. In our planning algorithm (see the next section), the purpose of methods is to provide guidance for what parts of the search space to examine next.

5.2 Planning Algorithm

Algorithm 2: A nondeterministic version of GoDeL. Initially, (D, s, gn) is the planning problem, M is a set of methods, and π is $\langle\rangle$, the empty plan. used_methods, a global variable, is a mapping from states to methods applied in those states; it is initially set to the empty map. subgoals_inferred is a boolean initially set to false.

```
Procedure GoDeL $(D, s, gn, M, \pi)$
if $gn$ is empty then return $\pi$
nondeterministically choose a goal formula $g$ in $gn$ without any predecessors
if $s \models g$ then
    return GoDeL$(D, s, gn - \{g\}, M, \pi)$
$\mathcal{U} \leftarrow$ {operator and method instances applicable to $s$ and relevant to $g$}
$\mathcal{U} \leftarrow \mathcal{U} -$ used_methods$[s]$
while $\mathcal{U}$ is not empty do
    nondeterministically remove a $u$ from $\mathcal{U}$
    if $u$ is an action then
        res1 $\leftarrow$ GoDeL$(D, \gamma(s, u), gn, M, \pi \circ u)$
    else
        add $u$ to used_methods$[s]$ and set subgoals_inferred to false
        res1 $\leftarrow$ GoDeL$(D, s, \text{subgoals}(u) \circ gn, M, \pi)$
    if res1 $\neq$ failure then return res1

```

```
17 if subgoals_inferred $\neq$ true then
18     subgoals_inferred $\leftarrow$ true
19     lm $\leftarrow$ Infer-Subgoals $(D, s, g)$
20     if lm $\neq \emptyset$ then
21         res2 $\leftarrow$ GoDeL $(D, s, \mathrm{lm} \circ gn, M, \pi)$
22         if res2 $\neq$ failure then return res2
23 $\mathcal{A} \leftarrow$ {operator instances applicable to $s$}
24 if $\mathcal{A} = \emptyset$ then  return failure
25 nondeterministically choose an $a \in \mathcal{A}$
26 return GoDeL $(D, \gamma(s, a), gn, M, \pi \circ a)$
```

The **Go**al **De**composition with **L**andmarks (GoDeL) planning algorithm, when given a problem (D, s_0, gn), works as follows: It chooses a g from gn that has no predecessors and (1) first attempts to decompose g using one of the given methods, (2) if not, then it automatically infers subgoals to insert into gn using landmark-based techniques, and (3) if neither of the above steps work, it falls back to traditional action-chaining.

Algorithm 4 describes the GoDeL planning algorithm. It takes as input a planning problem $P = (D, s, gn)$, a set of methods M, and the partial plan π generated so far. Lines 2 – 5 specify the base cases of GoDeL. If these are not satisfied, the algorithm nondeterministically chooses a goal g with no predecessors and generates $\mathcal{U}$, all method and operator instances applicable in s and relevant to g. It then nondeterministically chooses a $u \in \mathcal{U}$ to progress the search (Lines 6 – 9). If u is an action, the state is progressed to $\gamma(s, u)$ (Line 20). If u is a method, the subgoals of u are added to gn, adding edges to preserve the total order imposed by subgoals(u) (Line 24). In either case, GoDeL is invoked recursively on the new planning problem.

If GoDeL fails to find a plan in the previous step, it then uses the Infer-Subgoals procedure to infer lm, a network of subgoals that are to be achieved enroute to achieving g (Line 19). The subgoals are added to gn, adding edges to preserve the partial order in lm. The algorithm is then recursively invoked on the new planning problem (Line 21). If this call also returns failure, then the algorithm falls back to action chaining (Lines 23 – 26), returning failure if no actions are applicable in s.

GoDeL maintains a global map used_methods that keeps track of the set of method instances already used in a given state s. This is used to prune out used methods from the set of options $\mathcal{U}$ (Line 7) and is updated when a new method is applied (Line 13). Similarly, GoDeL also uses subgoals_inferred, a boolean variable that keeps track of whether the goal network has been modified since the last time the Infer-Subgoals procedure is invoked, ensuring that the latter is invoked only once between changes to the goal network. As we shall see in Section 5.3, these steps are critical in ensuring GoDeL's completeness.

Subgoal Inference. We now describe the subgoal inference technique used in GoDeL. This aspect of GoDeL is motivated by the fact that sometimes the planner may have methods that tell how to solve some subproblems, but not the top-level problem. For instance, the `move-within-city` method in Figure 3.2 would not be applicable to problems involving transporting packages across cities, but it *is* applicable to the subprob-

Algorithm 3: Procedure to deduce possible subgoals for GoDeL to use. It takes as input a planning problem $P = (D, s, g)$, and outputs a poset of subgoals. It uses LMGEN, an abstract landmark generation algorithm that takes P and generates a DAG of landmarks for it.

```
1 Procedure Infer-Subgoals $(D, s, g)$
2 $(V, E) \leftarrow \mathsf{LMGEN}(D, s, g)$
3 $L \leftarrow \{v \in V : s \not\models v, g \not\models v$ and $\exists$ a method $m$ s.t. goal$(m)$ is relevant to $v\}$
4 if $L = \emptyset$ then return $\emptyset$
5 $E_L \leftarrow \{(u, v) : u, v \in L$ and $v$ is reachable from $u$ in $(V, E)\}$
6 return $(L, E_L)$
```

lems of moving the package between the start and goal locations and the corresponding airports. A natural question that then arises is the following: *How can we automatically infer these subproblems for which the given methods are relevant?*

To answer the above question, we use landmarks. A *landmark* for a planning problem P [12, 38] is a fact that is true at some point in every plan that solves P. A *landmark graph* is a directed graph whose nodes are *landmarks* and edges denote orderings between these landmarks. Therefore, if there is an edge between two landmarks l_i and l_j, this implies that l_i is true before l_j in every solution to P.

Therefore, a landmark for a problem P can be thought of as a subgoal that every solution to P must satisfy at some point. We can, as a result, use any landmark generation algorithm (for example, [12, 38]) to automatically infer subgoals (and orderings between them) for which the given methods are relevant.

Algorithm 3 is Infer-Subgoals, the subgoal inference procedure. It uses an abstract landmark generation procedure LMGEN that takes as input a classical planning problem P and generates a DAG of landmarks.

Infer-Subgoals begins by computing the landmark graph (V, E) for the input problem P. It then computes L, the subset of landmarks in V that have relevant methods (Line 3). It does not consider trivial landmarks such as literals true in the state s or part of the goal g. E_L is the set of all edges between landmarks $l_i, i_j \in L$ such that there exists a path from l_i to l_j in (V, E) (Line 5). Infer-Subgoals then returns the resulting network of landmarks $\text{lm} = (L, E_L)$.

5.3 Formal Properties

To show that GoDeL is sound and complete, it is necessary to show that GoDeL deals correctly with some "corner cases" involving methods and landmarks, e.g., that it can detect and recover from cases where the method recursion doesn't terminate. To keep the proof sketches simple, the following theorems all deal with the case where the planning problem $P = (D, s_0, gn)$ is classical, i.e., gn contains one node, hence one goal g. It is straightforward to generalize the theorems to arbitrary goal networks.

Theorem 8 (soundness). *Let $P = (D, s_0, gn)$ be as described above, and M be a set of methods. If a nondeterministic trace of* GoDeL$(D, s_0, gn, M, \langle\rangle)$ *returns a plan π, then π is a solution for P.*

Proof Sketch. The proof proceeds to show that π is executable by observing that GoDeL inserts an action only in a state in which it is executable. Moreover, since the invariant that g is always the last node in the goal network is maintained throughout GoDeL's execution, it must have been the last goal to be removed from the goal network. By Definition 1, Case 2, this must be in a state that satisfies gn's goal g. □

A planning problem P' is *reachable* by GoDeL from P if P' is in the search space produced by invoking GoDeL on P, i.e., if one of GoDeL's nondeterministic traces includes P'.

Theorem 9 (search space finiteness)**.** *If $P = (D, s_0, gn)$ is as described above, M is a set of methods, and* LMGEN *is a sound landmark generation algorithm, then the set of planning problems reachable by* GoDeL *from P is finite.*

Proof Sketch. Firstly, note that there are two ways by which goals get added to the goal network, method application and subgoal inference. Let us first consider the former. Note that GoDeL never uses the same method instance twice to decompose a goal network in the same state (see Lines 7 and 13 in Algorithm 4). Since there are finitely many methods and the size of the planning problem is finite, it follows that the size of the largest goal network reachable via method decomposition from gn's goal g in the state s is bounded.

With respect to subgoal inference via landmarks, note that for any classical planning problem $P_c = (D, s, g)$, the set of landmarks $\mathcal{L}$ for P_c is finite. Since LMGEN is sound, the subgoals it infers belongs to this set. Thus, it is easy to show that Infer-Subgoals can be run only a finite number of times, adding at most $|\mathcal{L}|$ nodes to the goal network each time. Therefore, we can place an upper bound on the goal network size in a given state s. The bound on the size of the largest goal network reached by GoDeL is thus at most $|S|$

times the previous bound, S being the set of reachable states. Using this bound, we can subsequently bound the number of reachable goal networks. Since S is finite, the number of reachable planning problems is also finite. $\square$

Theorem 10 (completeness)**.** *Let $P = (D, s_0, gn)$ be as described above, and M be a set of methods. If P is solvable and* LMGEN *is sound, a nondeterministic trace of* GoDeL$(D, s_0, \langle g \rangle, \langle\rangle)$ *will return a solution π for P.*

Proof Sketch. From Theorem 9 and the fact that GoDeL can always fall back to action-chaining to generate a solution if method decomposition does not work, the theorem follows. $\square$

5.4 Implementation and Experiments

We implemented GoDeL in C++. The nondeterministic choice among the options in the set $\mathcal{U}$ (Lines 6–9 in Algorithm 4) is implemented using depth-first search. We sort $\mathcal{U}$ using a variant of the GDP heuristic (described in Chapter 4).[1] For generating landmarks in Infer-Subgoals, we use LAMA's landmark generation code [12], which generates sound, acyclic landmark graphs. GoDeL's action chaining (line 23) uses depth-first search with the various actions sorted by the FF heuristic [7] value.

[1]Note that the GDP heuristic, while being a HGN planning heuristic, doesn't use the given methods in its heuristic computation. So the possibility that GoDeL might have only a partial set of methods doesn't affect the heuristic computation.

5.4.1 Experimental Design

The following research questions motivated our experiments:

- *Does* GoDeL*'s performance improve as more planning knowledge is given to it?*
- *Does* GoDeL*'s technique of inferring subgoals help discover useful intermediate goals which the domain knowledge can help solve?*
- *How well does* GoDeL *perform when given a complete set of methods?*

Our benchmark planning domains, and our reasons for choosing them, were as follows:

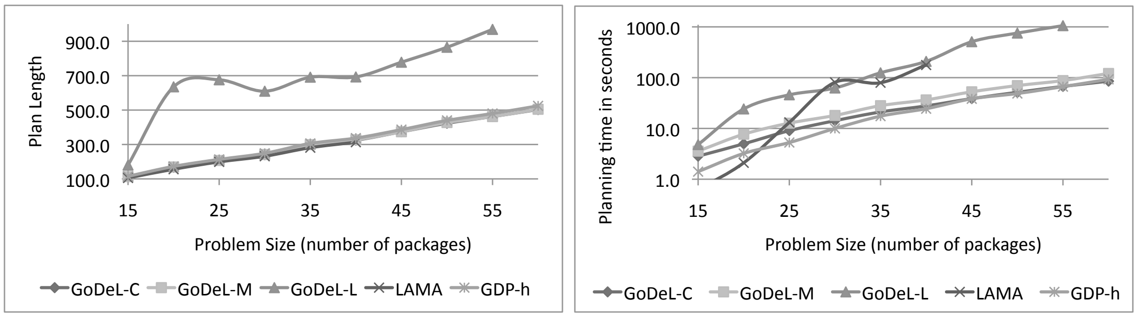
- Logistics [34]. Problem decomposition works well for these problems: they decompose neatly into (1) moving packages with the city, (2) moving packages between airports and (3) moving packages across cities.
- Blocks-World [35]. Blocks-World problems, unlike Logistics problems, do not decompose neatly. Instead, a good planning strategy needs to select actions to apply on a state-by-state basis [39, Sect. 4.5].
- Depots [36]. This domain combines aspects of the above two domains, hence is useful for evaluating the performance of planners with partial planning knowledge.

For each of these domains, we wrote three different method sets having complete (C), moderate (M) and low (L) amounts of domain knowledge respectively, as described below. We shall henceforth refer to GoDeL when using these various levels of knowledge as GoDeL-C, GoDeL-M and GoDeL-L respectively. We compared these variants of GoDeL against GDP-h [6] and LAMA [12], which are state-of-the-art hierarchical and domain-independent planners, respectively. We provided GDP-h the same methods given to GoDeL-C in each domain.

All experiments were run on 2GHz machines with 4GB RAM. We set a time limit of 30 minutes per problem, discarding data points not solved within this time.

5.4.2 Experimental Results

Logistics. For the Logistics domain, the complete method set required three methods: one each for (1) same-city delivery, (2) different-city delivery and (3) airport-to-airport delivery. We constructed the moderate (M) method set by removing method 2. The low (L) method set consisted of just method 1. We compared the planners across 10 n-package problems for each of $n = 15, 20, \ldots, 60$, from the 1998/2000 International Planning Competition distributions.



(a) (b)

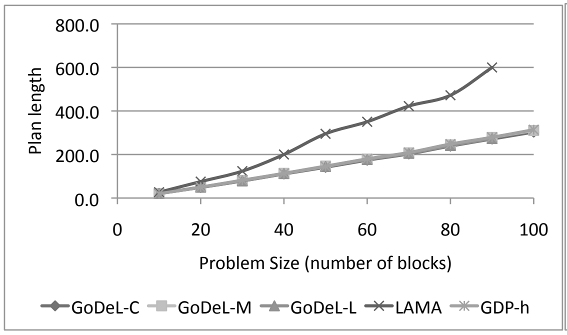
Figure 5.1: Average plan lengths and running times (in logscale) in the Logistics domain, as a function of the number of packages to be delivered. Each data point is the average across 10 problems. GoDeL-L could not solve 60-package problems while LAMA could not solve problems greater than 40.

Figure 5.1 compares the plan lengths and running times of the various planners in this domain. We see that GoDeL-C and GDP-h produced plans of similar length. GoDeL-M was also able to perform at par with these planners; this was because the subgoal inference algorithm automatically inserted the goals that method 2 (which is missing for GoDeL-M) would otherwise have inserted into the goal network. With regard to planning times for these planners, GoDeL-C, GoDeL-M and GDP-h all reported similar running times.

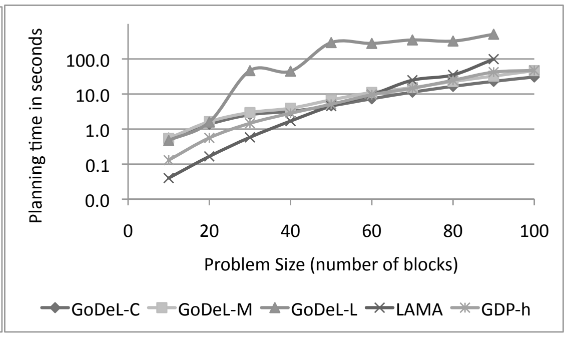
GoDeL-L, however, did not perform as well with its running times nearly an order of magnitude higher and its plan lengths nearly twice as compared to GoDeL-C. This is because it had to solve a larger portion of the problem via action-chaining due to the missing methods. Since this part of GoDeL is relatively simplistic (depth-first search with FF heuristic), GoDeL in a number of cases found it hard to recover from a bad action it chose to apply in the previous step, thus leading to longer plans and running times.

LAMA solved the smaller logistics problems (up to 40-package problems) very quickly, producing plans of roughly the same length as GDP-h, GoDeL-C and GoDeL-M; but it couldn't solve the larger problems within the time limit.

Blocks World. For the Blocks World domain, the complete set of methods included three methods: one each for (1) $\mathsf{on}(X, Y)$, (2) $\mathsf{clear}(X)$ and (3) $\mathsf{on\text{-}table}(X)$. The moderate (M) method set consisted of methods 1 and 2 while the low (l) method set consisted of just method 1. In addition, all three method sets used a $\mathsf{need\text{-}to\text{-}move}(X)$ derived predicate that determined whether X was in its final position. We compared the planners across 25 randomly generated n-blocks problems for each of $n = 10, 20, \ldots, 100$.



(a)



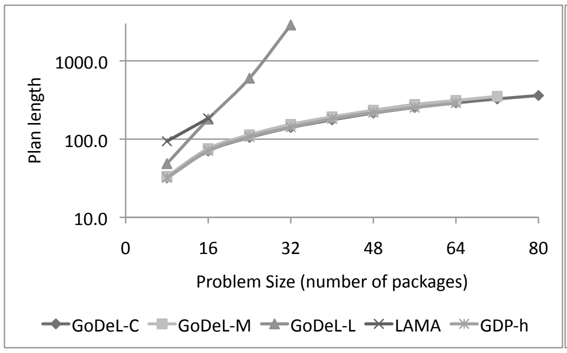
(b)

Figure 5.2: Average plan lengths and running times (in logscale) in the Blocks domain, as a function of the number of blocks. Each data point is the average of 25 randomly generated problems. GoDeL-L and LAMA could not solve 100-block problems.

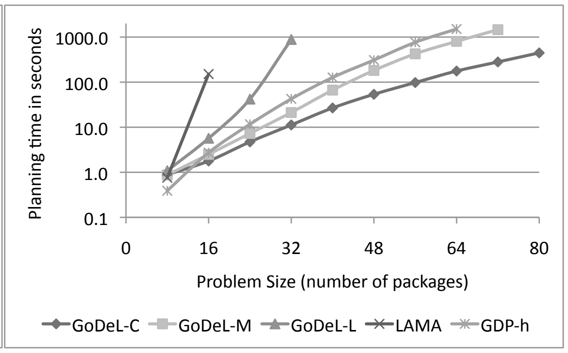
As shown in Figure 5.2, GoDeL-C, GoDeL-M and GDP-h have almost identical planning times. GoDeL-M is able to perform as well as the other two planners because on-table(X) (which the missing method 3 solves) is easily solvable through action chaining. GoDeL-L on the other hand takes significantly longer to produce the same plans. As with Logistics, this is because GoDeL-L has to generate a larger fraction of the plan via action-chaining, thus slowing the planner down. LAMA solves the smaller problems very quickly, but slows down on the larger problems, solving problems larger than 60 blocks slower than GoDeL-M.

With respect to plan length, all the planners produce plans of similar length with the exception of LAMA, which generates much longer plans than the others.

Depots. The complete method set for Depots consisted of five methods and the same derived predicate used in Blocks World. The moderate (M) method set consisted of the same set of methods, but with all knowledge related to the Logistics subproblem



(a)



(b)

Figure 5.3: Average plan lengths and running times (both in logscale) in the Depots domain, as a function of the number of packages to be delivered. Each data point is an average of 25 randomly generated problems. GoDeL-M could not solve 80-package problems; GDP-h, GoDeL-L and LAMA could not solve problems of size > 64, 32 and 16 respectively.

removed. The low (L) method set is identical to the low (L) method set of Blocks World, consisting of just a single method achieving $\mathsf{on}(X)$. We compared the planners on 25 n-package problems for each $n = 8, 16, \ldots, 80$.

As shown in Figure 5.3, GoDeL-C has the best running times, and is the only planner to solve all of the problems. Even GoDeL-M significantly outperformed GDP-h, which had access to the full method set. GoDeL-L, which was given only one Blocks-World method, solved significantly fewer problems. LAMA solved the fewest problems, having not solved any problems containing 24 blocks or more.

With respect to plan lengths, GoDeL-C, GoDeL-M and GDP-h produced plans of similar lengths for the problems they could solve. GoDeL-L and LAMA however generate significantly longer plans for the problems they could solve.

Table 5.1: Amount of action chaining as a function of the domain and the amount of domain-specific knowledge provided. The lower the fraction, the larger the fraction of the plan that is generated via method decomposition.

Domain	GoDeL-C	GoDeL-M	GoDeL-L
Logistics	0	0	0.66
Blocks World	0.09	0.48	0.59
Depots	0.08	0.59	0.75

5.4.3 Discussion

The main conclusions from our experimental study were:

- GoDeL *performed at par or even better than GDP-h when given complete domain knowlege.*

- GoDeL*'s technique of using subgoal inference and action-chaining in tandem helped it to cope with incomplete domain knowledge.* Table 5.1 gives the average fraction of action-chaining (as opposed to method decomposition or relevant-action application) used in generating plans as a function of the domain and amount of domain knowledge given to GoDeL. We can see that as the amount of domain knowledge is reduced, the fractions gradually increase, indicating that GoDeL is able to use whatever little knowledge is given to it. The fraction for GoDeL-M in Logistics is particularly striking; even with one of the methods removed, the plans were generated without resorting to any action chaining. This was because GoDeL was able to automatically infer the subgoals of the missing method and insert them into the goal network using its subgoal inference algorithm.

- GoDeL*'s performance improves as more planning knowledge is given to it.* This is evident from the fact that GoDeL-L (GoDeL with low knowledge) was outperformed by GoDeL-M (GoDeL with moderate knowledge), which in turn was outperformed by GoDeL-C (GoDeL with complete knowledge) across all three domains under consideration.

5.4.4 Related Work

GoDeL's goal semantics differs from the *goal task* semantics in [23]: for a task t and goal g, Erol's semantics can't invoke arbitrary methods that achieve g but aren't labeled with t. It also differs from the semantics for tasks in [26], which requires manually annotating tasks with preconditions and effects, and making special designations to indicate which conditions should be achieved via decomposition and how to achieve them.

[40] propose the use of *articulation functions* to translate literals in a higher level of abstraction to a lower level and use partial-order planning algorithms to resolve these open subgoals at each level. [41] also propose a similar framework; they annotate abstract tasks with preconditions and effects and then use similar partial-order planning algorithms to refine these tasks and generate primitive plans. Our approach differs from these both in the kinds of solutions we consider (extension of classical planning) as well as the techniques we use (forward state-space heuristic search; use of landmarks).

Duet [27] combined the SHOP2 HTN planner with the domain-independent LPG planner [42] to solve planning problems with partial knowledge. Analogously to [26]'s formalism, it required annotations on non-primitive tasks to indicate whether tasks should be solved by SHOP2 or by LPG.

[28] provided a technique for automatically translating a restricted form of HTN methods into PDDL, so that classical planners can be used to solve HTN planning problems. This translation technique, can be used (subject to some restrictions) with partial sets of HTN methods, but is correct only in domains where the recursion depth is bounded, and this bound needs to be supplied manually.

Another related work is that of [43] who propose a landmark-based heuristic for HTN planning. This is in contrast to our work, in which we use landmarks to generate part of the hierarchy.

5.5 Summary

In the modified HGN formalism, methods provide suggestions for ways to accomplish classical goals (rather than impose requirements for how to accomplish tasks, as in HTN planning formalism in [39, Chap. 11]). Thus, GoDeL can work correctly with complete domain knowledge (a full set of methods for every task), no domain knowledge (other than the basic actions, of course), or anything in between. It will terminate with a correct solution if one exists; the amount of knowledge only determines how fast that solution is found.

If GoDeL is given methods only for hard subproblems, its subgoal inference algorithm can automatically figure out subgoals for which those methods are relevant. When given complete domain knowledge, GoDeL performs at par or better than GDP-h [6]. Thus, we believe that GoDeL naturally supports ‘incremental’ domain authoring; users can build partial domain models, evaluate the planner’s performance with that, and then incrementally improve the model if required.

Chapter 6: Towards Integrating Hierarchical Goal Networks and Motion Planners to Support Planning for Human-Robot Teams

6.1 Introduction

Low-level motion planning techniques must be combined with high-level task planning formalisms in order to generate realistic plans that can be carried out by humans and robots. A representative application scenario is planning for fenceless assembly cells where robots can collaborate seamlessly with humans to perform assembly tasks. The advent of safer robots like Baxter and exteroceptive sensing based safety systems [44] are making this human robot collaboration (HRC) paradigm feasible. Some of the components required to achieve safe and efficient HRC include assembly sequence generation [45], task decomposition between human and robot [46], system state monitoring (tracking human, robot, and assembly parts) [47], automated instruction generation for human operations [48], ensuring human safety [44], and recovering from assembly errors [47]. In order to enable a coherent integration among these modules, a high-level planner, interleaved with low-level motion planners, is needed at several levels of the system hierarchy.

For example, given a CAD model of a product to be assembled, RRT based motion planning methods can be used to generate improved assembly precedence constraints [45]. In turn, these constraints can be compiled into a high-level planning problem. Humans and robots share complimentary strengths in performing assembly operations. Therefore, the planning framework must have the ability to incorporate this knowledge in order to decompose the tasks effectively. Further, an integral planner must be able to perform plan-repair in order to handle contingencies during assembly operations. The deviations causing these contingencies may be of two types: (1) low-level deviations in the geometric state without affecting the corresponding symbolic state (e.g., human places part in a wrong posture), which can be corrected at the motion planning level, or (2) deviations in the symbolic state itself (e.g., human picks an incorrect part, and an improved alternative assembly sequence may or may not exist), which needs to be corrected at both levels of planning.

Task planning formalisms typically used to achieve this integration are *Classical Planning* ([49–52]) and *Hierarchical Task Network (HTN) Planning* ([53–55]). Classical planning is not scalable enough to handle complex problems since it can't leverage domain-specific knowledge, and HTNs impose stringent completeness requirements on domain models, which are difficult to guarantee in open, dynamic environments. Therefore, solutions attempting to integrate these AI planning techniques into domains involving real robots and humans are not very satisfactory.

In Chapter 3, we developed a new hierarchical planning formalism called *Hierarchical Goal Networks* (HGNs) that combines aspects of HTN and classical planning into a single framework. This is a hierarchical planning formalism similar to HTN planning, but operates over hierarchies of classical goals instead of tasks. It thus can combine the scalability and expressivity advantages of HTN planning with the heuristic-search/reasoning capabilities of Classical planning.

This chapter now attempts to apply some of these task-level planning techniques to Robotics domains; in particular, integrating HGNs with motion planners to develop planning algorithms that can generate plans directly executable by robots. The aims of this work are twofold:

1. Design a general-purpose planning-and-execution framework that combines HGN planning and execution-time plan-repair algorithms with off-the-shelf motion planners

2. Formulate this planning framework in the context of planning for human-robot teams in assembly cells.

The novelty in this approach comes from the interaction between the heuristic search techniques of the HGN and motion planners, and the particular way in which they guide each other towards high-quality solutions. The HGN planner, when invoking the motion planner, also passes to it an upper bound $\tau\texttt{cost}(a)$, where τ is a user-specified tolerance parameter. The motion planner, being a heuristic search planner, can detect when the lower bound on the best possible solution it can generate exceeds this bound, and can return failure at that point. This is especially useful in cases when a bad goal configuration has been sampled, leading to unsolvable problems. Similarly, motion plan costs are also propagated to the task-planning level to update the heuristic estimates of the symbolic actions, and used to determine whether other actions might lead to better plan costs.

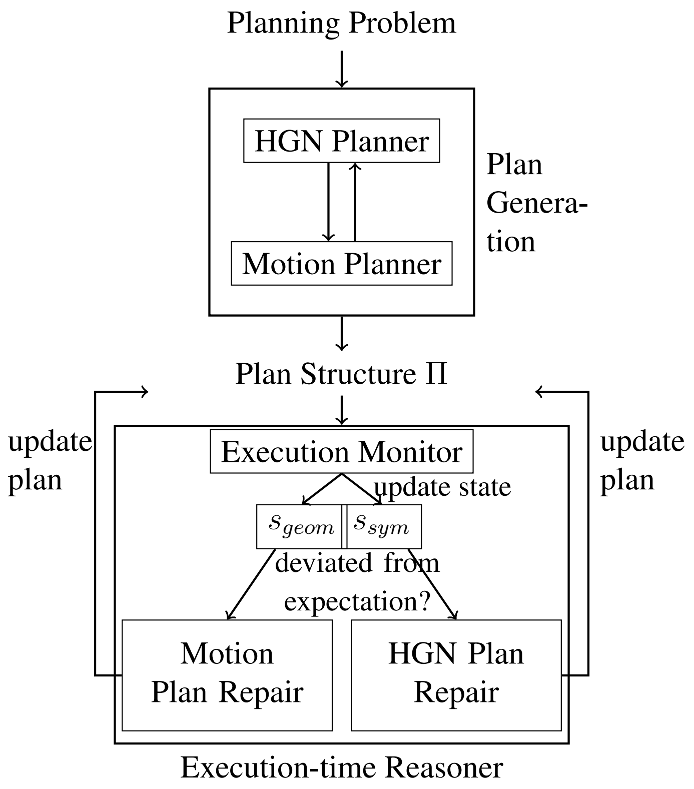


Figure 6.1: System Architecture

The rest of this chapter is structured as follows. Section 6.2 describes the architecture of the overall planning-and-execution system we envision building in this line of work. Formalisms and algorithms related to the *Plan Generation* component of the architecture, which is the main focus of this chapter, is described in sections 6.3 and 6.4. Section 6.5 then discusses the relevant literature. Finally, section 6.6 describes ways by which the techniques developed can be applied to the *Automated Manufacturing* domain.

6.2 System Architecture

Figure 6.1 depicts our proposed architecture of the hybrid planning-and-execution system. This is similar to the conceptual model of planning and execution proposed by

Nau [56], but specifically instantiated for planning domains of interest in this paper, with motion planning algorithms, HGN planning and plan-repair algorithms.

The system takes as input the planning problem P, which provides descriptions of the (1) initial state, (2) goals to be achieved, (3) base action models at the task-planning level, (4) control primitives at the motion planning level, and finally (5) procedures to translate between symbolic and geometric state descriptions. Section 6.3 describes this in more detail.

P is first input into the *Plan Generation* module. In this module, HGN planners and low-level motion planners interactively synthesize an executable plan structure Π[1] that achieves the given goals when applied from the initial state of the system. See section 6.4 for more details on this module.

The plan Π is then input into the *Execution-time Reasoner* module to (1) monitor plan execution, and (2) repair Π in case the deviations of the system state from expectation render the current plan inexecutable. These deviations can be of two types: (1) low-level deviations in the geometric state s_{geom} without affecting the corresponding symbolic state s_{sym} (e.g., human places part in a wrong posture) which can be corrected at the motion planning level, or (2) deviations in the symbolic state itself (e.g., human picks an incorrect part, and an improved alternative assembly sequence may or may not exist) which needs

[1]The uppercase Π is to differentiate it from the lowercase π, which in AI planning research typically denotes a sequence of abstract actions that achieves the goal. Π on the other hand is a multi-layered plan, consisting of HGN methods and actions in the upper layers, and the corresponding low-level plans they refine to (i.e. motor commands that can be understood by the execution platform on the robot) at the lower layers.

to be corrected at both levels of planning. In both cases, Π is then updated with the new plan and execution is resumed. We won't discuss these considerations in this chapter; these are instead discussed in the following chapter in Section 7.4.

6.3 Planning Formalism

When developing the integrated task and motion planning formalism, we need two components: (1) the task-level planning formalism, which in our case is going to be HGN planning, and (2) the motion planning formalism. We shall first summarize a version of the HGN planning formalism, adapted slightly to work with Robotics domains. In particular, we shall represent states using collections of *state variables* instead of *grounded atoms* in predicate logic. We shall then proceed to present the latter, as well as the integrated planning formalism below.

6.3.1 Hierarchical Goal Network Planning

Task Planning Domain. We define the task planning model M_{TP} as a five-tuple $(V_D, V_C, \mathcal{O}, \mathcal{M}, \gamma)$. V_D is the set of discrete state variables in the domain; they evaluate to either `true/false` (e.g. `empty(arm1)=true`) or to a discrete object in the domain (e.g. `pos(robot1)=loc1`). V_C on the other hand represents the set of continuous state variables in the domain, which can evaluate to a real number (e.g. `cpos(robot1,joint3)=4.78`).We refer to assignments to variables in V_D and V_C as symbolic and geometric states; a system state s is a pair (s_{sym}, s_{geom}).

$\mathcal{O}$ represents the set of primitive operator models in the domain, which are model actions that are executable in a single step at the task planning level. These are identical to operators in Classical Planning (ref. Chapter 3), with one exception: each $o \in \mathcal{O}$ also has an associated cost function that can evaluate to a non-negative value.

$\mathcal{M}$ represents the set of HGN methods, which models domain-specific knowledge that suggests ways to decompose goals into subgoals. This is identical to HGN methods defined in Chapter 3, and hence we shall elaborate any further here.

Finally, γ represents the state transition function, identical to the one defined in Chapter 3.

Task Planning Problems. A *HGN planning problem* is a triple $P = (M_{TP}, s_0, gn)$, where M_{TP} is a task planning model, s_0 is the initial state, and $gn = (T, \prec)$ is a goal network.

This formalism follows the same definition of solutions that is used by GoDeL (Chapter 5, Definition 2); this is to exploit GoDeL's advantages and plan robustly with partial domain-specific advice.

6.3.2 Motion Planning

Let $\chi = \mathbb{R}^d$, $d \leq |V_C|$ be the configuration space of the system. Let χ_{obs} be the obstacle region; thus $\chi_{free} = \chi \setminus \chi_{obs}$ represents the obstacle-free space. A motion planning problem is a triple $P = (\chi_{free}, x_0, \chi_{goal})$ where x_0 is an element of χ_{free} and the goal χ_{goal} is a subset of χ_{free}.

A path $\sigma : [0, 1] \to \mathbb{R}^d$ is a valid solution to P if (1) it is *continuous*, (2) it is *collision-free*, i.e. $\sigma(\tau) \in \chi_{free}$ for all $\tau \in [0, 1]$, and (3) the boundary conditions are satisfied, i.e. $\sigma(0) = x_0$ and $\sigma(1) \in \chi_{goal}$.

6.3.3 Connecting Task and Motion Planning Levels

HGN planning, being a task planning formalism, is primarily concerned with search for a path through the symbolic state space induced by the discrete state variables V_D, while motion planners search for paths through the geometric state space induced by the continuous variables in V_C. In order to connect these two levels of planning, we must first provide a way to switch between these two different state spaces: i.e. we must provide a way to (1) generate candidate geometric states consistent with a symbolic state, and conversely (2) generate the symbolic state corresponding to a given geometric state.

We assume the provision of the following domain-specific procedures as part of the domain description:

- Gen_{sym}which takes as input a geometric state s_{geom} and generates the corresponding symbolic state s_{sym}
- Gen_{geom}which takes as input a symbolic state s_{sym} and generates a candidate geometric state s_{geom}.

There is a lot of flexibility in the definition of Gen_{sym} and Gen_{geom}. For instance, Gen_{sym} could be a classifier that is learnt during the training stage [52] or a rule-based generator. Similarly, Gen_{geom} could be domain-specific *geometric suggesters* [53, 54] or a simple sampling-based generator [55].

Thus, the overall planning problem P is a 3-tuple $(\langle M_{TP}, \chi_{free}, \mathsf{Gen}_{sym}, \mathsf{Gen}_{geom}\rangle, s_0, gn)$ where s_0 and gn are the initial state and the goal network respectively.

Definition 3. *We can now provide the overall definition of solutions for a planning problem P as follows. Let π_{sym} be a solution of the underlying HGN planning problem $P_{sym} = (M_{TP}, s_0, gn)$.*

Case 1 . *If π_{sym} is empty, then $\Pi = \langle\rangle$ is a solution for P.*

Case 2 . *Let $\pi_{sym} = a \circ \pi'_{sym}$. Furthermore, let s_1^{sym} be the symbolic state after a is executed. Let s_0^{geom} be the projection of s_0 onto the variables in V_C, and s_1^{geom} be the geometric state generated by Gen_{geom} for s_1. If there exists a valid solution σ to the motion planning problem $(\chi_{free}, s_0^{geom}, s_1^{geom})$ and Π' is a solution to $P' = (\langle M_{TP}, \mathsf{Gen}_{sym}, \mathsf{Gen}_{geom}\rangle, \langle s_1^{sym}, s_1^{geom}\rangle, gn)$, then $\Pi = \sigma \circ \Pi'$ is a solution to P.*

6.4 The CHaMP Algorithm

Algorithm 4: A nondeterministic version of CHaMP. Initially, (D, s, G) is the planning problem, and π is $\langle\rangle$.

```
1 Procedure CHaMP($D, s, G, \pi$)
2 if $G$ is empty then return $\pi$
3 nondeterministically choose a goal formula $g$ in $G$ without any predecessors
4 if $s \models g$ then
5     return CHaMP($D, s, G - \{g\}, \pi$)
6 $\mathcal{U} \leftarrow \{(u, 0) : u$ is an operator/method instance applicable to $s$ and relevant to
  $g\}$
7 sort $\mathcal{U}$ in ascending order of $h_{HGN}(s, u, G)$
```

Algorithm 5: continued from Algorithm 4

```
8  while $\mathcal{U}$ is not empty do
9      remove $(u, n)$ with min $h$ value $(= h_u)$ from $\mathcal{U}$
10     if $u$ is an action then
11         for i=n to NumSamples do
12             $\sigma_{s,u} \leftarrow$ RefineAction$(D, s, u)$
13             if $\sigma_{s,u} \neq$ failure then
14                 $h_{new} = |\sigma_{s,u}| - \texttt{cost}(u) + h_u$ insert $(\langle u, \sigma_{s,u}\rangle, -1)$ into $\mathcal{U}$ with
                   $h = h_{new}$
15                 if $i <$ NumSamples then insert $(u, i)$ into $\mathcal{U}$, updating its
                   heuristic value using $h_{new}$
16                 break
17
18
19     else if $(u, \sigma_{s,u})$ is action, mot. plan pair then
20         res1 $\leftarrow$ CHaMP $(D,$ UpdateState$(s', \sigma_{s,u}), G, \pi \circ \langle u, \sigma_{s,s'}\rangle)$
21         if res1 $\neq$ failure then return res1
22     else
23         res1 $\leftarrow$ CHaMP$(D, s,$ subgoals$(u) \circ G, \pi)$
24      if res1 $\neq$ failure then  return res1
25 return failure
```

Algorithm 4 is the Combined HGN and Motion Planning(CHaMP) planning algorithm, an integrated HGN and motion planning algorithm, which combines elements of GoDeL[2] (see Section 5), modified to integrate motion planning into the plan generation.

6.4.1 Task Planning in CHaMP

Lines 2 – 5 specify the base cases of CHaMP. If these are not satisfied, the algorithm nondeterministically chooses a goal g with no predecessors and generates $\mathcal{U}$, all method and operator instances applicable in s and relevant to g. It then chooses a $u \in \mathcal{U}$ with the lowest heuristic value to progress the search (Lines 6 – 9). If u is an action, the state is progressed to $\gamma(s, u)$ (Line 10) and u is further refined into an executable motion plan (Lines 12–24). Section 6.4.2 talks about this in more detail. Else, the subgoals of u are added to G, adding edges to preserve the total order imposed on subgoals(u) (Line 22). In either case, CHaMP is invoked recursively on the new planning problem (Lines 20 and 24).

6.4.2 Motion Planning in CHaMP

Motion planners are used to refine actions at the task-planning level into executable motion plans. CHaMP can use any off-the-shelf motion planner; the only requirement is that the motion planner should be able to provide an *anytime lower bound* during the search. In other words, the motion planner at any time should be able to provide a lower

[2]Simplified for ease of presentation; this version doesn't include the auxiliary planning techniques needed to work with incomplete/incorrect methods, as these aspects are orthogonal to the focus of this chapter.

Algorithm 6: Procedure to refine an action at the task-planning level into an executable motion plan. It takes as input the action u and the current state s, and outputs either a valid motion plan $\sigma_{s,u}$ or failure. It also takes as input two user-specified parameters, τ and NumSamples.

```
1 Procedure RefineAction$(D, s, u)$
2 begin
3     $s' \leftarrow \gamma(s, u)$
4     $x_{s'} \leftarrow \mathsf{Gen}_{geom}(s, s'_{sym})$
5     $\sigma_{s,u} \leftarrow \mathsf{MotionPlanner}(\chi_{free}, s_{geom}, x_{s'}, \tau \times \mathtt{cost}(u))$
6     return $\sigma_{s,u}$
7 end
```

bound on the best solution its going to find henceforth. Therefore heuristic motion planners such as ARA* [57], R* [58], which do (weighted) A* search are ideally suited for use here.

The motion planning in CHaMP is parametrized by two extra user-specified constants, τ and NumSamples. The model of any action u at the task-planning level has a cost function which provides a rough estimate of how much any refinement of u would cost. τ is an upper bound on the number of times by which the motion plan cost is allowed to overshoot $\mathtt{cost}(u)$. NumSamples is the maximum number of geometric configurations the motion planner is invoked on for a given symbolic state before backtracking.

When it reaches an action u, CHaMP invokes RefineAction to generate a motion plan that refines u (Line 12). RefineAction picks a geometric configuration consistent with

u's effects, and passes it to MotionPlanner along with the upper bound of $\tau \times \texttt{cost}(u)$. MotionPlanner thus either returns a plan within this bound or returns failure. This feature in particular helps in cases when no motion plan exists, which RefineAction can now automatically detect and stop the motion planner.

When CHaMP receives the motion plan $\sigma_{s,u}$, it inserts it into $\mathcal{U}$ with heuristic value of $\sigma_{s,u}$ (Line 14). It also reinserts the original action u annotated with the number of samples remaining with the heuristic value updated with $\sigma_{s,u}$ (Line 15), as this is a more accurate estimate of the cost of u.

The purpose of this reinsertion is the following: suppose $\texttt{cost}(u)$ was overly optimistic and the real estimate of u, $|\sigma_{s,u}|$, is significantly larger, we would like to give other actions a chance to refine to cheaper motion plans. If $\sigma_{s,u}$ is indeed the best option, it would be removed from $\mathcal{U}$ in the next iteration, and CHaMP will be recursively called in the state resulting from it (Line 20). In this way, we are tightly integrating task and motion planning search spaces by using results from one to help direct the search in the other.

This process is repeated until either each action has been refined NumSamples times and failed, in which case the planner backtracks, or a recursive call on the state resulting from a particular motion plan generates a valid plan, in which case CHaMP succeeds and returns the generated plan.

6.5 Related Work

Most of the related work on integration of task and motion planning techniques either use *Classical Planning* or *Hierarchical Planning* techniques.

Classical Planning & Robotics. The seminal work in integrating Classical Planning and Motion Planning is by Cambon et al [49]. They use the FF planner as a heuristic estimator to guide the search of the motion planner. Erdem et al [50] integrate task and motion planning by extending the $\mathcal{C}+$ action description language to allow incorporation of external predicates which allow interfacing with external programs; thus one could run arbitrary domain-specific geometric reasoners and motion planners as part of precondition checks of primitive actions. Similarly, Dornhege et al [51] propose a similar framework to specify 'semantic attachments' as part of PDDL domain descriptions and extend the FF planner to generate plans for such problems. Burbridge et al [52] suggest a novel machine learning approach to interface between the discrete and continuous state spaces, and use this to integrate motion planners with classical planners. Our approach differs from these by the use of hierarchical planning techniques which allow increased expressivity and scalability, and by the particular ways in which flow of heuristic estimates from the task planner to the motion planner and vice-versa allow for more robust search and explicit plan cost optimization.

Hierarchical Planning & Robotics. Wolfe et al [55] combine HTN planners with motion planners by discretizing the choice of geometric states via finite sampling during refinement of a symbolic action. Another line of work [53, 54] builds a special-purpose regression-based hierarchical planner that they combine with motion planners. Our approach differs from these techniques by the use of HGN planners, which allow for use of heuristics during hierarchical decomposition, and as explained in the previous paragraph, by the novel way in which heuristic estimates from each level of planning guide the other.

6.6 Planning for Assembly Cells: A Potential Application of CHaMP

One of our aims is to ground our proposed architecture (described in Fig. 6.1) in a manufacturing domain. Figure 6.2 shows a snapshot of such a system. The shop floor is divided into four regions: (1) Part storage, (2) Tool storage, (3) Subassemblies building, and (4) Final assembly.

The proposed system takes a CAD model of the product that needs to be assembled as input and performs the following:

- **Assembly Precedence Constraints Generation**: automatically generate assembly precedence constraints and detect useful subassemblies using techniques in [45] and compile this into an HGN planning problem.

- **Plan Generation**: Use the integrated HGN and motion planning algorithm to generate an executable plan structure Π – actions to be performed by humans can be provided at the symbolic level, while those performed by robots need to be refined into motion plans. Factors that determine whether a particular task needs to be executed by a human or robot can be modeled in the action and method preconditions; the planner, based on the particulars of a given task, can use these conditions to generate compatible assignments. Higher-level protocols that need to be followed (such as reserving a tool before use) can be modeled as HGN methods.

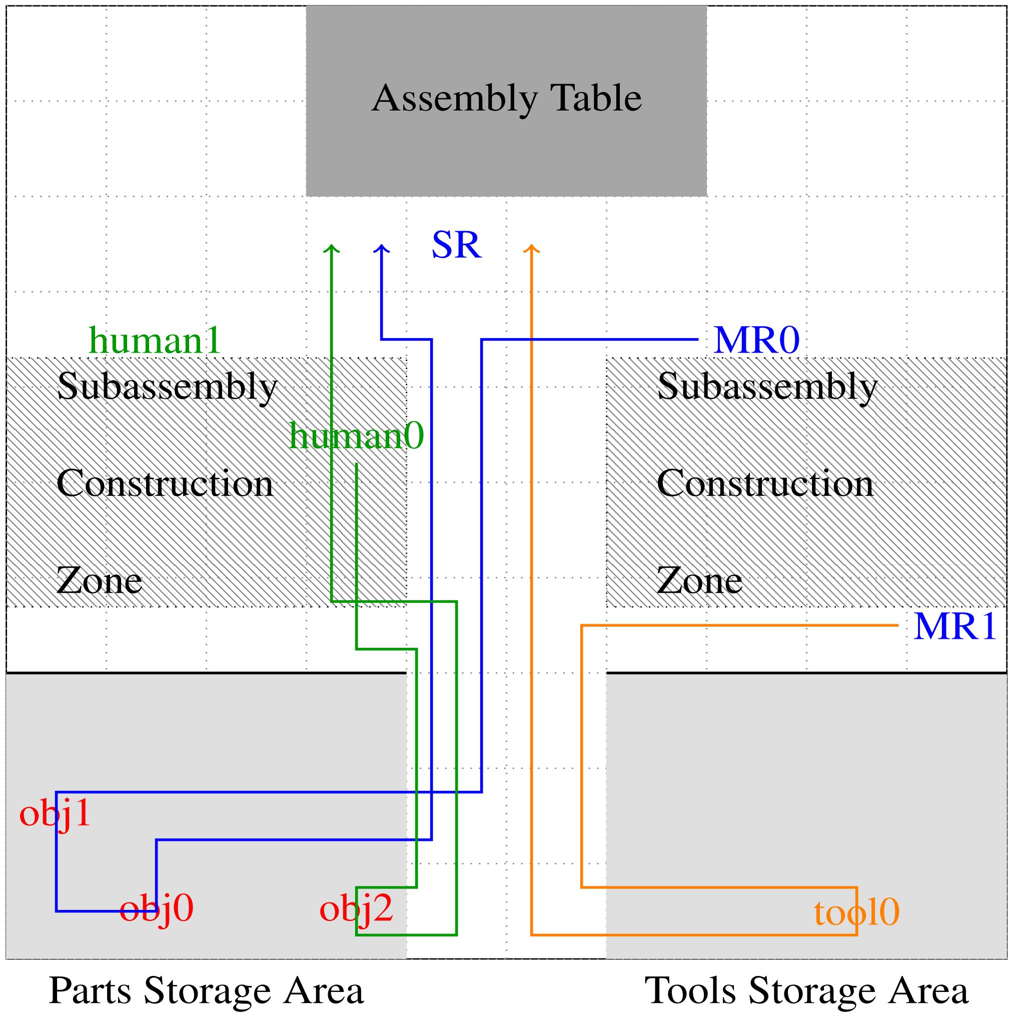


Figure 6.2: Sample assembly planning problem:`MR0` and `MR1` are mobile robots, `human0` and `human1` are humans, and `SR` is a static robot. The goal is to assemble `obj0`, `obj1` and `obj2` together using `tool0`. One of the solutions induced by the task planning model is also shown.

- **Execution-time Reasoning**: Execute Π, while continuously monitoring the geometric state of the system using sensors, both vision-based and otherwise, that monitor locations of workers, tools and parts, as well as other information such as battery levels of robots, stress in robot arms while carrying heavy loads, etc. The system will repair Π using HGN and motion plan repair algorithms, when deviations from both the expected symbolic and geometric states are observed.

  As mentioned previously, the main focus of this chapter is on the *Plan Generation* module in the overall architecture, and therefore developing these capabilities is future work (more details on how this could be done in Chapter 7).

We model the action models of the domain in **P**lanning **D**omain **D**escription **L**anguage (PDDL), a standard language to encode task-level planning domain descriptions.

The base actions in this domain are as follows:

1. `Worker-Move (workerId start-loc goal-loc)`
2. `Worker-Pickup-Object (workerId objId obj-loc)`
3. `Worker-Putdown-Object (workerId objId goal-loc)`
4. `Worker-Move-Near-Object (workerId current-loc objId)`
5. `Worker-Move-Near-Location (workerId current-loc goal-loc)`
6. `Mark-Worker-Free (workerId)`
7. `Mark-Worker-Busy (workerId)`
8. `Asmbl-Objs (workerId obj1Id obj2Id)`

These action models, in addition to modeling causal preconditions and effects, also model procedural and geometric preconditions over the continuous variables; for instance, the `Worker-Pickup-Object` action checks if the object is graspable by the worker, its weight is within the carrying capacity of the worker, etc. Thus, the planner can flexibly assign humans or robots to execute actions based on their compatibility with the preconditions. If a robot is assigned, only then is the action is further refined using motion planners. If a human is assigned, the system then passes the symbolic action directly for execution; however, since the system has no control over the exact geometric state that results from the action execution, the planner chooses one nondeterministically and relies on the plan-repair system to repair the plan if needed.

6.7 Summary

The ultimate objective of this work is to build a general-purpose planning-and-execution system that enable long-term autonomous behavior by human-robot teams. This chapter develops a possible approach towards realizing one of the components needed to achieve this, a planning system that tightly integrates task and motion planning. The main contribution of the techniques developed here lies in leveraging the advantages of *HGN planning*, which allows combining hierarchical decomposition with heuristic search, and *heuristic search motion planners*, which can provide useful information about the status of their search as feedback to the HGN planner, to achieve this tight integration. The combined planning algorithm allows for using search information from the task planner to guide the search for a high-quality motion plan by the motion planner, as well as using search results from the motion planner to revisit decisions made by the task planner, thus potentially enabling tighter interleaving of the two levels of planning.

Chapter 7: Looking Ahead: Discussion and Open Problems

As mentioned previously, there is a real need for efficient and scalable techniques that can enable long-term autonomous behavior in realistic environments; popular planning techniques based on Classical and HTN Planning have critical limitations that limit their usefulness in these situations. So one of the contributions of this thesis is a new planning formalism and associated algorithms that are better placed to handle the difficulties in these kinds of environments.

Having said that, this is simply a first step; there are several other capabilities that need to be developed to achieve robust autonomy. The following sections discusses some of these research questions, as well as possible approaches one could take to solve them from a HGN planning standpoint.

7.1 Computing Optimal and Near-Optimal Plans

GDP and GoDeL used a primitive version of heuristic search (inadmissible heuristics combined with DFS) which worked well enough. However, there are many domains in which it is critical to come up with a high-quality or even optimal solution. Depth-first search based algorithms in general are not well-suited for these requirements, and therefore we must look to other heuristics and search algorithms; for instance, A^* [59] or depth-first branch-and-bound (DFBB) can explicitly optimize the solution. The key issue then becomes designing good (or in the case of optimal planning, *admissible*) heuristics to guide these search techniques. The good news is that we already have these in HGNs! It is also fairly straightforward to adapt admissible classical planning heuristics (such as the LA heuristic [13] or the LM-cut heuristic [14]) to provide admissible estimates in HGN planning. This is in contrast to HTN planning, where it is unclear how to perform a similar adaptation.

Optimal planning in classical planning literature is typically many orders of magnitude slower than *satisficing* planning, which concentrates on just finding a solution without worrying about its optimality. So, another reason to look for optimal plans in the context of HGN planning is that the extra domain-specific advice that HGN planners can leverage will help improve the efficiency of optimal planning.

The above ideas could be generalized in a couple of ways:

1. Use the weighted version of A* [60], which allows for trading optimality for performance.

2. Develop anytime versions of these algorithms that quickly generate a plan using GDPor GoDeL, and then using these heuristics to iteratively improve the generated plans by exploring parts of the search place thus far unexplored.

7.2 Execution-Time Plan Repair

One of the natural extensions to the offline planning techniques developed in this thesis is to augment them with execution-time plan repair capabilities to handle unexpected exogenous events that break plans during execution. Hierarachical planning approaches are particularly well-suited for this problem since the hierarchical structure generating during the planning process is invaluable localize the faults in the plan effectively as demonstrated in previous HTN plan-repair algorithms [61, 62]. However, there are two critical limitations with these algorithms: (1) just like while planning, the lack of heuristics makes it difficult to find a high-quality plan during the plan-repair process, and (2) due to the brittleness of HTN planners, the amount of knowledge that needs to be provided to handle all possible contingencies during execution is prohibitively expensive. HGN planning techniques can help address both of these limitations. Therefore, developing an execution semantics for HGNs and associated plan-repair techniques can be useful for execution-time deliberation.

7.3 Learning Planning Strategies

Both HTN and HGN planners need sophisticated domain-specific strategies to be provided. GoDeL can operate with partial amounts of planning advice, but at the cost of planning performance. So automatically learning planning strategies from past plan traces would be really useful. It also helps that we do have a HGN planner that can operate with partial amounts of automatically-inferred domain knowledge.

There has already been some work on this in the context of HTN planning [63–65]. However, HGNs have a nice connection with landmarks, therefore there are interesting possibilities in learning HGN methods from landmark graphs computed during the planning phase.

7.4 Combined HGN and Motion Plan Repair Algorithm

The CHaMP algorithm described in Chapter 6, which combines HGN planning techniques with heuristic search motion planning techniques from Robotics, is only responsible for generating executable plans. Additional machinery is required to repair these multi-layered plans during execution. This would involve combining HGN plan repair algorithms with incremental motion plan repair algorithms such as LPA* [66] and D* [67].

A possible way to solve this problem would be to, like CHaMP, maintain two levels of state information, the raw continuous state information required by motion planning and plan-repair algorithms require, and the symbolic state information that HGN planning and plan-repair algorithms need. Minor deviations in the raw state that don't change the corresponding symbolic state during execution (for example, wheel drift of mobile robots) can be accounted for at the motion planning level itself. Larger deviations that result in change in the symbolic state as well (for example, a robot breaking down, a pathway being blocked, etc) require repairing the plan at both levels.

The main challenge, however, would be interleave the two levels of plan-repair effectively (possibly similar to the way CHaMP interleaves the two levels of plan generation) in a way that the process is both tractable as well as generates high-quality plans.

Chapter 8: Conclusion

This thesis makes three important contributions:

- **Hierarchical Goal Networks Planning Formalism**: This formalism combines advantages of Hierarchical Task Network planning and Classical planning, two of the most popular AI planning frameworks. In particular, it allows one to leverage the ability to specify domain-specific problem-solving strategies of HTNs along with the heuristic search and other reasoning techniques from Classical planning.
- GDP **and** GoDeL**- Offline Planning Algorithms**: GDP is similar to state-of-the-art HTN planners like SHOP/SHOP2, but leverages domain-independent heuristics to automatically deduce a good order of method application. GoDeL combines GDP with classical planning techniques to work with arbitrary amounts of planning knowledge , thus being robust to errors and inadequacies in the input domain knowledge. GoDeL can thus perform at par with state-of-the-art classical planners when given only the action models, state-of-the-art HTN planners when given complete planning knowledge, and everything in between. These two algorithms therefore help realize the potential advantages the HGN formalism had to offer.
- **A Combined HGN and Motion Planning Algorithm**: The final piece of this the-

sis describes CHaMP, an integrated planning algorithm that utilizes the advantages of HGN planners and extends their scope to Robotics; in particular, it combines the hierarchical task-level planning techniques developed in this thesis with off-the-shelf motion and object manipulation planners used for low-level planning in Robotics. CHaMP uses the unique properties of heuristic search motion planners to tightly integrate the two planning levels. It achieves this by allowing the HGN planner to monitor the search progress at the lower levels and proactively suspend and switch to another option if needed.

There are a number of theoretical implications of the techniques developed in this thesis. Firstly, the HGN formalism provides stronger correctness guarantees than HTNs. In particular, it is possible to write HTN methods for your planning domain in such a manner so as to allow unsound solutions. HGNs on the other hand do not allow this; plans generated by HGN planners are always sound. Furthermore, the HGN formalism allows for easier incorporation of Classical Planning techniques; for instance, it was possible to adapt Classical Planning heuristics that could guide the planner towards high-quality solutions and other useful reasoning techniques such as landmark generation to plan with only partial amounts of domain-specific knowledge. The latter allowed us to develop planning algorithms that provide stronger completeness guarantees than standard HTN planners.

From a practical standpoint, the HGN planning formalism and algorithms remedies several deficiencies present in hierarchical planning formalisms like HTNs, thus potentially enabling more widespread use of automated planning research in practical applications. Also, the integrated task-level and motion planning techniques proposed in this thesis will help refine otherwise abstract plans that are generated by the HGN planners into more concrete plans that are directly executable by robots.

There is a significant research effort underway in equipping robots to work with minimal human intervention in diverse sectors such as in manufacturing plants, as personal assistants for the elderly in hospitals, and as servants at home. Given the unstructured nature of these environments and the obvious need for high-level deliberation capabilities, we believe the planning-and-acting systems that are developed as part of this thesis could provide unique insights into ways to realize these systems in the real world.

Bibliography

[1] Rob Sherwood, Andrew Mishkin, Tara Estlin, Steve Chien, Paul Backes, Brian Cooper, Scott Maxwell, and Gregg Rabideau. Autonomously generating operations sequences for a mars rover using artifical intelligence-based planning. In *IROS*, October 2001.

[2] Ghallab Malik, S. Nau Dana, and Traverso Paolo. The actor's view of automated planning and acting: A position paper. *Artif. Intell.*, 208:1–17, 2014.

[3] M. Chung, M. Buro, and J. Schaeffer. Monte carlo planning in RTS games. In *IEEE Symp. Comp. Intel. Games*, 2005.

[4] Remco Straatman, Tim Verweij, and Alex Champandard. Killzone 2 multiplayer bots. In *Game AI Conference*, June 2009.

[5] Ugur Kuter, Evren Sirin, Dana S. Nau, Bijan Parsia, and James Hendler. Information gathering during planning for web service composition. *JWS*, 3(2-3):183–205, 2005.

[6] Vikas Shivashankar, Ugur Kuter, Dana S. Nau, and Ronald Alford. A hierarchical goal-based formalism and algorithm for single-agent planning. In *AAMAS*, pages 981–988, 2012.

[7] J. Hoffmann and Bernhard Nebel. The FF planning system. *JAIR*, 14:253–302, 2001.

[8] Vikas Shivashankar, Ugur Kuter, Dana S. Nau, and Ronald Alford. The GoDeL planning system: A more perfect union of domain-independent planning and hierarchical planning. In *IJCAI*, 2013.

[9] A. Gerevini and I. Serina. LPG: a planner based on local search for planning graphs. In *AIPS*, pages 968–973, 2002.

[10] Yixin Chen, Chih wei Hsu, and Benjamin W. Wah. SGPlan: Subgoal partitioning and resolution in planning. In *International Planning Competition, Deterministic Track*, pages 30–32, 2004.

[11] M. Helmert. The Fast Downward planning system. *JAIR*, 26:191–246, 2006.

[12] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)*, 39:127–177, 2010.

[13] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In Craig Boutilier, editor, *IJCAI*, pages 1728–1733, 2009.

[14] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *ICAPS*, 2009.

[15] A. J. Coles, A. I. Coles, M. Fox, and D. Long. Temporal planning in domains with linear processes. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI)*, July 2009.

[16] Andrew Coles, Maria Fox, Keith Halsey, Derek Long, and Amanda Smith. Managing concurrency in temporal planning using planner-scheduler interaction. *Artificial Intelligence*, 173(1):1–44, 2009.

[17] J. Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *JAIR*, 20:291–341, 2003.

[18] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Colin: Planning with continuous linear numeric change. *J. Artif. Intell. Res. (JAIR)*, 44:1–96, 2012.

[19] K. Currie and A. Tate. O-Plan: The open planning architecture. *Artif. Intell.*, 52(1):49–86, 1991.

[20] A. Tate, B. Drabble, and R. Kirby. *O-Plan2: An Architecture for Command, Planning and Control*. 1994.

[21] D. E. Wilkins. Domain-independent planning: Representation and plan generation. *Artif. Intell.*, 22(3):269–301, April 1984.

[22] Kutluhan Erol, James Hendler, and Dana S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.

[23] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, pages 249–254, June 1994. ICAPS 2009 influential paper honorable mention.

[24] Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. The SHOP planning system. *AI Mag.*, 2001.

[25] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, December 2003.

[26] S. Kambhampati, A. Mali, and B. Srivastava. Hybrid planning for partially hierarchical domains. In *AAAI*, pages 882–888, 1998.

[27] Alfonso Gerevini, Ugur Kuter, Dana S. Nau, A. Saetti, and N. Waisbrot. Combining domain-independent planning and HTN planning. In *ECAI*, pages 573–577, July 2008.

[28] Ron Alford, Ugur Kuter, and Dana S. Nau. Translating HTNs to PDDL: A small amount of domain knowledge can go a long way. In *IJCAI*, July 2009.

[29] Michael P. Georgeff and Amy L. Lansky. Reactive reasoning and planning. In *AAAI*, pages 677–682, 1987.

[30] F.F. Ingrand and M.P. Georgeff. An architecture for real-time reasoning and system control. *IEEE Expert*, 6:33–44, 1992.

[31] E. Sacerdoti. The nonlinear nature of plans. In *IJCAI*, pages 206–214, 1975.

[32] Kutluhan Erol, James Hendler, and Dana S. Nau. Semantics for hierarchical task-network planning. Technical Report CS TR-3239, UMIACS TR-94-31, ISR-TR-95-9, University of Maryland, March 1994.

[33] Kutluhan Erol, James Hendler, and Dana S. Nau. Complexity results for hierarchical task-network planning. *AMAI*, 18:69–93, 1996.

[34] Manuela M. Veloso. Learning by analogical reasoning in general problem solving. PhD thesis CMU-CS-92-174, Carnegie Mellon University, 1992.

[35] Fahiem Bacchus. The AIPS '00 planning competition. *AI Mag.*, 22(1):47–56, 2001.

[36] Maria Fox and Derek Long. International planning competition, 2002. http://planning.cis.strath.ac.uk/competition.

[37] Vikas Shivashankar, Ugur Kuter, and Dana Nau. Hierarchical goal network planning: Initial results. Technical Report CS-TR-4983, Univ. of Maryland, May 2011.

[38] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *JAIR*, 22:215–278, 2004.

[39] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. May 2004.

[40] L Castillo, J Fdez-Olivares, and A González. On the adequacy of hierarchical planning characteristics for real-world problem solving. *European conference on planning (ECP-2001)*, pages 169–180, 2001.

[41] S. Biundo and B. Schattenberg. From abstract crisis to concrete relief–a preliminary report on combining state abstraction and HTN planning. In *Proc. of the 6th European Conference on Planning*, pages 157–168, 2001.

[42] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *JAIR*, 20:239–290, 2003.

[43] Mohamed Elkawkagy, Pascal Bercher, Bernd Schattenberg, and Susanne Biundo. Improving hierarchical planning performance by the use of landmarks. In *AAAI*, pages 1763–1769, 2012.

[44] Carlos Morato, Krishnanand N. Kaipa, Boxuan Zhao, and Satyandra K. Gupta. Toward safe human robot collaboration by using multiple kinects based real-time human tracking. *ASME Journal of Computing and Information Science in Engineering*, 14(1):011006, 2014.

[45] Carlos Morato, Krishnanand N. Kaipa, and Satyandra K. Gupta. Improving assembly precedence constraint generation by utilizing motion planning and part interaction clusters. *Computer-Aided Design*, 45(11):1349 – 1364, 2013.

[46] Krishnanand N. Kaipa, Carlos Morato, Jiashun Liu, and Satyandra K. Gupta. Human-robot collaboration for bin-picking tasks to support low-volume assemblies. In *Human-Robot Collaboration for Industrial Manufacturing Workshop, held at Robotics: Science and Systems Conference (RSS 2014)*, 2014.

[47] Carlos Morato, Krishnanand N. Kaipa, Jiashun Liu, and Satyandra K. Gupta. A framework for hybrid cells that support safe and efficient human-robot collaboration in assembly operations. In *ASME Computers and Information Engineering Conference*, 2014.

[48] Krishnanand N. Kaipa, Carlos Morato, Boxuan Zhao, and Satyandra K. Gupta. Instruction generation for assembly operations performed by humans. In *ASME Computers and Information Engineering Conference*, 2012.

[49] Stéphane Cambon, Rachid Alami, and Fabien Gravot. A hybrid approach to intricate motion, manipulation and task planning. *I. J. Robotic Res.*, 28(1):104–126, 2009.

[50] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4575–4581, May 2011.

[51] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. In *ICAPS*, 2009.

[52] Chris Burbridge and Richard Dearden. An approach to efficient planning for robotic manipulation tasks. In *ICAPS*, 2013.

[53] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *ICRA*, pages 1470–1477, 2011.

[54] Dylan Hadfield-Menell, Leslie Pack Kaelbling, and Tomas Lozano-Perez. Optimization in the now: Dynamic peephole optimization for hierarchical planning. In *ICRA*, 2013.

[55] Jason Wolfe, Bhaskara Marthi, and Stuart J. Russell. Combined task and motion planning for mobile manipulation. In *ICAPS*, pages 254–258, 2010.

[56] Dana S. Nau. Current trends in automated planning. *AI Magazine*, 28(4):43–58, 2007.

[57] Maxim Likhachev, Dave Ferguson, Geoffrey J. Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artif. Intell.*, 2008.

[58] Maxim Likhachev and Anthony Stentz. R* search. In *AAAI*, pages 344–350, 2008.

[59] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Sciences and Cybernetics*, pages 1556–1562, 1968.

[60] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *J. ACM*, 32(3):505–536, July 1985.

[61] N. Fazil Ayan, Ugur Kuter, F. Yaman, and Robert Goldman. Hotride: Hierarchical ordered task replanning in dynamic environments. In *Proceedings of the ICAPS-07 Workshop on Planning and Plan Execution for Real-World Systems – Principles and Practices for Planning in Execution.*, 2007.

[62] Julien Bidot, Bernd Schattenberg, and Susanne Biundo. Plan repair in hybrid planning. In *KI 2008: Advances in Artificial Intelligence*, pages 169–176. 2008.

[63] Chad Hogg, Héctor Muñoz-Avila, and Ugur Kuter. HTN-MAKER: Learning HTNs with minimal additional knowledge engineering required. In *AAAI*, pages 950–956, 2008.

[64] Hankz Hankui Zhuo, Derek Hao Hu, Chad Hogg, Qiang Yang, and Hector Munoz-Avila. Learning htn method preconditions and action models from partial observations. In *IJCAI*, 2009.

[65] Chad Hogg, Ugur Kuter, and Hector Muñoz-Avila. Learning methods to generate good plans: Integrating HTN learning and reinforcement learning. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[66] Maxim Likhachev and Sven Koenig. A generalized framework for lifelong planning A* search. In *ICAPS*, pages 99–108, 2005.

[67] Maxim Likhachev, David Ferguson , Geoffrey Gordon, Anthony (Tony) Stentz, and Sebastian Thrun. Anytime dynamic A*: An anytime, replanning algorithm. In *ICAPS*, 2005.