LIMITATIONS OF AND EXTENSIONS TO HEURISTIC SEARCH PLANNING

Daniel Burfoot

School of Computer Science
McGill University, Montréal

August 2006

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Master of Science

ACKNOWLEDGMENTS

I would like to thank several people for helping me with this thesis. Particular credit goes to my supervisors, Gregory Dudek and Joelle Pineau, who supported and encouraged me to pursue my ideas. In addition to being generally great people, it's hard for me to imagine a better pair of academic mentors. I would also like to thank the members of the Mobile Robotics Laboratory for answering my many questions about Linux, C++ programming, and Latex formatting; and also for being a great group of friends and colleagues. This work would not have been possible without the help of several researchers who generously made their software systems available: Jörg Hoffmann, Bernhard Nebel, Malte Helmert, Blai Bonet, Hector Geffner and David Avis. Finally I would like to thank my parents and family for all their support over many years.

ABSTRACT

This thesis explores limitations of heuristic search planning, and presents techniques to overcome those limitations. The two halves of the thesis discuss problems in standard propositional planning (STRIPS) and in planning with numeric state variables respectively.

In the context of STRIPS, the primary focus is on the widely used relaxed plan heuristic (h^+). A variety of cases are shown in which h^+ provides systematically bad estimates of goal distance. To address this breakdown, a planning system called RRT-Plan is presented. This system is inspired by the concept of Rapidly-exploring Random Trees, which was originally developed for use in mobile robot path planning. Experimental results show that RRT-Plan is comparable to leading planners in terms of number of problems solved and plan quality. We conclude that the effectiveness of RRT-Plan is based on its ability to search the space of artificial goal orderings.

The second half of the work considers heuristic search planning in numeric domains. Two particularly significant obstacles are identified. The Curse of Affluence is due to the vast blowup in the search space caused by the addition of numeric variables. The Curse of Poverty relates to the difficulty of finding relevant lower bounds on resource consumption.

Exploration of the Curse of Affluence leads to the new concepts of reduced search and enhanced states. In reduced search, certain simple operators are not used to expand states. Instead, enhanced states are constructed which represent all possible states which could be achieved by suitably inserting simple operators in the plan. Enhanced states are represented by a set of constant discrete variables, and a convex hull of numeric values. This representation can be queried and updated in a natural way. Experimental results show that there are domains for which reduced search gives order of magnitude performance improvements over Metric-FF, a leading heuristic search planner for numeric domains.

RÉSUMÉ

Cette thèse explore les limites des heuristiques de planification de recherche et présente des techniques pour surmonter ces limites. La thèse est divisée en 2 parties. La première traite de problèmes standards de planification propositionnelle et la seconde de planification avec variables numériques d'état.

Dans le contexte de STRIPS, l'attention est portée sur l'heuristique "Relaxed Plan" (h^+) qui est couramment utilisé. Une série d'exemples pour lesquels h^+ donne systématiquement des résultats erronés est présentée. Pour résoudre ce problème, un système de planification appelé RRT-Plan est proposé. Ce système s'inspire du concept d'exploration rapide d'arbres aléatoires qui a d'abord été développé pour la planification de trajectoires de robots mobiles. L'efficacité de RRT-Plan repose sur sa capacité à chercher l'espace d'ordonnancement de buts artificiels. Lorsque le problème est partitionné de cette faon, h^+ donne un meilleur estimé. Les résultats présentés démontrent que RRT-Plan peut être comparé aux techniques de pointe en terme de nombre de problèmes résolus et de qualité des plans obtenus.

La seconde partie de la thèse se penche sur les heuristiques de planification de recherche dans les domaines numériques. La *Malédiction de l'Affluence* est un problème qui résu lte de l'explosion de la taille de l'espace de recherche lors de l'ajout de variables numériques. La *Malédiction de la Pauvreté* est liée à la difficulté de déterminer une limite inférieure à la consommation de ressources.

L'exploration du problème de la *Malédiction de l'Affluence* amène les nouveaux concepts de *recherche réduite* et *d'état enrichi*. La recherche réduite s'effectue en remplaant l'utilisation de certains opérateurs simples dans le développement des états par des états enrichis qui représentent tous les états qui auraient pu être atteints par l'utilisation des opérateurs simples éliminés. Un état enrichi est représenté par un ensemble de constantes discrètes et une enveloppe convexe de valeurs numériques. Cette représentation facilite la requête et la mise à jour. Les résultats expérimentaux démontrent qu'il existe certains domaines pour lesquels la recherche réduite permet d'améliorer la performance d'un ordre de grandeur par rapport à Metric-FF, un heuristique de planification de recherche de pointe pour les domaines numériques.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Overview and Motivation

This thesis describes research in the field of discrete planning. Broadly speaking, planning is the selection of a set of actions (a plan) that will affect the state of the world in some desirable way. Often this is to move from the initial state of some system to a goal state. Within this framework, discrete planning as discussed in this thesis has the further requirements that:

- There is no randomness.
- The state of the system is fully observable.
- There is a finite set of actions from which to choose.

These are strong assumptions, nevertheless they admit a vast number of possible problems and applications. Historically, discrete planning research has focused on simple examples such as Blocks-World [**1**]. The idea was that by starting simple, useful techniques and concepts could be identified, and the scope of the technology could gradually broaden to include more and more advanced problems. Recently, there has been a trend toward more realistic problems, such as airport traffic control [**2**], oil pipeline flow planning [**3**], and model checking [**4**]. These reflect collaborations between the scientific community and industrial partners who have an interest in planning technology. A more basic application of planning systems is to logistics problems. Here, the system must route packages or passengers to their destinations with a limited supply of trucks or airplanes, while possibly also limiting the amount of fuel or time expended. Needless to say, such problems are often encountered in the real world.

The open-endedness of the problem statement is a source of both frustration and interest. One might imagine an ideal discrete planner which would be able to analyze the task definition, identify important features and characteristics, and select the best algorithm to use. The development of such a system would constitute an significant success for Artificial Intelligence (AI). It would not only provide a powerful tool to be used for other AI projects such as robotics, but the techniques employed would undoubtedly be generalizable. This dream is far away, unfortunately.

One fundamental issue is that computationally intractable problems can be encoded as discrete planning tasks. Because of this there exist broad task families which can never be solved efficiently by planning systems. Researchers in this field are thus faced with the question of defining precisely what they are trying to achieve. Two leading practitioners, Jörg Hoffmann and Bernhard Nebel advocate the following approach [**5**] :

- Identify common features of existing benchmark domains,
- Develop algorithms which perform well on those domains,
- Formalize the class of problems for which the algorithms work well.

This thesis roughly follows the above outline. We focus particularly on heuristic search planning, a widely used technique. Shortcomings and limitations of this paradigm are found, illustrated by simple problems that modern planners cannot efficiently solve. To overcome these limitations, new algorithms are proposed. Finally, we analyze circumstances under which the new algorithms enjoy an advantage over the current methods.

1.2 Planning as Heuristic Search

In discrete planning, the system upon which the agent acts is represented by a set of variables. An assignment of values to variables is called a state. The problem is defined by an initial state, a set of goal conditions, and a set of operators which modify the state in some deterministic way. A solution to the planning problem is a list of actions, called a plan, which leads from the initial state to a goal state, of which there may be many.

Given the relatively abstract definition above, one can see that there are many possible formalisms which would acceptably instantiate the basic idea. Moreover once the formalism has been defined, there are many ways in which the problem can be formulated. Some systems, such as LPG [**6**], are based on iterative plan repair. They start out with a set of operators which is probably invalid, and then iteratively attempt to resolve inconsistencies with it. The SatPlan system [**7**] translates the planning problem into a satisfiability problem, and then invokes a satisfiability engine to generate a solution.

A widely used formulation, and the one we examine in this thesis, is that of planning as heuristic search. The planning problem is represented as a large graph, with each node corresponding to a state of the world. A link between two nodes A and B exists if there is some operator that will transform state A into state B. Now the problem is simply a graph search.

For finite problems, this algorithm is complete: assuming a goal node exists and is reachable, it will be found in finite time. Unfortunately, in practice this will only work for extremely simple problems, because the number of nodes in the graph is exponential in the number of variables. In order to proceed, it is necessary to guide the search somehow, to reduce the number of states that must be explored. To do so we define a heuristic function. This function attempts to estimate, for the state under consideration, the distance to a goal state. This distance is typically measured in number of actions, but other methods can be used to find plans which optimize time, resource consumption, or other plan quality metrics. Finding good heuristics is by no means trivial and constitutes a significant area of interest [**8, 9**].

Equipped with this heuristic function, a standard method such as Best First Search or the A* algorithm [**10**] is applied to find a path from the starting node to a goal node. Best First Search is most often used because it increases the speed with which a solution is found, while the A* algorithm is slower but will produce optimal plans.

Planning systems which operate in this way are called heuristic search planners. The original planning system of this type is called HSP [**11**]. Heuristic search planning is widely used: of the last four International Planning Competitions [**12, 13, 14, 15**], two were won by pure heuristic search planners and another (the most recent) was won by a hybrid system which includes heuristic search [**16**].

Methodology of Thesis. Having introduced the necessary concepts, the methodology the thesis can be stated as follows:

- Critically analyze heuristic search planning,
- Identify limitations and common failure modes,
- Present methods to overcome these limitations.

This process is performed twice, once for standard propositional domains and once for numeric domains. The remainder of this chapter provides an overview of the findings and results.

1.3 Limitations of Standard Heuristic Search Planning Techniques

By definition, heuristic functions are not exact. If the heuristic gives an accurate estimate of the distance from a given node to a goal state, the search will proceed rapidly. In the worst case when the heuristic gives completely erroneous information, the time required for the search will be on the order of the number of nodes in the graph.

It should be noted that while some heuristic functions are better than others, only a limited amount of progress can be made by attempting to define better functions. Finding the exact distance to the goal in the state space graph is equivalent to the planning problem itself. Furthermore, the heuristic must be computed many times (once for each state that is explored), meaning that additional complexity will come at a high price.

Heuristic functions often give *systematically* erroneous estimates about the goal distance. Because of this, many problems cannot be solved except by exhaustive search. A standard technique to deal with systematically misleading information is randomization. The planner described in Chapter 3 interleaves randomized exploration with directed heuristic search. This allows the system to exploit the power of heuristic search in regions where the information is accurate, while providing some degree of robustness to situations in which the heuristic gives incorrect estimates.

A specific issue that often causes the heuristic to report misleading or useless information is the existence of rival goals in the problem. Roughly speaaking, two goal conditions are rival if an action which leads towards one leads away from the other. If this happens, the heuristic will typically assign large regions of the state space an identical value, requiring exhaustive search.

Inspiration from Mobile Robot Path Planning. Path planning for mobile robots is a well-studied problem in the field of robotics [**17, 18, 19**] (see Section 2.9 for a brief discussion). This problem bears some resemblance to discrete planning. In both cases, the system is attempting to find a path from the initial state to a goal state. The main difference is that in mobile robot path planning the set of plans is uncountable, while in discrete planning it is countable.

In high-dimensional path planning, it is often the case that solutions are abundant and optimality is not paramount. A recent line of research by LaValle and Kuffner [**20**] seeks to exploit this observation by defining the Rapidly-exploring Random Tree (RRT). The basic principle is to grow a tree, rooted at the initial position, outward into the state space. The growth process is based on randomization, and biases expansion so that the nodes in the tree will converge to an uniform sampling of the state space of the problem. Because of this, a node will eventually be created in the region of the goal, and so a local planner can connect directly from there to the goal. The success or failure of the RRT method depends not on the dimensionality of the planning problem, but rather on the abundance or scarcity of solutions.

The idea of the first half of this thesis is to apply the concept of RRTs to discrete planning. In the following, the difficulties in translating the RRT idea from one domain to another are discussed. A discrete planning algorithm called RRT-Plan is presented. We show through analysis and empirical testing that there are categories of problems which RRT-Plan can solve efficiently, while other planners cannot.

We conclude that RRT-Plan's success is based on the existence of artificial goal orderings in many domains. In contrast to natural goal orderings which specify a required order in which goals *must* be achieved, an artificial goal ordering is permissible: the full goal *may* be achieved by following it. Using a bad artificial goal ordering can make the problem unsolvable, but if a good one is found it can make the problem much easier. RRT-Plan is effective at obtaining a good artificial goal ordering when one exists.

1.4 Limitations of Heuristic Search Planning in Numeric Domains

The second half of this thesis deals with heuristic search in numeric domains. These domains constitute an important extension of the planning formalism to problems which involve continuous as well as discrete variables. The starting point is the identification of two deep problems.

1.4.1 Two Curses. Two critical issues can be readily identified when attempting to apply the concept of heuristic search to numeric domains. The first, of primary interest for this research, is the Curse of Affluence. We also discuss (but do not address) the converse problem, called the Curse of Poverty.

The Curse of Affluence is caused by unconstrained resources. There are many domains which allow the agent to produce a resource. If the domain does not impose maxima on the amount that can be produced, then the size of the state space becomes infinite. Even if there is a production ceiling, the state space still grows by a factor equal to the number of values the variable can take on.

In finite state spaces, there are almost always small regions which must be searched exhaustively. These are caused by incorrect estimates provided by the heuristic function. When the state space becomes infinite, these formerly small regions can no longer be searched exhaustively. This causes a standard search procedure to fail.

The Curse of Poverty involves problems in which certain resources must be conserved in order to reach the goal. The issue is that it is basically impossible to find meaningful lower bounds on resource requirements. This is because the relevant lower bounds are often not specific to one variable, but to combinations of variables. In order to find a plan which conserves the appropriate resources, it is necessary to prune states which have insufficient resource availability. However, because of the difficulty of obtaining lower bounds, this is quite hard to do.

1.4.2 Reduced Search with Enhanced States. In traditional heuristic search, each node in the search tree represents a state of the problem, and the state is fully specified. The contribution of Chapter 4 is a method for handling the Curse of Affluence by constructing enhanced states, which represent many actual states. This will significantly compress the search space.

An enhanced state is created when a simple operator becomes applicable. Deferring the precise definition of "simple operator", consider the case of a refuel action in a Logistics domain. This action has some preconditions but no propositional effects. The only effect is to increase the fuel. This implies that it can be applied multiple times (barring a maximum fuel level). A standard search algorithm would create multiple successor states, each representing a certain number of refuel actions. A reduced search creates one enhanced state. This enhanced state carries information relating to the range of values the fuel variable can take on by inserting the refuel operator.

In order to represent the reachable numeric space associated with an enhanced state, a connection is made to the concept of the convex hull. This representation permits queries, updates, and expansions to be carried out in a natural way. Furthermore, the convex hull has been widely studied in the computational geometry literature, and efficient algorithms exist for computing it [**21, 22**].

1.5 Thesis Contributions

The underlying theme of this thesis is the recognition of limitations of heuristic search planning, and the development of techniques to circumvent those limitations. These techniques constitute the major contributions of the thesis, and are instantiated by the development of two planning systems:

- RRT-Plan, a randomized algorithm for STRIPS planning (Chapter 3)
- Convex Hull Causal Graph Planner (CHCGP), a planning system which uses **reduced search** to compress the numeric state space (Chapter 4)

Two minor contributions related to CHCGP are the extension of the Causal Graph heuristic [**23**] to numeric domains, and the development of a method that can find lower bounds on resource consumption in certain cases.

The motivation for these planning systems comes in part from a critical analysis of the limitations of heuristic search planners in various contexts. This analysis constitutes an additional contribution of the thesis. Planning domains which illustrate the limitations are developed and presented in Appendix B.

1.6 Organization

Chapter 2 gives background on previous discrete planning systems, robot motion planning, and convex hull concepts. Chapter 3 describes RRT-Plan, a discrete planning algorithm based on the idea of RRTs. Chapter 4 introduces the notion of reduced search with enhanced states. Chapter 5 provides some concluding remarks. Appendix A describes technical details relating to the implementation of CHCGP. Appendix B presents the planning domains used for experimental evaluations of the systems described in Chapters 3 and 4.

CHAPTER 2

Background

We begin this chapter with a brief introduction to the STRIPS language, giving definitions and two classic examples. Definitions regarding the numeric version of STRIPS are then given. Next, we review results about the complexity of STRIPS planning. The majority of the chapter is spent on a discussion of historically important planning systems. In particular three important heuristic planners are described - HSP, Fast Forward (FF), and Fast Downward, as well as a version of FF that can solve numeric planning problems. A brief discussion of mobile robot path planning algorithms is given; the method of Rapidly-exploring Random Trees is used in Chapter 3. The chapter concludes with a consideration of the convex hull, a concept from computational geometry that is relevant to Chapter 4.

2.1 STRIPS planning

In 1971, Fikes and Nilsson introduced one of the first discrete planning systems [**24**]. They named their system STRIPS, for Stanford Research Institute Problem Solver. Since then the word STRIPS has come to refer primarily to the language used by the system, rather than the system itself. Most discrete planning research has used the STRIPS language.

Information represented by the STRIPS language is divided into two parts, a domain definition and a problem specification. The domain definition contains a set of **predicates** and **operators**. The predicates indicate properties of objects or relationships between objects. The operators correspond to the types of actions that the agent can choose from. A problem specification provides a set of **objects**, which are used to instantiate the predicates and operators. A instantiated predicate becomes a **proposition** (also called **atom** or **fact**), an instantiated operator becomes a **grounded operator**. A **state** s of the system is simply a set of propositions. The problem definition also includes a set of propositions called the **initial state** s_i, which has an obvious meaning, and another set called the **goal conditions** s_*. A state s such that $s \supseteq s_*$ is a goal state.

The set of grounded operators in the problem is denoted $\mathcal{O}$. For every operator $o \in \mathcal{O}$, there are three associated proposition sets: add effects $add(o)$, delete effects $del(o)$, and preconditions $prec(o)$. An operator is applicable in a state s if $prec(o) \subseteq s$. The result of applying an operator to s is:

$$result(s, o) = \{s \setminus del(o)\} \cup add(o) \tag{2.1}$$

A plan is a sequence of operators $\{u_1, u_2 \ldots u_n\}$. A plan combined with an initial state s_i creates a sequence of states $\{s_0, s_1, \ldots s_n\}$ such that $s_0 = s_i$ and

$$s_k = result(u_k, s_{k-1}) \tag{2.2}$$

A **valid** plan is one for which $s_k \supseteq prec(u_{k+1})$. A **solution** is a valid plan where $s_n \supseteq s_*$. We now give two examples of classic STRIPS planning domains.

2.1.1 Example - Blocks World.

This domain (Figure 2.1) depicts stacks of blocks on a table. The initial state is given as some configuration of blocks and the goal is to achieve some other configuration. The agent is allowed to move the top block from a stack onto the table or onto some other block that is clear. The goal is a set of *ON* propositions which partially specify a stack of blocks.

An interesting observation about this domain is that certain sets of propositions are inconsistent. For example, it is impossible for both *ON*(A, B) and *ON*(B, A) to be true at the same time. There is also a simple invariant function of the number of blocks and the number of each proposition type that remains constant under all operator applications. Thus one way of approaching STRIPS planning is to attempt to discover these rules and invariants. Another interesting fact about this domain is that there are **natural goal orderings**. These are orderings in which the subgoals must be achieved for the full problem

```
PREDICATES: clear(?a), on(?a, ?b), ontable(?a)
OPERATORS:
      MOVE(?a ?b ?c)
            prec: clear(?a), on(?a ?b), clear(?c)
            add: on(?a ?c), clear(?b)
            del: on(?a, ?b), clear(?c)
      MOVEFROMTABLE(?a ?b)
            prec: clear(?a),  clear(?b), ontable(?a)
            add: on(?a ?b)
            del: clear(?b), ontable(?a)
      MOVETOTABLE(?a ?b)
            prec: clear(?a), on(?a ?b)
            add: ontable(?a), clear(?b)
            del: on(?a ?b)
```

FIGURE 2.1. The Blocks World domain definition.

to be solved. Specifically, the blocks at the base of a column must be stacked before the higher blocks.

2.1.2 Example - Logistics.

Another well-studied STRIPS domain is Logistics (Figure 2.2). This domain involves a set of packages that must be delivered to various locations. The packages can be driven around after being loaded into a truck. A *link* predicate indicates that two locations are connected, so a truck can drive from one to the other. Note the way type predicates are used to ensure that trucks aren't loaded into other trucks and packages can't drive themselves.

Like Blocks-World, Logistics has strong invariants regarding the relationship between the number of trucks, packages, and propositions. Specifically, there must be exactly one true *at* proposition for every truck and exactly one true *at* or *in* proposition for every package. Also, there are constant predicates (*link*) which never change.

Unlike Blocks-World, goals in the Logistics domain can be achieved in any order. Furthermore, propositions corresponding to the locations of trucks do not depend on any other non-constant propositions. A truck can be driven around regardless of the status of the other packages or trucks. This observation is exploited by the planning system Fast Downward (see Section 2.8).

```
PREDICATES: at(?t ?c), in(?p ?t), link(?ca ?cb),
      truck(?t), package(?p)
OPERATORS:
      LOAD(?p ?t ?c)
            prec: at(?p ?c), at(?t, ?c),
                  truck(?t), package(?p)
            add:  in(?p ?t)
            del:  at(?p ?c)
      UNLOAD(?p ?t ?c)
            prec: at(?t ?c), in(?p ?t),
                  truck(?t), package(?p)
            add:  at(?p ?c)
            del:  in(?p ?t)
      DRIVE(?t ?ca ?cb)
            prec: at(?t ?ca), link(?ca ?cb), truck(?t)
            add:  at(?t ?cb)
            del:  at(?t ?ca)
```

FIGURE 2.2. The Logistics domain definition.

2.1.3 STRIPS planning with Numeric Variables. Over the years various efforts have been made to extend the STRIPS language to allow it to handle a broader class of problems. These efforts produced several more advanced language features, including the ability to represent first-order logic [**25**], temporal constraints [**26**], and most recently goal preferences [**27**].

This work is concerned with STRIPS planning with numeric variables. To enable this, the notion of numeric **functions** was introduced [**26**]. These are analogous to predicates, except an instantiated function has continuous, rather than binary, values. A state s thus includes, in addition to a list of propositions, an assignment of values to functions. In this document functions are called **numeric variables**. Functions can be incorporated into the planning problem in three ways:

- They can constrain the applicability of operators.
- They can be changed by the application of operators.
- They can participate in **goal metrics**, supplying a means of evaluating the quality of plans in a way other than by plan length.

In this work the focus is on the first two roles. Numeric constraints (item 1) and numeric effects (item 2) are evaluated in terms of *expressions*. The syntax of expressions in PDDL 2.1 is quite simple:

$$< expression >::= (< op >< expression >< expression >) \mid < constant > \mid < function >$$

Here *op* is one of the standard arithmetical operations {+, -, *, /}. Numeric constraints are tuples $(exp_1, relop, exp_2)$, which are evaluated by comparing the two expressions using one of the standard relational operators $\{\leq, <, =, >, \geq\}$. Numeric effects, similarly, are tuples of the form $(targ, assignop, exp)$ which changes the value of the function $targ$ by evaluating exp and applying the given assignment operator, which is one of {+=, -=, *=, /=, :=}. Note that when applying an operator, constraints are evaluated before effects, and effects are evaluated simultaneously.

2.2 Theoretical Considerations

Several researchers have studied the complexity of STRIPS planning in various forms [**28, 29**]. In this section we focus on the work of Bylander [**1**], which deals with standard STRIPS, and Helmert [**30**], which studies STRIPS planning with numeric variables. We will give proof outlines for two of the simplest and most powerful results and state several others.

Because basic STRIPS tasks consist of a finite set of propositions N, the size of the search space is bounded by 2^N and the problem is decidable, albeit potentially costly. A brute force algorithm that simply visits every state will be complete.

Given this fact, we now proceed to consider the complexity of STRIPS planning. Let $PLANSAT$ be the problem of determining if a plan exists for a STRIPS planning task. In particular, consider a restricted version $PLANSAT_{1+}$ of the problem where operators are allowed to have only one positive effect, i.e. $del(o) = \emptyset$ and $|add(o)| = 1$ for all operators.

Theorem $PLANSAT_{1+}$ is NP-complete (Bylander [**1**]).

Hardness is shown by a polynomial reduction to 3SAT, which is known to be NP-complete [**31**]. We consider a 3SAT problem which consists of M literals and K clauses, and give a STRIPS task that is polynomially equivalent. The domain is given in Figure 2.3.

There are two operators which represent making literals true or false, and two which satisfy a clause of the problem if the literal has the correct value.

```
PREDICATES: TRUE(?LIT), FALSE(?LIT), SAT(?CLS),
      ENTERSTRUE(?CLS, ?LIT), ENTERSFALSE(?CLS, ?LIT)
OPERATORS:
      MAKETRUE(?LIT)
            prec: NOT(FALSE(?LIT))
            add: TRUE(?LIT)
      MAKEFALSE(?A)
            prec: NOT(TRUE(?LIT))
            add: FALSE(?LIT)
      SATCLAUSETRUE(?CLS, ?LIT)
            prec: TRUE(?LIT), ENTERSTRUE(?CLS, ?LIT)
            add: SAT(?CLS)
      SATCLAUSEFALSE(?CLS, ?LIT)
            prec: FALSE(?LIT), ENTERSFALSE(?CLS, ?LIT)
            add: SAT(?CLS)
```

FIGURE 2.3. Domain definition for the 3SAT reduction.

The initial state is just a list of propositions specifying the way literals enter into clauses. Thus if clause A is $(r \vee s \vee \neg t)$, then the initial state would include $ENTERSTRUE$(A, r), $ENTERSTRUE$(A, s), $ENTERSFALSE$(A, t). Note that the literals are initially in an "unassigned" state, but once an assignment is made it cannot be reversed. The goal is for all clauses to be satisfied. A solution to this STRIPS task would give a solution to the corresponding 3SAT problem, so $PLANSAT_{1+}$ is NP-hard.

Propositions are never deleted, and so every state is strictly larger than its predecessor. Therefore if a solution exists it is linear in the number of propositions in the task. Because of this $PLANSAT_{1+}$ is in NP, and therefore NP-complete.

Note that if instead of defining the predicates $ENTERSTRUE$ and $ENTERSFALSE$ we had simply written out 3 operators for each of the K clauses explicitly, such that each operator would have a precondition on a *specific* proposition (e.g., $TRUE(w)$, where w is an object listed in the task definition), the operators would now be limited to a single precondition. Thus even $PLANSAT_{1+}^{1}$, where operators have single preconditions and one positive effect, is NP-complete.

Bylander [**1**] also proves that the following problems are PSPACE-complete:

- $PLANSAT$: operators have general preconditions and effects;
- $PLANSAT^1$: operators have only one precondition and arbitrary effects;
- $PLANSAT_2^{2+}$: operators have two positive preconditions and two total add or delete effects.

The following problems are polynomial:

- $PLANSAT_1^+$: operators have positive preconditions and one add or delete effect.
- $PLANSAT^1(g)$: operators have one precondition and the number of goals is limited to a constant g.
- $PLANSAT^0$: operators have no preconditions.

The first item has an elegant proof which is based on the observation that the operators in a valid $PLANSAT_1^+$ plan can be reordered such that all the operators with positive effects come before all the operators with negative effects. Bylander also shows several results for the problem of finding an optimal plan, $PLANMIN$. Clearly this is strictly harder than the $PLANSAT$ problem, and only highly restricted versions are tractable.

We now discuss the work of Helmert [**30**] who studied the complexity of STRIPS planning with numeric variables. Helmert defines the problem as $PLANSAT(\mathcal{G}, \mathcal{P}, \mathcal{E})$ where $\mathcal{G}$ and $\mathcal{P}$ are the types of numeric constraints allowed on the goal and on operators, respectively; $\mathcal{E}$ denotes the type of numeric effects allowed. The constraints $\mathcal{G}$ and $\mathcal{P}$ are classified as follows:

- $\mathcal{C}_\emptyset$: no numeric conditions.
- $\mathcal{C}_0$: compare a variable to zero.
- $\mathcal{C}_c$: compare a variable to a constant.
- $\mathcal{C}_=$: compare two variables to one another.
- $\mathcal{C}_p$: compare a polynomial of a single state variable to zero.
- $\mathcal{C}_{p+}$: compare a polynomial of the state variables to zero.

The following families of numeric effects are also considered:

- $\mathcal{E}_{\emptyset}$: no numeric effects.
- $\mathcal{E}^{=c}$: assign a constant.
- $\mathcal{E}_{+1}$: increase a variable by one.
- $\mathcal{E}_{\pm 1}$: increase or decrease a variable by one.
- $\mathcal{E}_{\pm c}$: increase or decrease a variable by a constant.
- $\mathcal{E}_{p}$: assign to one variable a polynomial function of the other variables.

There are several other types, but for the most part they end up being equivalent to the ones listed above. The first result given by Helmert, which we summarize, is the following theorem.

Theorem $PLANSAT(\mathcal{C}_{p+}, \mathcal{C}_{\emptyset}, \mathcal{E}_{+1})$ is undecidable (Helmert, [**30**]).

The proof is based on the decision problem of determining if a Diophantine equation has a solution in the natural numbers (a Diophantine equation is simply a polynomial in several variables). This is Hilbert's Tenth Problem, which Matiyasevich showed to be undecidable [**32**].

The simple numeric STRIPS task to which we reduce the above problem is as follows. There are no propositions in the problem. The are N numeric variables which are initialized to zero. Each variable v_i has a corresponding operator which increments it. The goal is to achieve a value of the v_i such that $p(v_1, v_2, \ldots v_n) = 0$, for some polynomial p. This planning problem can be decided only if the Diophantine equation decision problem can be decided. Thus numeric STRIPS planning, even under the rather restrictive conditions imposed above, is undecidable.

Helmert goes on to show that numeric STRIPS planning is undecidable in the following cases (referring to the notation given above):

- $PLANSAT(\mathcal{C}_{=}, \mathcal{C}_{\emptyset}, \mathcal{E}_{p})$
- $PLANSAT(\mathcal{C}_{0}, \mathcal{C}_{\emptyset}, \mathcal{E}_{p})$
- $PLANSAT(\mathcal{C}_{\emptyset}, \mathcal{C}_{0}, \mathcal{E}_{\pm 1})$
- $PLANSAT(\mathcal{C}_{\emptyset}, \mathcal{C}_{=}, \mathcal{E}_{+1})$

The latter two results seem particularly discouraging. It should be noted, however, that the results for $\mathcal{E}_{\pm 1}$ and $\mathcal{E}_{+1}$ depend on the fact that there are no limits to the values the variables may take on. If such limits existed or could be imposed (as may be the case in many practical applications), the state space would become finite and thus the problem would be decidable. Helmert goes on to demonstrate that $PLANSAT(\mathcal{G}, \mathcal{P}, \mathcal{E})$ is decidable if the following conditions hold:

- $\mathcal{E} \in \{\mathcal{E}_{+1}, \mathcal{E}_{+c}, \mathcal{E}_{+1}^{=c}, \mathcal{E}_{+c}^{=c}\}$, $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} \notin \{\mathcal{C}_{=}, \mathcal{C}_{p+}\}$
- $\mathcal{E} \in \{\mathcal{E}_{\pm 1}, \mathcal{E}_{\pm c}, \mathcal{E}_{\pm 1}^{=c}, \mathcal{E}_{\pm c}^{=c}\}$, $\mathcal{G} \neq \mathcal{C}_{p+}$, $\mathcal{P} = \{\mathcal{C}_{\emptyset}\}$

2.3 GraphPlan

GraphPlan was introduced by Blum and Furst [**33**] and marks an important advance in the history of discrete planning. GraphPlan achieved a marked improvement in performance over previous planners and strongly influenced later systems. Fast Forward (see below) and LPG [**6**] both rely on the notion of the planning graph ("LPG" stands for Local search on Planning Graphs). GraphPlan attempts to solve standard STRIPS problems.

Unlike the heuristic search planners discussed later in this chapter, GraphPlan is a partial order planner. This means it allows non-interfering operators to be executed simultaneously. For example, if a planning problem involves two packages that must be delivered, and has two trucks available to deliver them, then GraphPlan will return a plan in which drive-truck operators are simultaneous.

The central idea of the GraphPlan system is that of a **planning graph**. This is a directed, levelled graph. The levels alternately represent propositions and operators that could be active in a given epoch. An epoch is a timestep during which several operators may be applied. An arc from a proposition p to an operator o in the next action layer exists if $p \in prec(o)$. An arc from an operator o to a proposition p in the next fact layer exists if $p \in add(o)$.

The growth process for the planning graph is as follows. The zeroth level of facts represents the initial state. Each operator layer is constructed by finding the operators for which all preconditions are contained in the previous fact layer. The subsequent fact layer contains all facts that are added by some operator in the previous action layer.

Some observations about the above process are in order. First, because delete effects are not considered, the graph will grow quickly and then level off. If at any time two consecutive fact layers contain the same set of atoms, then all subsequent layers will be identical, and so the growth process can be terminated. Any atom which is not present in the final layer of the graph is unachievable. In the worst case, the number of fact layers in the graph is bounded by the number of atoms in the problem.

Unfortunately the planning graph does not directly yield a valid plan. However, a plan can be extracted from the planning graph, by "activating" subsets of action nodes in each layer. Care must taken when performing this activation, because many actions will conflict with one another in some way, and so cannot be simultaneously activated. Indeed, detecting and exploiting these mutual exclusion (mutex) relationships is an important additional component of the GraphPlan system.

As the planning graph is being constructed, mutual exclusion relationships are calculated between nodes within a given layer. Two propositions p_A and p_B are marked as mutually exclusive if every action which adds p_A is mutex with every action which adds p_B. A pair of actions can be marked as mutex for two reasons:

- **Interference** If either of the actions deletes a precondition or add effect of the other.
- **Competing Needs** If there exists a precondition of one action which is mutex with a precondition of the other in the preceding fact layer.

This technique does not find all possible mutual exclusion relationships. Interlayer mutexes are not considered. There could also be tertiary mutexes such that for a given set of three facts, only two can be true simultaneously. Binary mutexes do succeed in capturing important information. For example in a Logistics domain, there are many facts that can represent the location of a single truck. Clearly only one of these propositions can be true at any given time. The mutex propagation system will mark these facts as mutually exclusive.

Once the planning graph has been constructed, the plan extraction algorithm can be summed up with the code sketch of Figure 2.4. The function EXTRACT_PLAN is called on a set of goal atoms $\mathcal{G}$ and a layer in which they are to be achieved. FACTS(i) and MUTEX(i) refer to the achievable facts and proposition mutexes in goal layer i, respectively.

If this function fails, the planning graph is extended by another level and a new attempt is made. Note that while subsequent levels will not have more propositions, they might have fewer mutex relationships.

```
function EXTRACT_PLAN(goals $\mathcal{G}$, level $i$)
  if $\mathcal{G} \not\subseteq$ FACTS(i) or $\exists g_i, g_j \in \mathcal{G} : \{g_i, g_j\} \in$ MUTEX($i$) then
    return FAIL
  end if
  if $i = 0$ then
    return SUCCESS
  end if
  for each non-mutex action set $\mathcal{A} : \mathcal{G} \subseteq add(\mathcal{A})$ in operator level $i$ do
    if EXTRACT_PLAN($prec(\mathcal{A}), i - 1$) then
      return SUCCESS
    end if
  end for
  return FAIL
```

FIGURE 2.4. The GraphPlan plan extraction algorithm.

From the above description, we see that the algorithm can be classified as a regression search. It starts from the goal and searches backwards toward the initial state. The major pitfall of regression planners is that they can often spend large amounts of time searching backwards from states which are not actually achievable themselves. If the mutual exclusion reasoning works well, then this problem is ameliorated; but the mutex information is not comprehensive, as noted. In contrast, forward search planners are always sure that every state they evaluate is reachable.

2.4 SatPlan

Kautz and Selman are well-known for their work on the GSAT algorithm [**34, 35**]. GSAT is able to solve SAT encodings of empirically difficult NP-hard problems such as graph coloring. The success of GSAT motivated them to attempt to formulate planning as satisfiability [**7**]. In the following we refer to STRIPS propositions as facts so as to avoid confusion with the new SAT propositions.

The first step in transforming discrete planning into satisfiability is to add a new argument to the facts. This argument denotes the timestep at which the fact is true. Thus instead of AT(T, C) we have AT(T, C, 4). This simply means that AT(T, C) is true in the fourth timestep. It is easy to encode the initial state with the above system. Encoding the goal conditions requires an existentially quantified formula such as:

$$\exists i. \quad at(P_1, C, i) \wedge at(P_2, D, i) \tag{2.3}$$

This would represent the requirement that at some timestep package P_1 is at location C and P_2 is at location D.

A similar set of propositions are introduced to represent operator application. These are of the form **DRIVE**(T, C, D, 5), which is true if the given action was performed in the fifth timestep. For each action type we must also create an axiom to represent preconditions and effects. For example, the **DRIVE** action in the Logistics domain takes the following form:

$$\forall t, a, b, i. \quad drive(t, a, b, i) \supset (at(t, a, i) \wedge at(t, b, i + 1) \wedge \neg at(t, a, i + 1)) \tag{2.4}$$

Notice that preconditions and effects are treated symmetrically. Another set of axioms ensure that some operator is applied at every timestep, and that only one action occurs at a time.

Simple logical rules are used to rewrite the above axioms and propositions into a Boolean satisfiability sentence. Note that quantifiers involving the timestep i pose some difficulty, since there is no obvious limit to the values that i can take on (i.e., the length of the plan cannot be known in advance). This is solved by imposing some arbitrary value N, and then trying larger values if failure is reported. Once the problem has been rewritten in conjunctive normal form, a satisfiability engine is invoked to find an acceptable truth assignment, which corresponds to a valid plan.

Kautz and Selman argue that progress in SAT solver technology advances rapidly, and that their formulation allows such progress to carry over directly to discrete planning. This argument is backed up by good empirical results. Another argument made in favor of the formulation is that it allows a more sophisticated specification of goal conditions. For example, if for some reason package P_k can never be placed in truck T_j, this requirement can be encoded as:

$$\forall i. \quad \neg in(P_k, T_j, i) \tag{2.5}$$

In later work [**36**] Kautz and Selman examine the possibility of combining SAT-based planning with planning graph techniques. The mutexes provided by the planning graph give a set of additional constraints on the problem. These constraints can be written as axioms and compiled into the satisfiability sentence. Any valid plan must follow these constraints anyway, so knowing them in advance is potentially valuable.

Mutex computation can be considered a special case of *limited inference* that is specific to the discrete planning problem. Limited inference is the application of logical rules to the SAT sentence before sending it to the solver. More general limited inference techniques exist and can sometimes also be beneficial to use.

2.5 Heuristic Search Planner - HSP

The concept of planning as heuristic search was introduced by Bonet and Geffner [**11**]. They named their planning system HSP, for heuristic search planner[1]. The basic idea of planning as heuristic search was given in the introduction, so we proceed directly to a more specific discussion.

In order to create a reasonably efficient heuristic search planner, one must first define a good heuristic function. This is some function $h(s)$ that, given a state s, estimates the distance to the goal. Observe that this notation $h(s)$ is somewhat misleading: it would be more proper to write $h = h(s, s_*, \mathcal{O})$, as the goal conditions and operator list must also be taken into account (though they are constant throughout the problem instance).

The heuristic $h(s)$ defined by [**11**] is based on the idea of the relaxed planning problem. This is a simplification of the original problem where the delete effects of all operators are removed. Under a simple assumption, every solution to the relaxed planning problem is also a solution to the real planning problem. Furthermore, deciding if a relaxed planning problem has a solution can be done in polynomial time. This specific heuristic function is denoted $h^+(s)$.

The assumption referred to above is that all preconditions are positive (i.e., a precondition cannot require a proposition to be false). Note that all STRIPS planning problems can be transformed to satisfy this requirement by the introduction of *inverted* propositions.

[1]This name causes some confusion, since many other planning systems inspired by HSP are also heuristic search planners. For that reason we refer to the system described in [**11**] solely by its acronym HSP.

```
function ATOM_COST($s$)
  atom cost $g(p) = \infty, \forall p$
  set $\mathcal{P} = \emptyset$
  priority queue Q
  for each prop $p \in s$ do
    INSERT(Q, (0, $p$))
  end for
  while size(Q) $> 0$ do
    $(g(p), p)$ = REMOVE(Q)
    for each operator $o : prec(o) \not\subseteq \mathcal{P}, prec(o) \subseteq \{\mathcal{P} \cup p\}$ do
      $c = intrinsic\_cost(o)$ + AGGREGATE($g(prec(o))$)
      for each $p \in add(o)$ do
        UPDATE(Q, $(c, p)$)
      end for
    end for
    $\mathcal{P} := \mathcal{P} \cup p$
  end while
```

FIGURE 2.5. The HSP heuristic calculation.

Thus the relaxed planning problem is a case not considered by Bylander [**1**] which may be denoted $PLANSAT_{+}^{+}$ (all preconditions and all postconditions are positive).

Because all solutions to the relaxed problem are also solutions to the real problem, if the optimal relaxed solution could be found it would constitute an *admissible* estimate of the distance to the goal. That is, the estimate would provide a guaranteed lower bound. Unfortunately, Bylander showed even $PLANMIN_{1+}^{1+}$ to be NP-complete. Therefore we must be content with an approximation of optimal relaxed plan distance. This approximation can be obtained using the algorithm shown in Figure 2.5.

The ATOM_COST function maintains an queue of all propositions p in the problem. When a proposition comes out of the queue, the associated cost is guaranteed to be the lowest possible. Furthermore, all operators that become applicable by the addition of p to all other propositions that have already been processed ($\mathcal{P}$) also have lowest possible cost. Thus all propositions and all operators will be considered at most once by the above algorithm.

The atom cost of each p is given by $g(p)$. The routine AGGREGATE gives an evaluation of an operators cost based on the costs of its preconditions $g(prec(o))$. The authors consider two possibilities for AGGREGATE: either using the maximum cost of the elements $prec(o)$ or the sum of the costs. The former technique achieves admissibility, but the latter is preferred by the authors because it is more informative. The final heuristic evaluation of the state is AGGREGATE(s_*).

An alternate way of formulating the above process is through a fixed point calculation. For every operator o with precondition C that adds an atom p, the following update rule is applied:

$$g(p) := min[g(p), intrinsic\_cost(o) + g(C)] \tag{2.6}$$

Here $g(C)$ = AGGREGATE(C). This process is continued until the costs cease to change.

Bonet and Geffner use 1 for $intrinsic\_cost(o)$. This will give an estimate of goal distance in terms of the number of operators required to reach it. However, the above algorithm would work exactly the same way using many alternate definitions of the intrinsic cost of the operator. If it were desired to evaluate the quality of a plan based on criteria other than plan length, the intrinsic cost could be appropriately modified. Furthermore, if one wanted to find lower bounds on the consumption of some resource, the intrinsic cost could be defined as the operator's usage of that resource.

Given the heuristic function, a variety of search algorithms can be used. HSP uses hill-climbing, to attempt to obtain a solution as quickly as possible, regardless of the quality of the solution.

2.6 Fast Forward

Fast Forward (FF), a planning system developed by Hoffmann and Nebel [**5**], represents a significant advance in heuristic planning technology. In basic structure it is similar to HSP. However, FF contains several important refinements.

2.6.1 New Relaxed Plan Estimation Technique. FF's first refinement is to use a more sophisticated method than the algorithm shown in Figure 2.5 to approximate h^+. This is basically to run GraphPlan [**33**] on the relaxed planning problem. In addition to providing better heuristic estimates, information created by the application of the GraphPlan algorithm is used to assist with several search shortcuts (see Sections 6.2 and 6.3 below).

Consider what happens when the GraphPlan algorithm is run on a relaxed planning problem. First observe that the planning graph grows at exactly the same rate as in the standard problem, since the delete effects are not considered in the growth process. Furthermore, there will be no mutual exclusion relationships between any atoms or operators. This can be seen that noting that the first mutex added in a normal problem must be caused by the Interference rule; the other mutex rules are defined recursively in terms of other mutexes. Since the Interference rule is based on delete effects, it will never apply for the relaxed problem. Because there are no mutexes, *any* choice of operators which support the goal atoms will constitute a valid relaxed plan, so the action set selection process will never backtrack. Finally, the GraphPlan algorithm guarantees that the resulting plan will be optimal in terms of makespan (the number of epochs of simultaneous operator application).

The planning graph based technique of approximating h^+ takes into account the fact that a single operator can add two goal atoms. Thus it usually gives better estimates of goal distance than the HSP technique. The remaining difficulty is selecting minimal action sets at each level in the planning graph. This turns out to be NP-complete also, but for this detail it would seem that an *ad hoc* solution is acceptable.

FF runs in two phases. The first is a highly optimized but incomplete phase which uses several "shortcuts" as described below. The second phase runs if the first fails, and is slower but complete. In Chapter 3 the distinction between these phases will become relevant, as only the faster version is used in RRT-Plan.

2.6.2 Helpful Actions. When applying a search algorithm to a large space, informed pruning is often beneficial. If it can somehow be determined that a certain edge should not be followed (or, equivalently, if a certain state should not be expanded), large regions of the search space can be bypassed. The idea of the **Helpful Actions** technique is to identify certain operators as the most likely candidates to lead to the goal and prune all other edges.

The technique arises naturally from the use of GraphPlan on the relaxed version of the problem. GraphPlan backtracks from the final layer of the planning graph, selecting action sets at each layer which support the subsequent layer's preconditions. The helpful actions are simply the operators selected in the first layer.

The fast phase of FF only follows edges in the search tree that correspond to helpful actions. This can significantly reduce the search space, but renders the search incomplete.

2.6.3 Added Goal Deletion. The technique of added goal deletion is motivated by the fact that many domains have strong goal ordering constraints [**37**]. In such domains, certain goals must be achieved before other goals. This implies that a goal which is achieved out of order will eventually need to be deleted, amounting to a backtrack in the search space. In most cases, obtaining a full goal ordering is equivalent to the planning task itself. This motivates us to make an informed guess of the goal ordering, and use this guess to determine whether or not to expand a node.

Again, the information used to formulate this guess comes from the application of GraphPlan to the relaxed task. For a given state s, a relaxed plan P' is obtained. A set is constructed including all the delete effects of the operators in P'. If this set contains a goal atom p that was added by the operator which led to s, then it is assumed that p was achieved out of order, and no successors to s are created.

As with helpful actions, this technique can often be highly effective, but also renders the search incomplete. A similar technique called Goal Subset Locking is an integral part of the RRT-Plan algorithm presented in Chapter 3.

2.6.4 Enforced Hill Climbing. The highly optimized first phase of FF utilizes the Helpful Actions and Added Goal Deletion methods, and an alternate search algorithm. This algorithm, called Enforced Hill Climbing, is simply Best First Search except that the open list is cleared when a state with lower heuristic value is discovered.

Enforced Hill Climbing works well when there are no dead ends in the search space. If a dead end state is mistakenly assigned a low heuristic value and the search reaches from it, the algorithm will fail. However, Hoffmann showed [**38**] that many of the planning benchmarks do not contain dead ends. Furthermore, this problem only occurs if the dead end is **unrecognized**.

2.7 Metric-FF

Metric-FF by Hoffmann [**39**] is an attempt to apply the technology of Fast Forward to numeric problems. The basic idea is again to construct a relaxed planning graph and use that to derive a heuristic estimate of the goal distance. The main concern is how to modify the planning graph constuction process to support numeric variables.

Hoffmann notes that the heuristic employed by FF has three desirable properties: *admissibility*, *polynomiality*, and *basic informedness.* Only the third property requires elaboration. Basic informedness requires that a zero-length plan is a solution for the relaxed problem if and only if it is also a solution for the real problem, and that the first operator applied in the relaxed plan actually is applicable in the given state. This property guarantees that a heuristic value of zero can only be achieved by a state which satisfies the goal conditions.

The basic strategy is to define a restricted version of a numeric planning problem in which relaxation concepts can be directly applied. Further development gradually removes some of the restrictions. The starting, highly simplified language is as follows. The only constraints allowed are those which use $\geq$ and $>$. The only effects allowed are += and -= (simple assignment, :=, is not allowed). The only expressions allowed are constants. Essentially we only allow the problem to add or subtract constants from the variables, and the goal is simply to raise those variables above some threshold.

Given this restricted language, we can obtain a relaxed plan heuristic simply by ignoring the subtraction effects. Since we allowed only $\geq$ and $>$ comparisons, the subtraction effects cannot bring us closer to the goal. The analogy to ignoring delete effects should be clear. The three properties listed above are easily achieved.

2.7.1 Monotonicity and Dynamic Relaxation. We now wish to extend the notion of ignoring subtraction effects while allowing non-constant effects. In general it cannot be determined in a preprocessing phase whether a non-constant effect will result in an increase or decrease of the target value. This is because variables (and expressions) can be negative. For example the assignment (v^i, +=, v^j) can result in a decreased value of v^i if v^j is negative. For this reason it is necessary to consider a *dynamic* relaxation in which the results of an effect are evaluated in the context of the actual state. If the effect evaluates to a negative value it is ignored.

This requires the problem to satisfy a new set of (less restrictive) conditions. These conditions require that increasing the value of a variable in a state s:

(1): will not violate a constraint if the constraint is satisfied in s,

(2): will never decrease the value of an expression in a numeric effect.

(3): will never decrease the magnitude of change caused by a numeric effect, *except for simple assignments.*

And furthermore, regarding the problem generally:

(4): All expressions diverge as their variables diverge.

(5): There is some set of values which satisfy all the goal conditions.

If the domain satisfies requirements (1-2), it is called **monotonic**. If it satisfies requirements (3-5), it is called **strongly monotonic**. Monotonicity ensures that every solution of the real planning task is also a solution of the relaxed task. For example if (1) did not hold, a real plan could involve an action which decreases a variable and so satisfies a constraint. But the relaxed version of this plan would ignore the decreasing effect, and so the outcome might not satisfy the goal constraint.

Strong monotonicity guarantees that the relaxed planning problem can be decided in polynomial time. The contribution of rules (4-5) to this guarantee should be obvious. The importance of rule (3) can be seen by considering an effect $(v, +=, 1 - \frac{v}{2})$. Assuming $v < 2$ to begin with, each application of the effect increases v, but it can never grow larger than 2. The numeric planning graph construction algorithm cannot detect this kind of convergence, so situations leading to it must be disallowed.

Simple assignments exist in many planning domains and do not satisfy Rule (3), so they must be excepted. For example the change caused by assigning a value to a constant (which is certainly an action we would want to allow) will decrease as the value increases. This issue is circumvented by posing an alternate requirement that the problem not have *acyclic* simple assignment effects, which will also guarantee that the planning graph growth ends in polynomial time.

2.7.2 Linear Tasks. A large number of planning tasks are *linear.* This means that only addition, subtraction, and simple assignment effects are allowed, and all expressions are formed by linear sums of variables. Notice that all constraints involving general comparisons of linear expressions can be rewritten with a single comparison operation such as $\geq$.

In order to transform linear tasks into monotonic tasks, **inverted variables** are introduced. These are new numeric variables which are always the exact negative of their standard counterparts. Now all expressions can be rewritten so as to be increasing in all of their variables, i.e:

$$exp(V) = \sum_i \lambda_i v_i, \quad \lambda_i > 0, \forall i \tag{2.7}$$

When put in this form and assuming there are no cyclic assignment effects, the domain satisfies the requirements for strong monotonicity. Metric-FF does not run on problems which are not linear.

2.7.3 Building the planning graph and Extracting a Plan. The relaxed planning graph is constructed using a technique similar to that of the standard propositional case. In the standard case, each new level is initialized by copying the current level. Then all positive effects of actions whose preconditions are satisfied in the current level are added. For the numeric case "positive" refers to positive numeric effects, rather than propositional add effects. For simple assignments we take the greater of the current max value and the assignment value.

In the propositional case, planning graph growth terminates when the set of facts does not change from the previous level. In the numeric case, a cutoff value $mneed(v_i, s)$ is defined which is the maximum value required for all constraints involving v_i to be satisfied, given the values of the other variables in s. A virtual "fact" can be defined as the condition that $v_i \geq mneed(v_i, s)$ The termination condition is then identical to the propositional case, using facts and virtual facts.

Once the relaxed numeric planning graph is built, a relaxed plan is extracted from it as follows. We find a set of actions that asserts the goal facts in the final level, and use their preconditions as a set of sub-goals that must be achieved in the previous level. For variables, for every constraint, we find an action that adds some amount to the variable (remember we only have $\geq$, $>$ constraints). Again, we add the fact and constraint preconditions for the given action to the set of subgoals that must be achieved at the previous level.

2.7.4 State Domination. Any search algorithm must take care to ensure that the new states being examined have not already been visited. In the pure propositional case, this is straightforward. In the numeric case, a new state may be created which is identical to another already visited state, except some numeric variable is smaller. Under the assumption of monotonicity, such a state cannot lead to more solutions than the previously visited state. This motivates the following definition: a state s *dominates* a state s' if:

$$p_i(s) = p_i(s') \quad \forall p_i \in P$$

$$v_i(s) \geq v_i(s') \quad \forall v_i \in V$$

Where P is the set of propositions and V the set of numeric variables. With this definition in hand the search is expedited by skipping any state which is dominated by a previously visited state. Note that this condition requires numeric variables that have relevant inverted counterparts to be identical in the two states.

2.8 Causal Graph Heuristic Planning

A recent paper by Helmert [**9**] describes a new heuristic function, based on the concept of *causal graphs*. Unlike h^+, the causal graph (CG) heuristic does not attempt to achieve admissibility. Rather the purpose is to obtain good estimates of goal distance.

The CG heuristic applies to problems which are expressed in the SAS+ planning formalism. SAS+ is an alternative to STRIPS, where instead of propositions there are variables which can take on a certain discrete set $\mathcal{D}$ of values. Operator preconditions are expressed as $\{v : d\}$, meaning v must have value d. Effects are of the form $\{v : d \rightarrow d'\}$, meaning the operator changes v from value d to d'. Evidently the STRIPS distinction between add and delete effects no longer applies. It is a non-trivial to convert STRIPS problems to SAS+ problems, but this issue has been studied by several researchers[**40, 41**].

We can now define the **Domain Transition Graph** (DTG). This is a graph defined for a specific variable v in the problem. The nodes of this graph correspond to values $d \in \mathcal{D}_v$ the variable can take on, while an edge (d, d') exists if some operator has an effect $\{v : d \rightarrow d'\}$. The edges are labelled with the preconditions on other variables that the operator requires. Deferring for the moment the question of how to obtain edge costs, define $cost(d, d')$ as the shortest cost path from d to d' in the DTG, which can easily be calculated with Dijkstra's algorithm in time $O(|\mathcal{D}_v|)$.

The next definition is that of the **Causal Graph**. This graph has a node for each variable in the problem. An arc exists between two nodes (v_1, v_2) if some operator with an effect on v_2 has a precondition on v_1. An example of a Causal Graph from a Logistics problem is shown in Figure 2.6. In Logistics, all nodes representing trucks have arcs leading

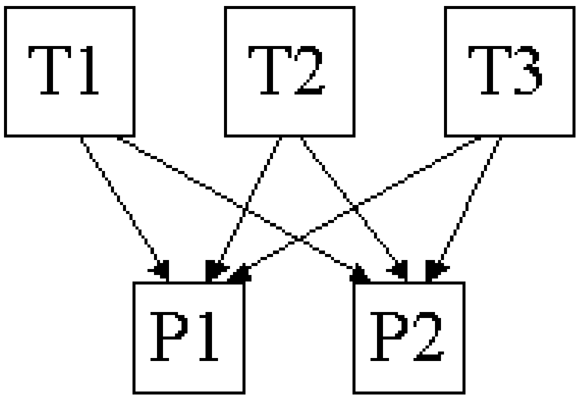

FIGURE 2.6. The Causal Graph for a Logistics problem. The Ti and Pi boxes represent truck location and package location variables respectively. The fact that the trucks have no incoming arcs shows that they can be moved independently of the other variables. The acyclic nature of the graph shows that Logistics problems can be solved easily.

to all variables representing packages. Importantly, the truck nodes have no incoming arcs. This implies that trucks can be moved independently of the other variables. In general, if the Causal Graph is acyclic and the Domain Transition Graphs are strongly connected, the problem can be solved easily.

Unfortunately, Causal Graphs will usually not be acyclic. For example in Blocks-World, there is an operator to place any block on any other block. Thus every Causal Graph node will be linked to every other node. Because of this we prune the graph by dropping arcs until it becomes acyclic. This simplification is analogous to the technique of ignoring delete effects in the relaxed plan heuristic. The resulting value will not give the exact distance to the goal, but hopefully still contains enough useful information to be used as a heuristic.

The simplified Causal Graph allows us to define the DTG edge costs. The cost of each transition is one plus the cost of achieving the preconditions of the operator that allows the transition, but only for those variables which are *above* the DTG variable in the pruned CG. Also, within the DTG traversal, a local copy of the relevant precondition variables is kept at each DTG node. The purpose of this technique is illustrated by Figure 2.7. Assume the state being evaluated is $(a = 1, b = 1)$. Then the $cost_b(1,3)$ will be $2 + cost_a(1,2) + cost_a(2,1)$. The full cost computation is illustrated in Figure 2.8.

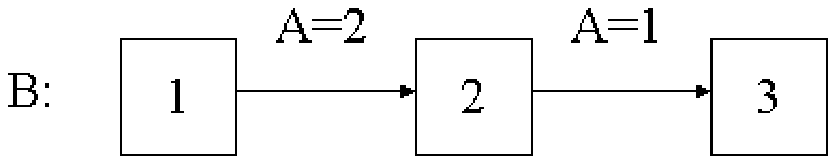

FIGURE 2.7. The use of local state information in the DTG traversal. The DTG for variable B is shown. If initially $A = 1$, then $cost_b(1,3)$ will include both $cost_a(1,2)$ and $cost_a(2,1)$.

```
function COMPUTE_COSTS($\Pi, s, v, d$)
  Let $V'$ be the set of immediate predecessors of $v$ in the pruned CG of $\Pi$.
  Let $DTG$ be the domain transition graph of $v$.
  $cost_v(d,d) := 0$
  $cost_v(d,d) := \infty$ for all $d' \in \mathcal{D}_v \setminus \{d\}$
  $local\_state_d := s$ restricted to $V'$
  $unreached := \mathcal{D}_v$
  while $unreached$ contains a value $d' \in \mathcal{D}_v$ with $cost_v(d,d') < \infty$ do
    Choose such a value $d' \in unreached$ minimizing $cost_v(d,d')$.
    $unreached := unreached \setminus \{d'\}$
    for each transition $t$ in $DTG$ leading from $d'$ to some $d'' \in unreached$ do
      $transition\_cost := 0$
      $cond\_set := cond(t)$
      for each pair $v' = e'$ in $cond\_set$ do
        $e := local\_state(v')$
        call COMPUTE_COSTS($\Pi, s, v', e$)
        $transition\_cost$ += $cost_{v'}(e,e')$
      end for
      if $cost_v(d,d') + transition\_cost < cost_v(d,d'')$ then
        $cost_v(d,d'') := cost_v(d,d') + transition\_cost$
        $local\_state_{d''} := local\_state_{d'}$
        for each pair $v' = e'$ in $cond\_set$ do
          $local\_state_{d''}(v') := e'$
        end for
      end if
    end for
  end while
```

FIGURE 2.8. Causal Graph heuristic computation. Notice that each value d has a corresponding local state of values for the CG predecessors of v. From [**23**]

The heuristic value of a state is just the computed cost of achieving each goal:

$$cg(s) = \sum_{v \in \mathcal{G}} cost_v(d(s), d_*) \tag{2.8}$$

Here $d(s)$ is the current value of v, and d_* is the goal value.

2.9 Robot Motion Planning

There is a wide literature on robot motion path planning [**17, 18, 19**]. Here several basic concepts and algorithms are discussed.

The path planning problem can be formulated as a search through a **configuration space**. This space contains all valid combinations of the robot's position and internal degrees of freedom (such as joints). Because joints are included, the configuration space can be high dimensional and non-Euclidean. Simple cubic obstacles in the three dimensional world can give rise to complex boundary surfaces in the configuration space, due to joint geometry.

Many standard path planning algorithms are *topological* in character. They form a graph representation of the topology of the configuration space, and then construct plans by a standard graph search. The basic algorithm is as follows:

- Construct a graph $\mathcal{G}$ representing the topology of the configuration space.
- Given start point s and goal g, find nearest neighbors $\bar{s}$ and $\bar{g}$ in $\mathcal{G}$,
- Attempt to connect s to $\bar{s}$ using a local planner,
- Find a path from $\bar{s}$ to $\bar{g}$ in $\mathcal{G}$,
- Connect $\bar{g}$ to g using a local planner.

Here the local planner is typically does something very simple, such as moving in a straight line, while checking for collisions. Because the local planner is simple, the topological graph must be quite comprehensive. Obviously then the main difficulty in the above algorithmic framework is to construct a good topological representation of the space. We now discuss several techniques to do this.

2.9.1 Deterministic Techniques. The technique called **Visibility graph planning** [**17**] is motivated by the following observation: in a world where the obstacles are polygonal, the shortest path between any two points will follow lines of visibility. Two points are mutually visible if the line between them does not intersect an obstacle.

The visibility graph $\mathcal{G}_v$ is constructed as follows. There is a node in the graph for each vertex in the obstacle set, and for the start and goal configurations for a given problem. An edge exists between two nodes if the corresponding points are mutually visible. There exist efficient algorithms to construct this graph which run in $O(N^2)$ time, where N is the total number of vertices of all the obstacles [**17**]. The visibility graph $\mathcal{G}_v$ is then used in the general algorithm described above.

There are two drawbacks to this approach. One is that the robot may be required to pass very near to the edges of the obstacles. In real-world situations, this can be dangerous. Another drawback is that in very high dimensional spaces, the number of obstacles can be enormous and the resulting visibility graph construction can be prohibitively expensive.

A second technique is based on the notion of *Voronoi diagrams*. The Voronoi diagram for a set $\mathcal{P}$ points $\{p_1, p_2 \ldots p_N\}$ is a partitioning of the space into a set of regions $\{v_1, v_2 \ldots v_N\}$ such that each v_i contains all points whose nearest neighbor in $\mathcal{P}$ is p_i. The *edges* of the Voronoi regions are the set of points which are equidistant from some $p_i, p_j \in \mathcal{P}$. By defining the region edges in terms of equidistance from elements of $\mathcal{P}$, the concept can be generalized to the case where $\mathcal{P}$ is a set of (obstacle) boundaries. The edges of the Voronoi regions form a topological graph $\mathcal{G}$, which can be searched as described above.

2.9.2 Probabilistic Techniques. Another important category of path planning algorithms involve probabilistic sampling of the configuration space. The two main techniques are called Rapidly-exploring Random Trees (RRTs) and Probabilistic Roadmaps (PRMs). We focus on the former, which inspires the STRIPS planning algorithm given in Chapter 3. Broadly, the idea of Chapter 3 is to use the expansion properties of RRTs to expand throughout the state space of large STRIPS problems.

The concept of Rapidly-exploring Random Trees was first introduced by LaValle [**42**]. They were used by Kuffner and LaValle [**20**] to solve multidimensional path planning problems. A good example of the kinds of problems that can be solved easily by RRTs is that of a humanoid robot picking up an object [**43**]. The robot has many degrees of freedom, and thus the planning problem is highly dimensional. The size of the search space is difficult for more traditional deterministic planners to deal with. However, solutions are abundant; there are many acceptable paths to the goal. Intuitively, RRTs allow us to find one solution out of many in high dimensional search spaces. Conversely, RRTs are not very good at finding a solution if only one exists. The growth process for RRTs is illustated in Figure 2.9.

The one parameter of this process is ϵ. This is the maximum distance towards q_{rand} that the local planner is allowed to move. This is to ensure that not too much time is spent attempting to reach any given point.

```
function EXPAND_RRT($s_{init}$)
   let $\mathcal{G} = \{s_{init}\}$
   while true do
      sample point $q_{rand}$ from configuration space
      let $q_{near} = \min_{q \in \mathcal{G}} dist(q, q_{rand})$
      let $q_{new}$ = LOCAL_CONNECT($q_{near}$, $q_{rand}$, $\epsilon$)
      INSERT_CHILD($\mathcal{G}, q_{near}, q_{new}$)
   end while
```

FIGURE 2.9. RRT expansion process. The local planner only extends a distance ϵ toward q_{rand}.

When the RRT is grown in this way, the following property holds. If the random points are chosen uniformly, then the tree expands to sample the configuration space uniformly. If the points are chosen according to some other distribution $p(x)$, then the tree will converge to a sampling of the space by $p(x)$ [**42**].

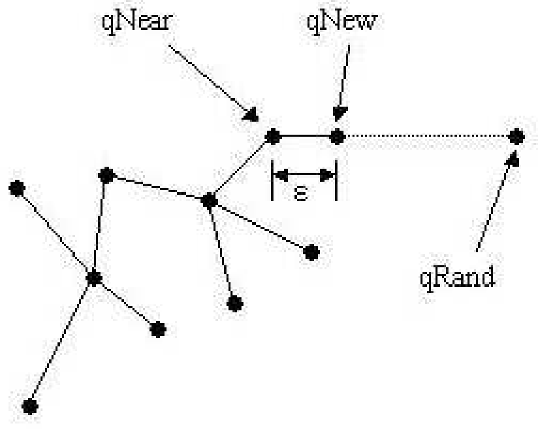

FIGURE 2.10. The RRT expansion process. qRand is the randomly sampled point, qNear is its nearest neighbor in the tree, and qNew is the new point added to the tree. The local planner only proceeds a distance ϵ toward qRand before stopping.

The RRT growth process can be slightly modified to produce a planning algorithm: after each growth phase, attempt to connect the new point q_{new} to the goal. Because of the coverage property, the algorithm is guaranteed to eventually expand into a region from which the goal is directly reachable. A further refinement uses another tree that grows outward from the goal.

A similar method to RRTs is the Probabilistic Roadmap [**44**] or PRM. The main difference here is that the PRM is intended to be built once and then reused several times.

Furthermore, once the PRM is built, paths can be generated very quickly. The PRM is constructed in two preprocessing phases, which are allowed to take a relatively long time. Also, the algorithm is designed to handle potentially complex spaces with many disconnected regions. Like RRTs, the focus is on finding any acceptable path, as opposed to finding an optimal or near-optimal path.

The PRM is a topological graph $\mathcal{G}$ which is constructed as follows. During the construction phase, points are chosen randomly from the configuration space. Given a random point p, a distance function is used to find set N_c of nearest neighbors in the PRM. Then a local planner is used to attempt to connect p to each $n \in N_c$. If the planner succeeds, an edge is added between p and n. Also, a neighbor n is skipped if it is already connected to p through the graph (from an edge that was added between p and a previous neighbor). Note that, unlike the case for RRTs, the point p is added to the PRM regardless of whether any connection attempt was successful.

After the PRM has been constructed, it is then "inflated". The point of the inflation process is improve the connectivity of the graph. A heuristic measure can be used to find nodes which seem to be in narrow passages of the space. Several possible heuristic measures can be defined; one example is to count the number of neighbor nodes lying within some distance of a given node. Once a node is selected by the heuristic, a new point is created nearby, and the planner attempts to connect this new point to the PRM. In this way, new points are added to narrow regions of the configuration space.

2.10 Convex Hulls

We now give a brief introduction to the idea of the convex hull. The notion of the convex hull is of widespread importance. It has been studied extensively, and many applications have been found [**45, 46**]. Chapter 4 presents a method for reducing the search space of a numeric planning problem that uses the convex hull of a list of points to represent a region of reachable numeric values.

2.10.1 Basic Concepts. Consider a set of points S in the d-dimensional Euclidean space E^d. The convex hull of S, denoted $conv(S)$, is the smallest convex set containing S. This will be a polyhedron, the vertices of which are elements of S. These vertices are called the **extreme points** of S, denoted $ext(S)$.

Given the extreme points $ext(S) = \{\bar{r_1}, \bar{r_2}, \ldots \bar{r_n}\}$, the set $conv(S)$ can be written as a convex combination of the $\bar{r_i}$:

$$conv(S) = \{\bar{x} : \bar{x} = \sum_{i=1}^{n} \lambda_i \bar{r_i}, \quad \sum_{i=1}^{n} \lambda_i = 1, \quad \lambda_i \geq 0\} \tag{2.9}$$

The problem of finding the extreme points of S has been extensively investigated, and linear time algorithms exist for two dimensions [**21**]. A related problem, that of finding the full description of the hull boundary (i.e., an ordering of the extreme points), can be reduced to a sorting problem in linear time and is therefore $\Omega(NlogN)$ [**21**].

2.10.2 Halfspace Representation. The planning technique presented in Chapter 4 uses the convex hull to represent a region of achievable numeric values. A key operation is updating this region to take into account new numeric constraints. This operation can be performed naturally by transforming the convex hull from the V-representation (specified by a set of points) to the H-representation (specified by a set of halfspace boundaries).

Consider a set H of halfspace boundaries in the d-dimensional Euclidean space E^d. The halfspace allowed by any given boundary is convex, and thus the intersection of all the halfspaces bounded by H is also convex. We call the set formed in this way a **convex polytope**. A classic theorem due to McMullen and Shepard establishes that if the space bounded by a convex polytope is finite, then it can be represented as the convex hull of a set of points [**21**]. The converse also holds: for any convex hull $conv(S)$ it is possible to find a set of halfspace boundaries that are equivalent to it. Converting between the two representations is a well established problem [**21**].

A halfspace boundary is said to *support* an extreme point $\bar{r}$ if $\bar{r}$ is on the boundary. An important relationship between the halfspace surfaces and the extreme points that equivalently define the hull is that, in a d-dimensional Euclidean space E^d, each extreme point is supported by at least d halfspace boundaries.

2.10.3 Gift-Wrapping Algorithm. Here we outline a simple method, called the Jarvis' March Algorithm [**47**], for calculating the convex hull in the plane. We choose this particular algorithm out of the many that exist both because of its simplicity and because it generalizes higher dimensions. The Jarvis' March Algorithm depends on the following theorem.

Theorem. Given a set of points S and two points $\{s, s' \in S\}$, the line segment l defined by $\{s, s'\}$ is an edge of the convex hull $conv(S)$ if and only if all other points in S lie on one side of it [**21**].

This theorem implies that, given a known point $\bar{s}$ of the convex hull[2] then the next point $\bar{s'}$ in the hull can be found by checking the line segments l defined by $\bar{s}$ and all possible choices of $\bar{s'}$. Note that in two dimensions, each vertex of the hull has exactly two edges. Therefore we may proceed ("march") around the hull by repeating the process on $\bar{s'}$.

The method of determining the correct subsequent hull point $\bar{s'}$ given $\bar{s}$ may be done as follows. Write the points $\{S \setminus s\}$ in polar form with $\bar{s}$ as the origin. Then $\bar{s'}$ is the point with lowest polar angle. This method allows $\bar{s'}$ to be found in linear time.

Roughly speaking, the above theorem generalizes to the higher dimensional case as follows. Instead of analyzing edges, we look at facets and subfacets. Facets are $d-1$ dimensional convex sets formed by $d-1$ points in S; subfacets are $d-2$ dimensional convex sets formed by $d-2$ points in S. Then given a known facet F of the hull, we may find adjacent facets by:

- Looking at a subfacet f,
- Considering facets F' formed by f and some $\bar{s'}$,
- Choosing F' such that the dot product of the normals to F and F' is maximal.

The main difficulty is now that each facet can have multiple neighbors. Thus we cannot march around the hull as in the planar case. For an efficient implementation, the lists of facets and subfacets under consideration must be carefully managed.

The complexity of the gift-wrapping method is given in terms of the number of facets and subfacets of the polytope under consideration. Let ϕ_{d-1} and ϕ_{d-2} be the number of facets and subfacets respectively. Then the complexity of the algorithm is $O(N\phi_{d-1}) + O(N\phi_{d-2})$. Since the number of facets of a d-dimensional polytope is $O(N^{\lfloor \frac{d}{2} \rfloor})$, the worst-case complexity is $O(N^{\lfloor \frac{d}{2} \rfloor + 1}) + O(N^{\lfloor \frac{d}{2} \rfloor} logN)$ [**21**]. Other methods exist for finding the convex hull in higher dimensions [**22, 48**], but they all face the same problem that the number of facets is exponential in d.

[2]An initial point can be found by taking the lexicographic minimum of S.

CHAPTER 3

RRT-Plan

3.1 Introduction

This chapter describes work on RRT-Plan, a randomized STRIPS planning algorithm inspired by RRTs. The basic premise is that for many STRIPS problens the state space is quite large, and therefore techniques based on searching the space can encounter difficulty. A mitigating factor is often present, however, which is that many solutions are possible. The goal is to translate the success of RRTs on path planning problems with large, complex spaces where solutions are abundant to discrete planning problems that are similarly large but unconstrained.

RRT-Plan uses the same conceptualization of the planning problem as the heuristic search planners described above. However, the method of exploring the space is quite different. RRT-Plan interleaves large scale stochastic exploration with deterministic local searches.

While heuristic search planners work well in many cases, they fail in many others. A significant reason for this failure is their reliance on deterministic procedures for both state evaluation and search. In particular, in many domains the heuristic function can give bad estimates, or can fail to adequately distinguish neighbouring states. By interleaving large scale stochastic exploration with deterministic local searches, the planner achieves a significant degree of robustness to heuristic breakdown.

The primary contribution of this chapter is a novel algorithm for solving STRIPS problems through the use of randomized search. We call this algorithm RRT-Plan. There are a variety of difficulties that arise with attempting to translate the RRT concepts from the continuous case to the discrete case. First, RRTs require the ability to randomly sample a state from the state space. While this is relatively straight-forward in continuous spaces, sampling feasible states can be challenging in discrete domains. Second, given a random state, RRTs require the ability to identify a "nearest neighbour" (in state space). Again, this has a straight-forward interpretation in continuous domains, but is less clear in discrete planning. Finally, a sub-planner must be invoked to try to connect the current state to the goal. This chapter discusses how RRT-Plan is able to overcome these problems specifically for the STRIPS representation, and argues that many of these ideas extend nicely to more general discrete planning problems.

In addition to presenting the core RRT-Plan algorithm, we present results validating performance of RRT-Plan compared to the state-of-the-art planners FF (on whose technology we rely significantly), and LPG [**6**]. We present several techniques which allow RRT-Plan to effectively focus on solving subgoals as well as to adapt its search parameters based on information from the growth of the tree. Finally, we describe problems that are particularly challenging for heuristic planners, and show that these can be easily overcome by RRT-Plan due to its randomized searching.

3.2 Motivation

Consider the situation of Figure 3.1. A truck starts out with two packages in the center of a chain of locations. It must deliver one package to each end of the chain. This simple example reveals an important failure of the relaxed plan heuristic. Moving the truck in either direction does not create a state with a better heuristic evaluation. The stumbling block is that the two goals in a sense compete with one another. We refer to such goals as **rival**.

Observe that this is basically a logistics problem, so from the results of Hoffmann [**38**] we would expect the relaxed plan heuristic to work very well. Indeed, if there were only one goal present, the relaxed plan would give perfect goal distance estimates. Building on this idea, we see that if the planner were required to achieve the goals in a particular order, each individual goal would be quite easy to achieve.

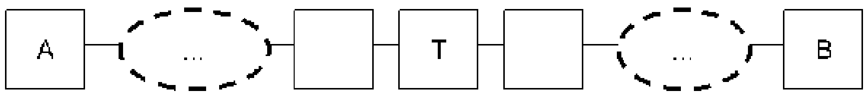

FIGURE 3.1. A simple example of rival goals. The truck T starts in the center and must deliver packages to locations A and B. Moving in either direction will leave the heuristic evaluation unchanged.

- Select random goal subset RGS
- Find Nearest Neighbor q_{NEAR}.
- Invoke local planner to connect q_{NEAR} to RGS.
- If state q_{NEW} is found that satisisfies RGS:
  - Add q_{NEW} to tree as child of q_{NEAR}.
  - Calculate atom costs for q_{NEW}.
  - Attempt to connect q_{NEW} to final goal.

FIGURE 3.2. An outline of the RRT-Plan algorithm.

Very often the problem of rival goals can be handled by imposing an artificial goal ordering. Also, in many cases there are many such orderings which are allowable (for example, in a typical Logistics problem the goals can be achieved in any order). One way of viewing RRT-Plan, the algorithm we present below, is that it conducts a search through the space of possible artificial goal orderings. If an allowable one is found, the relaxed plan heuristic gives much better estimates, and the search can be performed more quickly.

3.3 The RRT-Plan Algorithm

RRT-Plan is a randomized planning algorithm for discrete state spaces, in particular those that can be represented using the STRIPS planning language. The algorithm extends well-known heuristic planners such as HSP and FF, by introducing randomization in the exploration of the state space, therefore providing the ability to escaple plateaus in the heuristic function.

At a high level, RRT-Plan contains much the same steps as the RRT-Connect algorithm described above. The basic idea is to expand a tree over the discrete state space, in random directions (guided by the sampling of states), until a node is created that is sufficiently close to the goal that they can be connected by a short deterministic search. An outline is given in Figure 3.2.

There are several obstacles when trying to extend the concepts of RRTs to the discrete planning case. First, RRTs require the ability to randomly sample from the state space. Second, given a random state, RRTs require the ability to find its *nearest-neighbour* in the tree. Finally, a deterministic planner must be invoked as a sub-routine to try to connect nodes to random states and to the goal. We now discuss each step of our algorithm in detail.

Select random state (q_{rand}). The first step relies on the ability to sample points randomly from the space. While it is not difficult to sample randomly, it is practically impossible to sample the reachable space uniformly. This is because determining if any given state should be assigned a non-zero probability (i.e. determining if it is reachable) is equivalent to solving the planning problem itself.

Because of this difficulty, RRT-Plan generates target states q_{rand} by taking random subsets from the goal atoms. This approach has several advantages. It is easy to compute. If the problem is solvable, then every goal subset must itself be reachable. Finally, it tends to bias the search towards the goal. In the following we abbreviate “random goal subset” as RGS.

Some domains have natural goal orderings, meaning that some goal atoms must be achieved before others. The classic example of this is again Blocks World. Several methods exist to discover goal orderings [**49, 50**]; we use the heuristic technique of [**37**]. By utilizing this information, we can improve the selection of random goal subsets by respecting the ordering relationships between goal atoms. Specifically, no goal atom is included in the RGS unless all goal atoms which must be achieved before it are also included. While the ordering would eventually be discovered through random trial and error, using the computed goal agenda speeds up the process.

Find Nearest Neighbor. The second step in the expansion process requires finding the node in the tree that is closest to the random target. While this distance is well-defined in terms of the number of actions required to get from one state to the other, finding the precise value is again equivalent to the planning problem itself.

We therefore use a heuristic estimate of the distance between nodes. The h^+ heuristic can be used for this purpose. In particular we use the atom cost technique of HSP given by Equation 2.6. These costs only need to be calculated once per RRT node.

Note that it is possible that the nearest neighbor node in the tree actually contains the RGS target. In this case extra goal atoms are added to the RGS until it is no longer contained by the nearest neighbor.

Invoke sub-planner to connect q_{near} to q_{rand}. RRT-Plan requires a sub-planner at two steps - first to connect the nearest neighbor q_{near} to the target q_{rand}, and second to connect the new node q_{new} to the goal. For this purpose a modified version of FF is used.

Recall that FF features two stages of search: a fast phase of *Enforced Hill Climbing*, and a slow (but complete) phase of *Greedy Best First Search.* When applying FF as a subplanner in RRT-Plan, we only use only the first of these phases. In addition, we only expand a limited number of nodes. The search is restricted in these ways because the connection attempt might be impossible if q_{near} is a dead end, though significant effort is applied to prevent dead ends from being added to the tree (see below). If a certain point cannot be reached easily, we move on and do another expansion iteration.

When using FF to perform local search, the added goal deletion technique is turned off. A similar method called *goal subset locking* is used in RRT-Plan (see below). The helpful actions technique is used, but the stringency of action pruning is modulated as certain atoms begin to appear difficult to achieve.

Add q_{new} to tree. We maintain a simple tree structure in memory. When a goal subset q_{rand} is reached from q_{near}, a new node is added to the tree as a child of q_{near}, and the actions required to reach q_{rand} are stored. If the full goal is reached from a node in the tree, we can construct a full solution by following the path from the root to the node and then to the goal. In contrast to the continuous formulation, if the sub-planner fails to connect to the target, no new node is added to the tree.

Calculate atom costs for q_{new}. We use the HSP heuristic defined in Eqn 2.6 to calculate atom costs for any new node. These reflect the estimated cost of achieving the goal from state q_{new}. Note that these new costs reflect the fact that the goal subset atoms are now locked (see 3.3.1).

Attempt to connect q_{new} to goal. Again, we use the *Enforced Hill Climbing* phase of FF, with bounded node expansion. Normally the connection attempt does not succeed. However, the algorithm has invested time in finding a route to this new state, and should use the knowledge that the new state can be reached. Thus the best (smallest heuristic value) state discovered in the search is added to the tree.

Importantly, this allows is for the node expansion limit to be effectively bypassed. Consider the following scenario. A new RRT node is created, and then an attempt is made to reach the goal. This attempt fails because of the node expansion limit, but would have succeeded if it had been allowed to continue. Because the resulting state is also added as a new node, it can be selected as nearest neighbor on the next iteration, and the search can be continued from the point it was halted due to the expansion limit.

One common occurrence is as follows. The algorithm will perform a goal connection attempt, and make good progress, achieving many goal atoms and obtaining a low heuristic value. However due to some subtlety of the problem it will fail to achieve the goal. The best state is added to the tree, and on the following iteration it is selected as nearest neighbor. The search moves to the random goal state, which constitutes a slight detour but puts the goal in direct reach. A good example of this is the DriverLog domain. On the first goal attempt, all of the packages will be delivered to the correct locations, but getting the drivers and trucks to their destinations is more difficult. The tree continues to grow from this near goal state, and the problem is solved after a bit of trial and error.

3.3.1 Goal Subset Locking. Several of the problems encountered by h^+ planners are caused by the fact that the relaxed plan length heuristic does not penalize the deletion of goal atoms. If a goal atom can be achieved in an "easy" way through the deletion of an already asserted goal atom and in a "hard" way (in which other goal atoms are not deleted), the heuristic value is low, corresponding to the "easy" way. Furthermore, and perhaps more problematically, states which are closer to or further from the solution along the hard path are not accorded correspondingly better or worse values (see the discussion of the Push-Block domain below).

To avoid such situtations, when a RGS is achieved in the connection phase of RRT-Plan, the atoms of the RGS are *locked* so that any future action which deletes them is not considered. Additionally, any goal atoms which are locked in a parent node are also locked in its children. Importantly, this restriction is taken into account when calculating the atom cost estimates for the node (Eqn 2.6). States for which the final goal is accorded an infinite heuristic value are discarded. We define $h^+_{gsl}(s, gs)$ to be the length of the relaxed plan from

s to the goal where actions which delete atoms in goal subset gs are disallowed. It is clear that:

$$h^{+}(s) \leq h^{+}_{gsl}(s, gs) \quad \forall gs$$

This follows directly from the fact that the actions allowed in the goal subset locked relaxed plan are a subset of those allowed for the general relaxed plan. Reducing the number of available actions can only increase the length of the resulting plan.

3.3.2 Tree Growth Limits. When the tree is grown in this way, there is a limit on the total depth to which the tree can grow. Every time a new RGS is reached, two nodes are added to the tree - the new node which achieved the RGS, and the node representing the result of the goal connection attempt. Each new RGS node must have at least one additional goal atom more than its parent, because of Goal Subset Locking and the fact that we add more atoms to the RGS if the Nearest Neighbor already includes them all. Going from the root of the tree to the leaves, the number of achieved goal atoms must increase by at least one every other node. Thus the maximum depth of the tree is $2N$.

3.4 Adaptive Search Parameters

We can extract several important bits of information from the growth (or failure to grow) of the tree. This information is used to adapt the parameters of the sub-planner.

Three pieces of information are collected regarding tree growth. Each is the result of counting connection failures of the sub-planner. First, we keep track of how many times a given node has failed to connect to q_{rand} in the expansion phase. This counter is added to the distance function for a given node, so that a node with a large failure counter is selected less frequently as a nearest neighbor (eventually it is no longer selected at all).

The planner also keeps two similar counters for the goal atoms. These are incremented each time the atom is included in the RGS but cannot be reached. The sub-planner search can fail for two reasons: the node expansion limit is reached, or the list of states to expand is exhausted. We track the number of occurrences of both failure modes for each atom, and the parameters for the local search are chosen based on maximum failure counts for the atoms in the RGS.

When the algorithm has failed several times to reach an atom because of the node expansion tolerance, that parameter is simply increased. When the list of states has been exhausted, the remedy is to decrease the aggressiveness of action pruning. There are currently three levels of action pruning:

- TARGET_HA - consider only actions which are helpful in achieving the current RGS.
- FULL_HA - consider all actions which are helpful in achieving the full goal.
- ALL_APPLICABLE - consider all actions which are applicable in a given state.

Where we mean helpful actions in the sense of [**5**].

In the domains we have tested on, there is typically only one atom which is difficult to reach. For example in Blocks-World, the atom representing the blocks at the base of the tower is often very difficult to achieve, especially in a way such that $h^+_{gsl} < \infty$. As the tree expansion continues, RRT-Plan fails several times to reach that atom, until eventually the search parameters are sufficiently relaxed. After the atom is achieved, the goal can usually be reached rapidly.

Note here the contrast to FF. On such problems, FF will try the fast Enforced Hill Climbing phase, which will fail. Then the slow GBFS phase will run, and eventually succeed in achieving the critical atom in such a way that the full goal is now easy. But the algorithm remains in the slow phase, and the "good" state does not have a particularly low heuristic value, so many more node expansions are required to complete the search.

A precise and optimal strategy for relaxing the search constraints remains a topic for further work. In the following results the node expansion tolerance began at 500 and increased 10% per failure. The action pruning began with TARGET_HA, relaxed to FULL_HA after 10 failures, and relaxed to ALL_APPLICABLE after 50 failures. Given the fact that a sufficiently expansive search will eventually be conducted if necessary regardless of the initial settings of these parameters, we do not believe that the algorithm is particularly sensitive to the particular choice. Essentially no time was spent tuning the parameters.

3.5 Limitations of Standard Heuristic Search Techniques

Heuristic planners such as those described above (e.g. HSP and FF) use an approximation of the relaxed plan length as an estimate of the distance to the goal. FF uses a variety of additional techniques to prune the search space. These methods are generally quite effective, but are prone to failure in certain cases.

We now describe some of these situations with specific examples. Similar observations were made by Helmert [**9**]. We show how randomization can deal effectively with these issues. Broadly speaking, the issues that arise here involve the existence of **rival** goals in the problem. Two goals are rival if the most direct action leading to achieve one pushes the other farther away.

3.5.1 Issue with the h^+ heuristic. The h^+ heuristic function gives length of the relaxed plan from a state to the goal. The relaxed plan ignores the delete effects of actions. Since the goal cannot be achieved more rapidly by discarding delete effects, the h^+ function is *admissible.*

Unfortunately this heuristic is often somewhat uninformative. Indeed, in their original paper on HSP, Bonet and Geffner [**11**] argue that an approximation which is closer to a correct estimate of the true distance to the goal is more helpful than one which provides a strict lower bound. In any event, it often occurs that a large number of states are assigned an identical heuristic value. In the terminology of Hoffmann [**38**], this is called a *plateau.* When the search procedure reaches a plateau, it cannot make progress except by exhaustively examining states.

We can see an example of this in a modified logistics domain. Assume there is a long chain of locations, with one truck in the middle. The truck must deliver a different package to each end of the chain. In this case, the relaxed plan length heuristic assigns the same value to each state where one of the packages is not yet delivered. As a result, the heuristic is completely uninformative in terms of making progress towards either goal (in this case, the problem can still be solved because there are only as many states as links in the chain, but larger instances can be constructed with an exponential number of states). The two goals in this problem are rival.

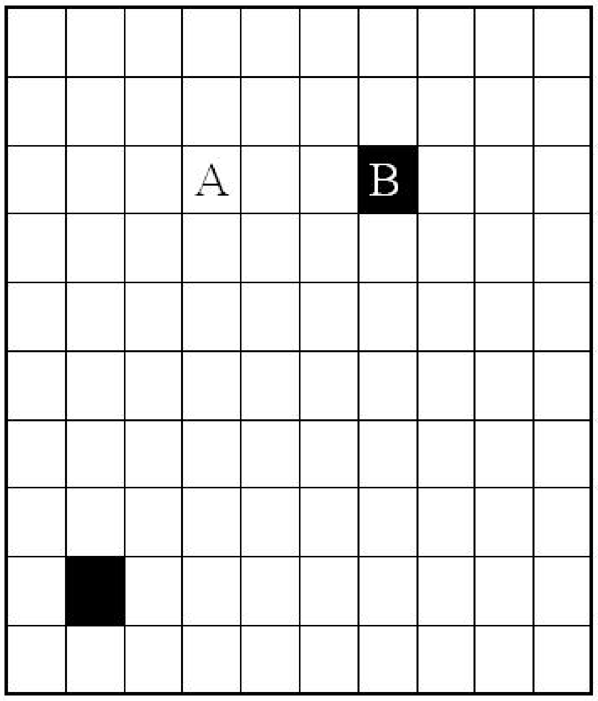

FIGURE 3.3. The Push-Block domain. The black squares are blocks. A and B denote locations to which the blocks must be pushed. Any state in which the lower block is closer or farther away from A has the same evaluation, because the relaxed plan simply moves the block from B to A.

In this domain, RRT-Plan will succeed immediately: one goal atom or the other will be randomly selected. Achieving either of the goal atoms (while ignoring the other) is easily done, and once the first is achieved, the second is also easy.

A related problem arises in a domain we have constructed called Push-Block. Here we have a set of blocks positioned on a regular grid. Blocks can be pushed horizontally and vertically, but only one block can occupy a location. There is only one meaningful predicate, *OCCUPIED*(X Y), which indicates whether a grid position has a block in it (there are other predicates which just encode adjacency relationships between rows and columns). The goal is a set of locations which must be *OCCUPIED*.

Consider the situation shown in Figure 3.3. The goal is to have blocks in positions A and B. Position B is already occupied and thus we need only move the block from the lower left corner to A. However, in the relaxed plan a short solution appears: move the block at B to A. Thus every state in which the lower block is nearer or closer to A has the same heuristic value of 3, which is the length of the relaxed plan.

If we ignored the block at B, the problem would be trivial. The h^+ heuristic would give a perfect evaluation of states, and the lower block would move directly to A. This is exactly what RRT-Plan does by locking the goal atom representing the block at B. In general, the more goals and blocks considered by the heuristic, the worse the quality of the evaluation provided.

From the two examples above we may make an interesting observation about RRT-Plan. By choosing goal subsets at random and locking goal atoms already achieved, the algorithm is effectively imposing an artificial goal ordering on the problem. If an artificial goal ordering is acceptable (i.e. the problem *can* be solved using it) then it has the effect of breaking a hard problem into a sequence of individually easy problems.

3.5.2 Issue with helpful action pruning. FF uses the notion of helpful actions to prune the search space during enforced hill climbing. Helpful actions are those which have add effects which are present in the first fact level of the solution to the relaxed planning problem. Intuitively, these are the actions which provide the most direct means of achieving the most pertinent atoms.

Hoffmann and Nebel [**5**] give an example where helpful actions can prune all of the solutions to the problem. Their example is somewhat abstract; we'll present a situation that arises in one of the actual planning domains, DriverLog (see Appendix B for a description).

DriverLog is basically the same as Logistics, except there are no airplanes, and the truck drivers can walk around as well as drive. Hoffmann [**38**] showed that FF can handle the logistics problems in polynomial time, so we might expect this domain to be similarly easy. However, consider the situation shown in figure 3.4.

Here a truck and driver are both at location A. Locations A and B are linked by a highway which can be driven across but not walked. A and B are also linked indirectly through C, and these paths can be walked by the driver. The goal is to have the truck at A, and the driver at B. The solution is clearly **WALK**(D1 A C) - **WALK**(D1 C B).

The relaxed planning graph will have only one level, because the first goal atom is achieved in the initial state and the second goal atom can be achieved with only one action. Thus the only helpful action considered is to drive from A to B.

RRT-Plan handles this case by randomly asserting AT(TRUCK A), and locking this atom when it is achieved. Now the relaxed plan solution will not be allowed to consider the

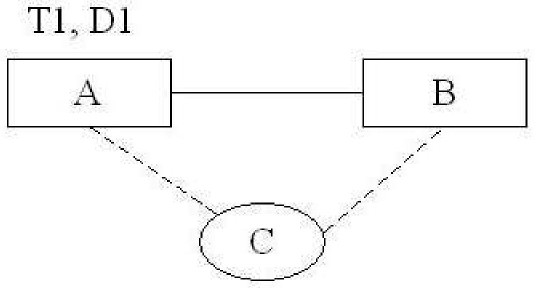

FIGURE 3.4. Problem encountered by FF in DriverLog domain. Truck and driver are both at A; goal is to move only the driver to B. Dotted lines indicate walking paths, solid line is unwalkable highway. The only action recognized as helpful by FF is to drive from A to B.

DRIVE action, and so will select the correct **WALK** action instead. Note that RRT-Plan has no way of knowing in advance that AT(TRUCK A) should be selected, it simply chooses it after several iterations of trial and error.

3.6 Experimental Results

To validate RRT-Plan, we compared its performance with that of well known planners FF and LPG [**6**] on problems from the planning competitions from 1998, 2000, and 2002 [**12, 13, 14**]. We also used the STRIPS version of the Pipesworld domain from the 2004 competition.

We also used the Push-Block domain described above. This is simply a 20x20 grid of positions which can be occupied by blocks. The blocks can be pushed left, right, up or down, but not into a location which already contains another block. The goal is simply a set of locations to which we must move blocks. We generated 20 domains, starting with one block/goal atom and moving up to 20 (see Appendix B for the PDDL definition of the domain).

Figure 3.5 shows the number of problems solved by the planners within a given time length. These statistics were generated by allowing the planners to run for up to five minutes and recording the time to completion. Table 3.1 shows the number of problems within a given domain which the planners could *not* solve within the time cutoff. Experiments were performed on a 3GHz Pentium 4 Linux machine with 2GB of RAM. Looking at Table 3.1,

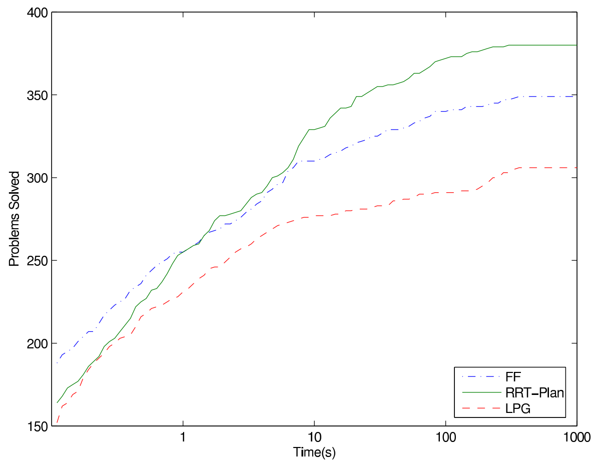

Figure 3.5. Number of plans solved as a function of time for FF, LPG and RRT-Plan. Time is on a logarithmic scale.

we see that RRT-Plan outperforms FF in several domains and is worse in only one, FreeCell. These results can be broken down as follows:

Rovers, Satellite, Logistics . In these domains, FF's fast Enforced Hill-Climbing phase is able to rapidly solve the problem. Because we use this same technique for the sub-planner, RRT-Plan will also rapidly solve the problem, after a small number of iterations. LPG also solves these problems easily.

Mystery, MPrime, FreeCell . The problems in these domains have a small number of goal atoms. This essentially cripples the randomization of RRT-Plan, because there are so few goal subsets to choose from. However, the search constraints will be relaxed until all the effort is expended by the sub-planner, thus RRT-Plan again becomes equivalent (modulo exact parameter settings) to FF. LPG does significantly better on the MPrime domain than both FF and RRT-Plan, and significantly worse on the FreeCell domain.

Blocks-World, DriverLog, Depot, Pipesworld, Push-Block. In these domains RRT-Plan seems to achieve consistently better results than FF. In the case of DriverLog and Push-Block, the reason should be clear from the discussion above.

For Depot and in particular Blocks-World, the cause is a bit more difficult to discern. These domains are characterized by strong goal orderings. However, the goal agenda technique used by FF [**37**] is able to discover the goal ordering and provide it to the planner. We speculate that FF does not consider the full goal when achieving goal subsets, which cripples the Goal Added Deletion heuristic. RRT-Plan will attempt to reach the first goal atom and reject several states as unacceptable ($h^{+}_{gsl} = \infty$). Thereafter the search constraints are relaxed, and finally the difficult atom is achieved acceptably, after which the problem becomes easy.

The goal of RRT-Plan is to find solutions to difficult problems. To that end, we were prepared to accept suboptimal plan lengths. We expected the plans generated to be uniformly worse than those created by other planners. Surprisingly, this was not always the case, as shown by Table 3.3. Indeed, for some problems the plan generated by RRT-Plan is about half the length of FF's. Figure 3.6 shows the number of plans generated of a given length.

RRT-Plan is of course a randomized algorithm and therefore can produce different results on different runs. While we have not performed exhaustive testing, the numbers in Table 3.1 change only slightly (in one test suite we observed only 6 failures on Pipesworld, in another 1 failure in Push-Block). This is probably because the average completion time for some problems is around the 5 minute cutoff and therefore variation can mean the problem sometime registers as a failure.

We have performed systematic tests on the hard DriverLog (16-20) problems. Table 3.2 shows the time required for the three planners on these problems. For RRT-Plan, the mean and standard deviation of the time required is shown for the hard problems. Table 3.3 shows the plan length of the solutions.

In comparing RRT-Plan to LPG, the most reasonable analysis is to look at the number of domains where one exceeds the other, rather than number of problems. With the exception of Push-Block, the score is seems to be tied: RRT-Plan wins in MPrime and loses in FreeCell, while in other domains performance is roughly equal. A greater number of problems and especially domains is needed to more rigorously compare performance of the planners.

Table 3.1. Performance of FF, RRT-Plan, and LPG on various domains. Entries list the number of problems the planner could not solve within five minutes of CPU time.

Domain	FF	RRT-Plan	LPG
Blocks World (35)	3	1	0
Driverlog (20)	5	0	0
Depot (22)	3	0	0
Freecell (80)	8	10	70
Logistics (63)	0	0	0
MPrime (35)	3	3	0
Mystery (30)	14	13	12
Pipesworld (50)	15	8	9
Rovers (20)	0	0	0
Satellite (20)	0	0	0
Push-Block (20)	15	0	19

Table 3.2. Time to completion (seconds) on DriverLog problems 10-20. For problems 16-20, mean and std dev are given for RRT-Plan.

Problem	FF	RRT-Plan	LPG
driverlog-11	0.00	0.00	0.05
driverlog-12	0.41	0.02	0.17
driverlog-13	0.17	0.05	0.47
driverlog-14	0.21	0.05	0.13
driverlog-15	0.06	0.09	0.18
driverlog-16	-	$\mu = 15.7$, $\sigma = 6.4$	274.79
driverlog-17	-	$\mu = 5.0$, $\sigma = 1.6$	2.14
driverlog-18	-	$\mu = 2.7$, $\sigma = 0.6$	38.34
driverlog-19	-	$\mu = 34.4$, $\sigma = 16.5$	215.41
driverlog-20	-	$\mu = 12.7$, $\sigma = 5.6$	9.77

Table 3.3. Plan Length for DriverLog problems 10-20. For problems 16-20, mean and std dev are given for RRT-Plan.

Problem	FF	RRT-Plan	LPG
driverlog-11	24	25	16
driverlog-12	51	48	40
driverlog-13	35	35	49
driverlog-14	37	50	41
driverlog-15	47	49	39
driverlog-16	-	$\mu = 144$, $\sigma = 17$	181
driverlog-17	-	$\mu = 114$, $\sigma = 4.6$	95
driverlog-18	-	$\mu = 107$, $\sigma = 5.3$	83
driverlog-19	-	$\mu = 191$, $\sigma = 15.2$	159
driverlog-20	-	$\mu = 143$, $\sigma = 2.9$	94

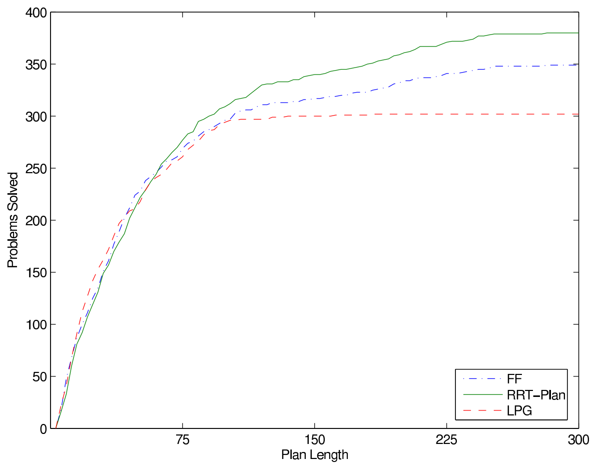

FIGURE 3.6. Plan length comparison for FF, LPG, and RRT-Plan.

3.7 Discussion

This chapter presents a randomized algorithm for STRIPs planning. The strong emphasis on randomization is conceptually novel, compared to current state-of-the-art discrete planners. The algorithm is shown to exhibits competitive empirical performance on a number of standard domains. The algorithm is inspired by Rapidly exploring Random Trees, a concept borrowed from the domain of continuous space path planning. However, significant alteration of the underlying RRT concept must be made for it to work in the discrete domain.

In general, one of the most important advantages of randomized over deterministic algorithms is that they avoid systematic errors. Heuristic planners such as FF use several techniques which are generally powerful and effective, but can sometimes fail completely even in simple situations. We have described a number of such situations above, and shown that our randomized algorithm provides better robustness in such cases.

The goal of the original RRT formulation is to expand the tree so that it reaches a uniform sampling of the configuration space. Our original idea was to try to achieve a similar thing for the discrete space search - to grow the tree until it samples the reachable states uniformly. However, this approach met with a variety of difficulties.

The basic problem in the analogy between the continuous and discrete cases is as follows. In the continuous case the region from which the sub-planner can achieve the goal directly is typically of the same dimension as the entire space. In the discrete case, the region can be exponentially smaller, depending of course on the nature of the sub-planner. Therefore even if we achieved uniform coverage of the space - which is itself quite difficult to do - the algorithm would still require exponential time and memory.

Instead of attempting to choose states randomly from the space, RRT-Plan instead randomly samples from goal subsets. As the number of achieved goal subsets in the tree grows, it moves closer and closer to the complete goal. In this sense, it is best to think of RRT-Plan as searching through the space of *artificial goal orderings.* With this reformulation in mind, we can make an interesting analogy. Continuous RRTs are highly effective in problems with high dimensionality but where solutions are not scarce, but encounter difficulty in problems where there is only a thin bundle of solutions (such as navigation in a narrow corridor). Similarly, RRT-Plan is effective at choosing an artificial goal ordering, in problems where there is no natural goal ordering. This artificial ordering can speed up the search significantly. RRT-Plan can also deal with problems with natural goal orderings, when those orderings are discovered using the method of [**37**]. We imagine that the algorithm will fare poorly in situations where natural goal orderings exist but are not discovered.

While the selection of random goal orderings is at the heart of our approach, several additional techniques are required. Most prominently is the idea of goal subset locking. By pointing the h^+ heuristic at a goal subset and preventing it from deleting already achieved goals, the topology of the search landscape is simplified. Local minima and plateaus which would otherwise hamper the planner's progress are removed. Also, as the tree grows certain goal atoms are identified as problematic, and more effort is devoted to achieving them.

Our results show RRT-Plan to be competitive with leading STRIPS planners. In particular, RRT-Plan can handle several types of problems which FF stumbles on, while performing only slightly worse than FF in other cases, such as when there are very few goal atoms. A main question for future work is how to extend RRT-Plan to deal with such cases.

The answer to this question will likely involve finding actions that assert the goal atom(s) and using the preconditions of those actions as a new set of goals, from which to randomly sample. This could even involve growing another RRT from the goal which expands in the regression space [**11**] of the problem.

CHAPTER 4

Heuristic Search Planning in Numeric Domains

One recent extension to the STRIPS planning formalism is the addition of numeric variables [**26**]. The benefits of this extension should be obvious: it is now possible to represent aspects of a problem such as the amount of fuel in a truck. In principle, complicated logical mechanisms can be used instead of numeric variables, but these mechanisms are awkward. As shown below, numeric variables have important properties that can be exploited.

We begin with a discussion of the problems encountered when attempting to apply heuristic search techniques to numeric planning problems. These general issues motivate the construction of a new planning technique called reduced search with enhanced states. A planner called CHCGP (described in Appendix A) is presented which uses this technique. Experimental results are given, which essentially show that there is a category of problems which traditional heuristic search planners cannot solve efficiently but CHCGP can, due to its use of reduced search.

4.1 Limitations of Heuristic Search in Numeric Domains

It is possible to make a rough analogy between heuristic search and the optimization technique of gradient descent. In both cases, the algorithm moves in the direction of decreasing value of a function (heuristic or objective).

Recall that gradient descent works roughly as follows. The goal is to minimize some objective function. A point is selected as a starting place. The gradient of the objective function is calculated at the point. Then a new point is found by moving in the direction of the gradient. This strategy will almost always succeed in finding a local minimum. The challenge is to efficiently find a *global* minimum.

The purpose of heuristic search is to find a zero heuristic value (goal) state. To move toward lower values of the heuristic function, states are removed from the open list in order of increasing heuristic value. As with gradient descent, the goal is to find a global minimum. Importantly, in both cases local minima tend to impede progress by drawing the search away from the true goal. The extent to which a local minima negatively impacts the search is related to the size of the surrounding "basin". This is simply the set of states that will lead to a given local minima[1].

Introducing numeric variables to the planning problem creates large basin local minima in two ways. First, the basins that exist in the propositional version become vastly larger if numeric variables are allowed (Curse of Affluence). Second, constrained resources create new broad-basin local minima. This is because all states in the search tree below an overconsumption mistake are dead ends. The Curse of Poverty prevents us from effectively determining if such overconsumption mistakes have been made until the resulting search tree has grown quite deep.

The difficulty of finding lower bounds on resource consumption, the cause of the Curse of Poverty, is well-known in the literature. In general, however, because of the general intractability of numeric planning and the diversity of planning systems, little effort has been made to analyze specific causes of difficulty and their effect on heuristic search planners. In some ways, the curses described below can be considered features of a numeric domain which make a basically simple task intractable by currrent heuristic search methods. Because of the essential simplicity of the task, it is reasonable to imagine that more advanced techniques will overcome the obstacle. Below, we show this to be true in certain cases for the Curse of Affluence. The Curse of Poverty is less amenable to improved methods, as discussed in Appendix A.

4.2 Curse of Affluence

In Chapter 3, several failure modes of the h^+ heuristic were noted. A clear example is the Logistics problem with a linked set of states as shown in Figure 4.1. Here we have

[1]Note that basins are not intrinsic properties of the space but are rather defined relative to the search mechanism being used.

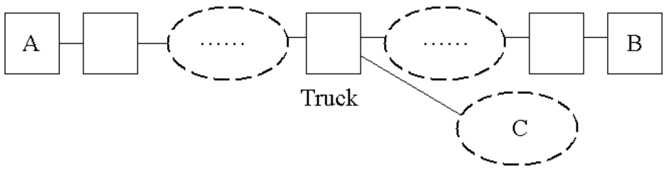

Figure 4.1. A Linked-State Logistics Problem. The truck begins near the middle of the chain, and must deliver packages to A and B. However, it has only one package, so first it must travel to region C and obtain another one. This will only be done after exhaustive search, because h^+ evaluation does not assign one-package states a sufficiently high heuristic value.

a truck in the middle of a long chain of locations. The truck must deliver packages to locations on either end of the chain. However, it has only one package, and in order to obtain an additional package it must travel outside of the chain to the region marked C. Unfortunately, h^+ will not recognize this necessity, because it disregards delete effects in its evaluation process [**11**] (see Section 2.5). Thus the planning system will exhaustively search all of the states in the chain before returning to region C to obtain the additional package. In this case the remedy is not terrible, because the number of states which must be searched is small (on the order of the number of locations in the chain).

However, consider what happens when a fuel variable is added to the problem. Assume each move from location to location requires one unit of fuel, but the truck can refuel at each location.

Because of this added dimension, the number of states that must be searched has now become vastly larger. In principle, each state in the non-metric problem now corresponds to an infinite number of states in the metric problem, if the fuel variable has no maximum. If the only piece of information the planner has at its disposal is the heuristic estimate, there is no way to verify that one state is identical to another except for some irrelevant amount of extra fuel, because there is no way to verify that the extra fuel really is irrelevant (it *might* be the exact amount necessary to achieve the goal). This is the Curse of Affluence: the possibly infinite explosion in the number of states caused by incrementally larger resource quantities (affluence), which do not bring the goal any closer. This is shown in Figure 4.2.

An interesting variant to this problem occurs when dead ends are caused by limitations of the numeric variables (as opposed to the discrete variables, such as those linked to the

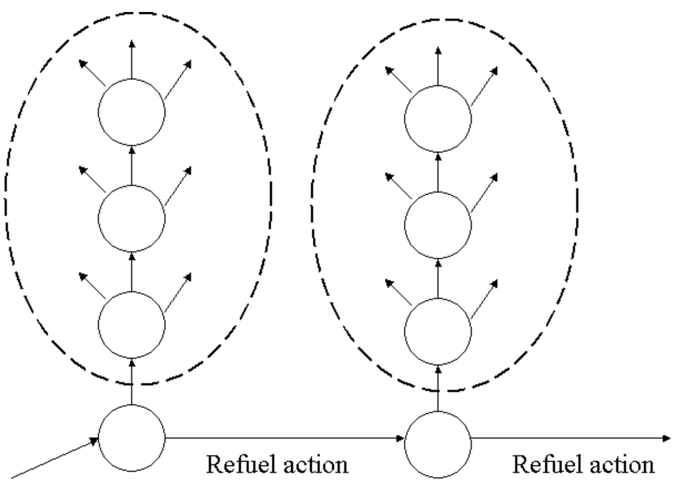

FIGURE 4.2. The Curse of Affluence. Circles represent states, arcs represent operators leading to new states, and dotted ovals represent regions which are identical except for a different fuel value. The initial state is a dead end, but it and the region above it are assigned incorrectly low heuristic value, and so are searched exhaustively before further progress is possible. This is can usually be done, but if a numeric effect is applicable, it creates a new region which is identical except for having more of an irrelevant resource. This region must now also be searched. In principle, an infinite number of such regions can be created by successive applications of the refuel operator. Another possibility is if the initial state is not a dead end, but many refuel actions are required; here the goal is located in an oval region far to the right.

packages). For example, imagine that the truck starts in region C and has the two necessary packages, but not enough fuel to reach both A and B. Additional fuel can be obtained in C, and so the question now becomes: how much fuel to add? A standard heuristic search will attempt to deliver the packages, fail, add more fuel, search and fail again, *etc* until the necessary amount of fuel is added. In Figure 4.2, this would correspond to a situation where the goal state is in one of the regions far to the right of the starting point (ie, after many refuel actions).

It is important to note that the failure modes described above are not unique to the relaxed planning heuristic h^+. Any heuristic which gives bad estimates for some states will experience breakdown when those states are inflated into large regions by the addition of numeric variables to the problem.

4.3 Curse of Poverty

A wide number of numeric planning problems involve resources that must be conserved. An ideal planner would be able to discover how much of a given resource is required to reach the goal. States which did not meet this resource requirement would be pruned from the search. In that way, the search would be confined to regions from which the goal can actually be reached.

In reality, it is very difficult to find such lower bounds on resource consumption. This is primarily because, even in simple problems, the relevant lower bounds are not on specific variables, but on combinations of those variables. This is illustrated by the Two Rovers problem discussed below.

Thus, the planner can be in a basin leading to a local minima, and have no real way to know about it. Because resource conservation is typically a factor that must be considered from the outset of the planning process, a mistake made early in the search can cause the planner to explore large regions of states which are all dead-ends. This is illustrated in Figure 4.3.

Two Rovers Problem Consider the following simple example given by Figure 4.4. The goal is to go to location D, obtain a soil sample, and return to A. It costs 10 fuel to get from A to D and back. In the first case, imagine that there is one Rover, with 15 fuel. Then, assuming a lower bound on the amount of fuel required to the reach the goal can be found, this state can be pruned because there is insufficient fuel. This lower bound on a specific variable could be found using a variation of the HSP atom cost fixed-point calculation ([**11**], see Section 2.5).

To see the problem, consider the case that there are two Rovers at A. Then, the lower bound for *both* fuel variables considered individually is zero. This is because either Rover is capable of completing the task. In this case, the appropriate lower bound is on the *maximum* of the two fuel variables. In principle, that could also be found, but there is no obvious way to decide in advance that that particular combination of variables is the critical one. Indeed, if Rover2 were already at D, then the appropriate lower bound would not be on $max(f_1, f_2)$ but rather on $max(f_1, f_2 + 10)$.

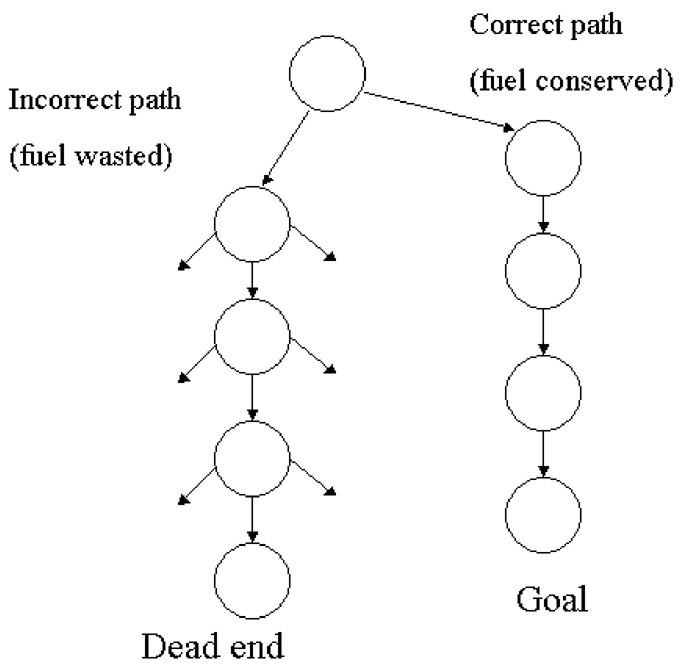

FIGURE 4.3. Constrained resource problems create broad basin local minima. Circles represent states, and arcs represent operators leading to new states, not all of which are shown. The planner initially follows the path on the left, but has made a mistake early on by failing to conserve fuel. This mistake typically cannot be detected until much later. This creates a very broad basin leading to a dead end. This basin must be searched exhaustively before returning to the point at which the mistake was made.

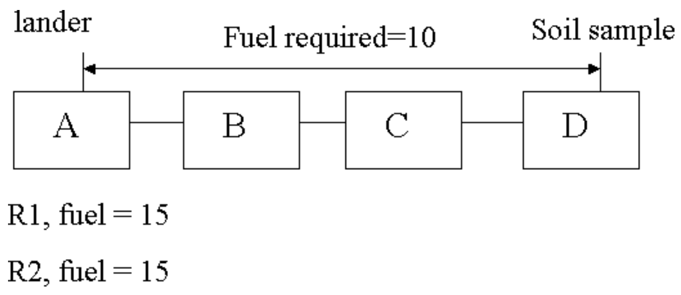

FIGURE 4.4. The Two Rovers problem. Boxes represent locations; the goal is to sample the soil at location D and return to the lander at A. 20 fuel is required to get from A to D and back. With one rover, a lower bound of 20 on the available fuel can be found, and a state with 15 fuel can be pruned. With two rovers, the relevant lower bound is on the maximum of the two fuel variables.

4.4 Definitions and Assumptions

In this work we use the numeric planning formalism of PDDL 2.1, as described in Section 2.1.3. The main restriction is that only linear expressions will be considered. Thus every expression in the numeric state variables $expr(V)$ can be written as:

$$expr(V) = \sum_{i=1}^{N} w_i v_i + C \tag{4.1}$$

Here C represents a constant value, while the w_i represent variable coefficients. With this definition, numeric constraints can be simplified to expressions that must be greater than zero:

$$expr(V) \geq 0 \tag{4.2}$$

This definition can be extended to include comparison operations such as $\{>, \leq, <, =\}$. A numeric effect is a pair $\{v_i, expr(V)\}$ such that in the resulting state:

$$v_i' = v_i + expr(V) \tag{4.3}$$

By appropriate choice of the w_i in *expr*, this definition can take into account the three PDDL operations INCREASE (+=), DECREASE (-=) and ASSIGN (:=).

In many domains of interest, variables can only take on values that lie between certain extrema. These extrema can be found as follows. If for every grounded operator $o \in \mathcal{O}$ that decreases the value of a variable by an amount N, there is a precondition which requires that the prior value of the resource is at least $N + M$, the minimum value of the variable is M. In principle, N can be some expression $N = expr(V)$, but it is generally a constant value. In most domains, variables have minimum values of zero. An analogous technique can be used to find maximum values.

Given the above definitions and observations, we now define a concept that will be of critical importance to this work.

Definition: **Pure Translational Operator**. An operator $o \in \mathcal{O}$ is pure translational if the following conditions are met:

- o has no discrete effects;
- o has only constant numeric effects;
- the only numeric constraints of o are those which enforce resource extrema[2].

[2]This requirement may be relaxed. However, it seems quite natural, and it simplifies the exposition.

The essential intuition behind the idea of a translational operator is that once the discrete preconditions have been met, it can be applied many times. An example of a translational operator is the **RECHARGE** action in the Rovers domain [**14**]. The Settlers domain [**14**] also exhibits many pure translational operators.

The techniques of reduced search discussed below rely on the existence of pure translational operators in the domain. The planner will still run if these operators do not exist, but it will not realize any improvement over standard heuristic search. See Section 4.10 for a discussion of why pure translational operators occur naturally in many domains.

A further definition which will be important is that of **individual monotonicity**. A variable v_i is individually monotonic if

- There is no expression $expr(V)$ in the domain for which $w_i < 0$, ie v_i only participates positively in constraints and effects.
- For all effects eff with expressions $expr(V)$ such that $w_i \neq 0$, $targ(eff)$ is monotonic.

This definition is very similar to Hoffmann's [**39**]. The difference is that Hoffmann defined monotonicity to be a property of a domain, while we define it to be a property of a variable. Our definition allows monotonicity to be exploited when it is present, but the techniques described below do not require it.

Note that the above properties of variable extrema, pure translational operators, and individual monotonicity can be discovered in a preprocessing step. The complexity of the preprocessing step is linear in the number of grounded operators in the problem.

4.5 Reduced Search with Enhanced States

In the most general terms, reduced search is a search in which certain operators are left out. The operators to remove from the search are chosen because they have simple effects. Because of this simplicity, it becomes possible to construct enhanced states. These are *regions* in the full state space that can be reached by inserting the simple operators into the reduced plan when appropriate. This is illustrated in Figure 4.5. Consider the following definitions.

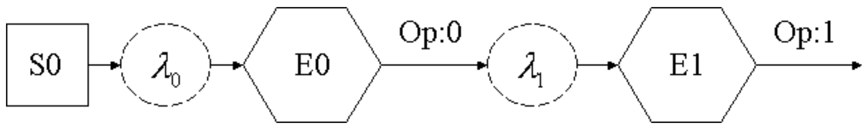

FIGURE 4.5. The reduced search process. The λ_i show points at which an arbitrary set of applicable simple operators may be applied, to construct corresponding enhanced states, shown as hexagons. S0 is the initial state, EO is the set of states which can be reached from S0 solely through the use of simple operators applicable in S0. E1 is the set of states reachable from first epoch of simple operators λ_0, the first general operator (denoted Op:0), and the second epoch of simple operators λ_1.

Definition **reduced plan**: Given a full plan $P = \{u_1, u_2 \ldots u_t\}$ and a set of "simple" operators O_s, the reduced plan P_r is $\{P \setminus O_s\}$, i.e. P with the simple operators removed.

Definition **enhanced state**: Given a reduced plan P_r leading from the initial state s_i, the enhanced state $s_e = result(s_i, P_r)$ is the set of all states s that can be obtained by inserting simple operators into P_r such that the resulting full plan is valid.

The above ideas are not useful without a clear identification of which operators will be considered "simple". This choice must be made bearing in mind that an overly broad identification will cause the enhanced states to become very difficult to represent and maintain. We must also redefine the conditions for operator applicability, and the results of general operators, when dealing with enhanced states. Assuming this can be done, the benefits should be clear: the space of the reduced search is simply much smaller because fewer operators are considered.

For the purposes of this work, pure translation operators as defined in Section 4 are considered simple. This allows the enhanced states to be represented compactly. The discrete variables are constant, while a convex region is used to represent the values that numeric variables can take on. Discrete preconditions and effects of general operators are unchanged, while numeric constraints and effects can be handled naturally. In the following the term **reachable space**, denoted $\mathcal{R}$, is used to distinguish between numeric regions and the enhanced states with which they are associated.

For a simple example of when the reduced search strategy can help, consider the Rovers domain. Here the RECHARGE operator is pure translational, and is thus left out of the search. When a state is reached in which a rover can be recharged, an enhanced state is constructed containing a representation of all values the energy variable can achieve, through repeated RECHARGE actions. Thus it is not necessary to apply RECHARGE during the actual search. In terms of Figure 4.2, the regions enclosed in dotted lines (and all regions further to the right, not shown in the figure) are compressed together. If a goal state is found as a descendent of the recharge state, the postprocessing step goes through the reduced plan and inserts a sufficient number of RECHARGE actions to ensure that subsequent energy requirements are met.

4.5.1 A Convex Hull Representation of Reachable Space.

This representation of the reachable space $\mathcal{R}$ must satisfy several requirements. It must be easily queriable in order to determine operator applicability. Also, it is clear that the reachable spaces can vary significantly from one enhanced state to the next. Numeric effects can change the space, and numeric constraints can cut out regions. Finally the space must be expanded when new pure translational operators become applicable.

The way that we have defined pure translational operators ensures that the discrete variables in enhanced states have constant values. Numeric variables are allowed to have multiple values, due to the multiple possible insertions of pure translational operators in the reduced plan leading to the enhanced state.

Because of this, and because of the assumption that numeric expressions are linear, a natural choice for the representation of $\mathcal{R}$ is the convex hull. For every enhanced state s_e a list of points $R = \{\bar{r_1}, \bar{r_2}, \dots \bar{r_w}\}$ is maintained. The reachable space is then approximated as the convex set of the R:

$$\mathcal{R} = conv(R) = \{\bar{x} : \bar{x} = \sum_{i=1}^{w} \lambda_i \bar{r_i}, \quad \sum_{i=1}^{w} \lambda_i = 1 \quad \lambda_i \geq 0\} \tag{4.4}$$

It is important to note that this is an approximation. The real reachable space is a countable subset of points within this region. In the domains we have experimented on, the approximation has been sufficiently accurate that the postprocessing step (see Section 4.6) can always find valid plans. Note the distinction between R (a list of points) and $\mathcal{R}$ (the convex set of R).

Now that we have decided on our representation, it is necessary to answer the following questions. Given an enhanced state s_e, an associated reachable space $\mathcal{R}$, and a general operator o_g with numeric constraints C and effects E:

(i) How do we determine if some point in $\mathcal{R}$ satisfies C?

(ii) How does $\mathcal{R}$ change based on both C and E?

(iii) How do we construct the new reachable space $\mathcal{R}'$ which results from the application of o_g to s_e?

Note that the third question is strictly more difficult than the second. This is because new enhanced state has different sets of pure translational operators which are available, and the potential effects of which must be factored into the new reachable space. These questions are answered in the following subsections.

In the following we refer to the complexity of calculating the extreme points of a set of N points in E^d as $CH(N, d)$. This complexity depends on the exact algorithm being used but is on the order of $N^{\frac{d}{2}}$. See Section 2.10 for more information. In our current implementation we use the LRS algorithm by Avis [**22**].

4.5.2 Querying the Hull. A convenient property of the convex hull reachable space representation is given by the following Lemma.

Lemma: Let $\mathcal{R}' = conv(R')$ be the convex hull formed by applying a constraint c to the convex hull $\mathcal{R}$. Then every new extreme point

$$\bar{r} \in R', \bar{r} \notin R$$

is created by the intersection of c with an edge of *conv*(R) [**21**].

This property of convex hulls are exploited in the function QUERY_HULL, shown in Figure 4.6. Here $C = \{c_1, c_2 \ldots c_n\}$ is the set of constraints we wish to know if the hull satisfies. This function proceeds by implicitly calculating the revised hull after each recursive constraint application.

The INTERSECT function calculates the intersection of a line segment with a constraint surface. The complexity of the overall algorithm is $O(|R|^{2|C|})$, as each recursive call can potentially require the creation of $O(|R|^2)$ intersection points. In the domains we have experimented with, $|C|$ is at most 3. The Settlers domain of IPC3 [**14**], a rather complicated numeric domain, the maximum value of $|C|$ is 2. Care is taken to minimize the number of queries that must be made: checks for operator applicability analyze discrete preconditions first, and only if all such preconditions are satisfied is the above function called. It is also worth observing that if $|C| = 1$ then the complexity of the function is in fact $O(|R|)$.

```
function QUERY_HULL($R, C$)
  if $|C| = 0$ then
    return true
  end if
  let $SAT = \{\bar{r} \in R : \bar{r} \models c_1\}$
  if $|SAT| = 0$ then
    return false
  end if
  let $UNS = \{R \setminus SAT\}$
  let $INT = \emptyset$
  for each pair $\bar{r_{sat}} \in SAT, \bar{r_{uns}} \in UNS$ do
    let $\bar{p_{int}} = INTERSECT(\overline{r_{sat} r_{uns}}, c_1)$
    add $\bar{p_{int}}$ to $INT$
  end for
  return QUERY_HULL($SAT \cup INT, \{c_2, c_3 \ldots c_n\}$)
```

FIGURE 4.6. The QUERY_HULL function. Determines if some point in $conv(R)$ satisfies a set of constraints C.

Note that the sets $SAT \cup INT$ can include more than just the extreme points of each revised hull. This is because we do not know which pairs of points in R are connected by edges in the hull. It is possible to obtain this information, and doing so may be an interesting optimization to the algorithm.

4.5.3 Updating the Hull.

Typically, the pure translational operators are not the only operators that involve numeric variables. General operators, which must be handled using the standard search procedure, may have numeric effects and constraints. When creating new states as the result of these operators, the reachable space from the predecessor state is copied and then updated to take into account of both the constraints on and effects of these operators.

When an operator with some numeric constraints is applied to an enhanced state, the new reachable space is constructed by (potentially) cutting out some region of the previous $\mathcal{R}$. This simply takes into account the fact that if some operator has a constraint $v_1+v_2 > 4$, and the numeric effects of the operator do not change v_1 or v_2, then the new reachable space $\mathcal{R}'$ includes no points for which the constraint does not hold.

In order to update the hull by applying a set of constraints, a simple variation of the QUERY_HULL algorithm is used. Notice that at every step of the algorithm, the set $SAT \cup INT$ is guaranteed to contain every extreme point of the revised hull. Therefore, we

```
function EXPAND_HULL($\mathcal{R}$, $o$)
  let $\bar{v}$ represent translation due to operator $o_{pt}$
  for each point $\bar{p} \in ext(\mathcal{R})$ do
    calculate $\lambda_{max}(\bar{v}, \bar{p})$
    create new point $\bar{p_n} = \bar{p} + \lambda_{max}\bar{v}$
    add $\bar{p}, \bar{p_n}$ to set $\mathcal{A}$
  end for
  set $\mathcal{R} := ext(\mathcal{A})$
```

FIGURE 4.7. The EXPAND_HULL algorithm.

simply recalculate the extreme points of the set $SAT \cup INT$ that are obtained after applying the final constraint c_n. The complexity of this process in the worst case is $O(CH(|R|^{2|C|}, d))$.

After the new hull taking the constraints into account has been computed, the operator's numeric effects are applied. This process is quite simple. We simply evaluate the effects at every point:

$$\bar{r_i'} = \bar{r_i} + eff(\bar{r_i}) \tag{4.5}$$

4.5.4 Expanding the Hull. After updating the hull based on the result of a general operator, we must now expand it to account for the new pure translational operators that may have become applicable. In the Rovers example above, this happens when the rover enters a state where the recharge action is applicable.

To begin, we explain the process for expanding the hull as a result of just one translational operator. This is done by the EXPAND_HULL function shown in Figure 4.7.

The complexity of this operation is $O(CH(|R|^{2|C|}, d))$. The calculation of λ_{max} is done simply by finding the intersection between the ray extending from $\bar{p}$ in the direction given by $\bar{v}$ with the boundary values for the problem. These are just the extrema for the variables, or some maximum allowable value that should be greater than the interesting ranges of values in the problem. The final step is a recomputation of the extreme points of the convex hull, which is done to remove extraneous points.

This technique allows us to expand the hull to take into account the effect of multiple applications of a single pure translational operator. We now consider how to proceed when many such operators are applicable.

```
function EXPAND_ALL($s, s_{prev}$)
  let set $O_{pt}$ be all pure translational operators applicable in $s$ but not $s_{prev}$
  let $\mathcal{R} = \emptyset$
  let $\mathcal{R}' = reachable(s)$
  while $\mathcal{R} \subset \mathcal{R}'$ do
    let $\mathcal{R} = copy(\mathcal{R}')$
    for each operator $o \in O_{pt}$ do
      call EXPAND_HULL($\mathcal{R}', o$)
      for each operator $o' \in amset(o)$ do
        insert $o'$ into $O_{pt}$
      end for
    end for
  end while
```

Essentially what we will do is to order the pure translational operators in a list and iterate through it, applying the EXPAND_HULL routine for each one. The process terminates when the hull ceases to grow. A simple refinement is to start with only those operators which were not applicable in the previous state, and to consider new operators if their preconditions are modified by the others. This can reduce the number of EXPAND_HULL calls that are necessary.

The notation $amset(o_{pt})$ denotes the set of pure translational operators whose constraint variables are changed by o_{pt}. The requirement that $\mathcal{R} \subset \mathcal{R}'$ simply means that if the hull has not grown in one iteration, it is guaranteed to never grow in further iterations and so the process terminates.

Conceptually, this process is quite similar to the construction of the planning graph ([**33**], see Section 2.3). Like the planning graph, the reachable space is guaranteed to grow at every iteration. Because each iteration depends only on the previous iteration, when the space ceases to grow during one step it will never grow in any later step.

4.6 Plan Post-Processing

The final stage of the reduced search comes when an enhanced state is found which satisfies all of the goal conditions. We must now go over the reduced plan and insert pure translational operators when appropriate to ensure that the full plan is correct. In the following, two things should be noted. First, because we have approximated the reachable space as a compact region rather than a discrete lattice, it is possible for this process to fail. In this case the planner simply resumes the search. Second, while some of the following steps can be expensive, they are small compared to the cost associated with the search itself.

Given a reduced plan $\mathcal{P}_r = \{o_1, o_2 \dots o_m\}$, the goal of the post-processing step is to construct valid *derivations* for the points in the reachable space of the final enhanced state. A derivation is a set of integers that represent the number of times a pure translational operator is applied in every enhanced state:

$$\Lambda = \{\lambda_1^1, \lambda_2^1, \dots \lambda_n^1; \lambda_1^2, \lambda_2^2, \dots \lambda_n^2; \dots \lambda_1^m, \lambda_2^m, \dots \lambda_n^m\}$$

Here n is the number of pure translational operators in the problem and m is the number of general operators in the reduced plan. The element λ_i^j denotes the number of times the i-th pure translational operator is applied between general operators o_{j-1} and o_j in the reduced plan (or prior to first general operator o_1).

A reduced plan $\mathcal{P}_r$ and a derivation Λ uniquely specify a sequence of points $\{\bar{p_0}, \bar{p_1}, \bar{p_2} \dots \bar{p_m}\}$ such that $\bar{p_0}$ represents the values in the initial state, and each subsequent point is the result of applying both the pure translational operators in stage j and the general numeric effect eff_j of operator o_j in the reduced plan:

$$\bar{p_j} = eff_j(\bar{p_{j-1}} + \sum_{i=1}^{n} \lambda_i^j \bar{v_i}) \tag{4.6}$$

Here again the $\bar{v_i}$ represent the effects associated with the i-th pure translational operator. A derivation for a point $\bar{q}$ in the reachable space of the enhanced state following general operator o_j will be called valid if the numeric constraints for every general operator $o_k, k \leq j$ are satisfied by previous point modified appropriately:

$$(\bar{p_{k-1}} + \sum_{i=1}^{n} \lambda_i^{k-1} \bar{v_i}) \models const(o_k) \tag{4.7}$$

We also require that $\bar{q} = \bar{p_j}$, which simply means that the derivation must actually *produce* the point to which it is attached.

The basic strategy used to construct derivations is to go through the same process of updating, expanding, and applying constraints to the hull as described above, except now ensuring that every point in the reachable spaces has a valid derivation. The derivations at each step are used to obtain the derivations for the subsequent step.

The hull expansion process is done in the same way as described above, with one additional detail. Now when a new point $\bar{p'}$ is created, a copy of the derivation of the original $\bar{p}$ is made, and the λ_{max} value is added. Clearly if all points in $\mathcal{R}$ have valid derivations at the beginning of an EXPAND_HULL call, the derivations of new points (with the addition of λ_{max}) will also be valid. Applying the numeric effects of general operators is done without changing the derivations.

The main complexity of the process comes when the hull must be modified to take a new constraint into account. When expanding the hull, new extreme points are created by applying pure transitional operators to old extreme points, so it is easy to update the derivations. When applying constraints, new extreme points arise as the result of intersections between the hull and the constraint surface, so the derivations cannot be obtained in an obvious way.

Let us first simplify the problem by assuming that the λ_i^j can take on continuous values. If this were the case, then a point satisfying the applied constraint could be found in the following way. Given a segment between two points $\bar{a}, \bar{b}$ that do and do not satisfy the constraint, respectively, write the point given by the intersection of the segment and the constraint as:

$$\bar{p} = \bar{a} + \phi(\bar{b} - \bar{a}) \qquad 0 \leq \phi \leq 1 \tag{4.8}$$

Then if the Λ_a, Λ_b are the derivations associated with $\bar{a}, \bar{b}$, the derivation of Λ_p is simply:

$$\Lambda_p = \Lambda_a + \phi(\Lambda_b - \Lambda_a) \tag{4.9}$$

It follows by induction and linearity of effects that when a new derivation is created which is a convex sum of two other derivations, as in Equation 4.9, all of the points in the new sequence will be convex sums of the corresponding points in the original derivations. That is,

$$\bar{p_j} = \bar{a_j} + \phi(\bar{b_j} - \bar{a_j}) \tag{4.10}$$

Thus, if the original derivations Λ_a and Λ_b are valid, the new derivation Λ_p produced by Equation 4.9 is also valid. It produces the correct point $\bar{p}$. Also, it satisfies all upstream numeric constraints. This can be seen by the fact that the $\{\bar{p_j}\}$ are formed by convex sums of the $\{\bar{a_j}\}$ and $\{\bar{b_j}\}$, and both of those sequences satisfy the relevant constraints. Therefore if the λ_i^j could take on non-integer values, the problem would now be solved.

4.6.1 Rounding Hypercubes. To deal with the requirement that the λ_i^j be integral we simply round the values $\phi(\Lambda_b - \Lambda_a)$. However, it is necessary to consider all possible rounding choices. This is because hull points are by definition located on constraint surfaces, and so minor changes in their derivation can cause them to become invalid. The function ROUNDING_HYPERCUBE generates all derivations that are possible by making different rounding choices for epoch/operator pairs i, j such that ${}^a\lambda_i^j \neq {}^b\lambda_i^j$. This is not a hypercube in the numeric space of the problem but rather in the space of derivations.

```
**function** ROUNDING_HYPERCUBE($\Lambda_a, \Lambda_b, k$)
  let derivation set $\mathcal{D} = \{\Lambda_a\}$
  **for each** pair $i$, $j$ such that ${}^a\lambda_i^j \neq {}^b\lambda_i^j$ **do**
    let derivation set $\mathcal{E} = \emptyset$
    **for each** $\Lambda_d \in \mathcal{D}$ **do**
      $\Lambda_e = copy(\Lambda_d)$
      ${}^e\lambda_i^j$ += $\lfloor \phi({}^a\lambda_i^j - {}^b\lambda_i^j) \rfloor$
      ${}^d\lambda_i^j$ += $\lceil \phi({}^a\lambda_i^j - {}^b\lambda_i^j) \rceil$
      add $\Lambda_e$ to $\mathcal{E}$
    **end for**
    add $\mathcal{E}$ to $\mathcal{D}$
  **end for**
```

Once the final set of derivations $\mathcal{D}$ is obtained, each one is examined to ensure that it satisfies all constraints in the reduced plan, including the new one that necessitates the process. If several possible derivations remain, the one which is furthest from the point $\bar{a}$ is chosen. Finally, the resulting point is recalculated to take into account the rounding (i.e. the new $\bar{p}$ is modified to agree with its derivation, not Equation 4.8).

4.6.2 Partial Optimality. The basic purpose of the post processing algorithm is to construct derivations Λ for each extreme point in the reachable space of the final (goal-satisfying) enhanced state. Now that the derivations have been found, the planner is free to select the best point according to whatever goal metric has been given for the problem. If the goal metric is a linear sum of the variables, one of the extreme points in R will be optimal. Thus we have a guarantee of partial optimality. Given a reduced plan, the post processing algorithm can select the optimal set of pure translational operators that are consistent with ensuring plan validity.

4.7 Exploiting Monotonicity

The techniques described above do not require that the underlying variables be monotonic. However, in problems that exhibit monotonicity, this property can be exploited. In order to clarify how to do so, consider the following general definition.

Definition **State Domination**: a state s dominates a state s', denoted $s \rhd s'$, if every plan which is valid starting from s is also valid starting from s'.

In the context of numeric planning, the following condition on s and s' implies that $s \rhd s'$:

$$v_i(s) = v_i(s') \qquad \forall v_i \in P \tag{4.11}$$

$$v_j(s) \leq v_j(s') \qquad \forall v_j \in V_m \tag{4.12}$$

Here the set P is the set of all discrete variables and non-monotonic numeric variables, while V_m is the set of monotonic numeric variables. This is the technique used by Hoffmann in his paper on Metric-FF [**39**] (see Section 2.7). However, he did not address the possibility of having mixed monotonic and non-monotonic variables.

We will use this property in two ways. First, we can discard points in the representation of the reachable space that will never contribute to a solution. Second, when we come to states in the search tree that have already been explored, we can bypass them (described below). Note that two identical states dominate each other, so this idea is a generalization of the standard technique of ignoring states in the closed list that have already been expanded.

In general an enhanced state includes many states, some of which may dominate others. If the state corresponding to some $\bar{r'} \in \mathcal{R}$ dominates some other $\bar{r_m} \in R$, then we can safely discard $\bar{r_m}$. This can be determined by analyzing the constraint surfaces that support each extreme point. For a convex hull in d dimensions, each extreme point $\bar{r_m}$ must be supported by at least d constraint surfaces $supp(\bar{r_m})$. A constraint surface S can be expressed by the following inequality:

$$\sum_{j=1}^{d} w_j v_j + C > 0 \tag{4.13}$$

We can now state the following Lemma.

Lemma: Let j be the dimension corresponding to some numeric variable v_j, and let $supp(\bar{r_m}) = \{S^1, S^2 \ldots S^{d'}\}$ be the constraint surfaces[3] supporting the point $\bar{r_m}$. Then if the following condition on the hyperplane coefficients w holds:

$$w_j^b \geq 0, \quad b = \{1, 2, \ldots d'\}$$

there exists a point

$$\bar{r'} \subseteq conv(R \setminus \bar{r_m}) \quad : \quad \bar{r'} \rhd \bar{r_m}$$

This point may or may not itself be a member of R.

Proof : Let $\bar{r'} = \bar{r_m} + \gamma v_j$. This point must satisfy all of the supporting surfaces $supp(\bar{r_m})$, since by assumption moving in the positive j direction can only contribute to the left hand sides of Inequality 4.13. Also, by the definition of monotonicity, increasing the value of v_j in a given state cannot invalidate any plans leading from that state. Thus $\bar{r'} \rhd \bar{r_m}$. Finally, the value of γ can be increased until the new point intersects some other constraint surface of the hull. The point of intersection will be on the boundary of the hull, which implies that it is a convex combination of the other $\{R \setminus \bar{r_m}\}$. $\square$

If a dominated point $\bar{r_m}$ exists in the hull, any generic point $\bar{p} \subseteq conv(R)$ can be replaced with a new point $\bar{p'}$:

$$\bar{p} = \sum_i \lambda_i \bar{a}_i \qquad \sum_i \lambda_i = 1 \qquad \lambda_i \geq 0 \tag{4.14}$$

$$\bar{p'} = \sum_{i \neq m} \lambda_i \bar{r_i} + \lambda_m \bar{r'} \qquad \sum_i \lambda_i = 1 \qquad \lambda_i \geq 0 \tag{4.15}$$

And it must be the case that $\bar{p'} \rhd \bar{p}$. Therefore we may safely discard the point $\bar{r_m}$ from our representation of the reachable space of the state, and in so doing we will not discard any solutions. This technique can significantly reduce the number of points required to

[3]note that $d' \geq d$.

represent the reachable numeric space of an enhanced state. For convenience we make the following definition:

Definition **Relevant Hull**: for a given set of monotonic variables V_m, the relevant points of R are the extreme points of R that are supported from the above in every monotonic dimension, i.e. the set of supporting hyperplanes includes a coefficient $w_j < 0$ for all $j : v_j \in V_m$. The relevant hull (denoted $conv_r$) is the convex set of the relevant points.

The intuition for this is simply that only the upper bounds for monotonic variables are worth taking into account. These upper bounds are caused by constraint surfaces with $w_j < 0$. In the Rovers domain, the energy value the rover has when it enters a state where RECHARGE is applicable does not matter; only the maximum energy value it can achieve is relevant. Note that this modified definition of the hull is used in both the expansion function EXPAND_ALL described above, and in the closed list checking scheme described below.

Enhanced State Closed List

An essential component of heuristic search is maintaining a list (the closed list) of states which have already been visited. When a new state is created, it is checked against the states in the closed list. If the new state is dominated by some state in the closed list, it should not be expanded. For discrete variables it makes sense to check if every variable has the same value in both states.

In numeric planning, this process becomes more difficult. It is possible to check states exactly, but since the numeric variables can in principle take on an infinite range of values, the benefit of checking the closed list will be much reduced.

Fortunately, the definition of state domination given above extends naturally to the case of enhanced states. An enhanced state s_e dominates s'_e if the discrete variables are identical, and

$$conv_r(R) = conv_r(R \cup R') \Longleftrightarrow \mathcal{R} \supseteq \mathcal{R}' \tag{4.16}$$

Where $\mathcal{R}$ and $\mathcal{R}'$ are the reachable spaces associated with s_e and s'_e respectively. As indicated this can easily be checked by adding together all the points in R and R' and recomputing the hull. In this way the criterion for discarding states because they have already been explored becomes much more general and so many more enhanced states can be safely discarded.

4.8 Full Planning System

The concepts relating to reduced search with enhanced states presented above constitute the main contribution of this Chapter. In order to incorporate these ideas into a full planning system, we have developed the Convex Hull Causal Graph Planer (CHCGP). This system performs a reduced search, using a modified form of the Causal Graph heuristic [**23**]. It is important to note that the reduced search technique can be used in conjunction with *any* heuristic function that can be modified to take into account numeric constraints and effects. Thus it would be possible, for example, to develop a version of Metric-FF that uses reduced search. The implementation of CHCGP is described in Appendix A.

4.9 Experimental Results

In this section we briefly compare CHCGP to Metric-FF. We compare our planner to Metric-FF for several reasons. While it is probably not considered the state of the art, the planners that outperform it on numeric domains use a different formulation of the problem (LPG [**51**]) or are hybrid systems such as SGPlan [**16**]. SGPlan was the top performer in the most recent International Planning Competition [**15**] and uses Metric-FF as a subplanner.

When the Curse of Affluence as described above is relevant to a problem, and when the techniques described in this chapter are applicable, CHCGP performs much better than Metric-FF. These domains were constructed to be as simple as possible and to illustrate the advantage that CHCGP enjoys over other heuristic search planners. We do not argue that the problems are particularly realistic, but certain aspects of real-world domains will bear some resemblance to the types of situations described here.

We show results which illustrate the validity of our technique in certain cases. We do not show results on the standard domains for two reasons. Our current implementation of CHCGP is not highly optimized (techniques such as helpful actions and caching remain to be implemented). For the most part, the standard domains do not satisfy the conditions for reduced search to be useful. One exception is Rovers, where reduced search simply means increasing the fuel variable to its maximum value when a state is entered where the recharge action is applicable. This simple technique helps a surprising amount. The second exception is Settlers. Here the problem is that there are simply too many numeric variables, so convex hull computations will grind to a halt.

See Appendix B for a more detailed description of the domains discussed in this section, including domain file PDDL.

4.9.1 T-Logistics. This is the problem described in Figure 4.1. There is a set of locations in the form of a "T". The single truck starts at the intersection of the T. The goal is to deliver packages to both edges of the T. There are two identical packages: one starts in the truck, and the other is waiting at the base of the T. The truck requires fuel to move, and fuel can only be obtained at the initial location. The height of the T is fixed at 10, while the width is a parameter.

Figure 4.8 shows the performance of Metric-FF and CHCGP on this domain, as the width of the top of the "T" is increased. This problem illustrates the Curse of Affluence. In order for the problem to be solved, the truck must move to the base of the T to obtain the second package. However, these states have a lower heuristic values than the states at the top. The addition of the fuel variable creates an unlimited number of low heuristic states at the top of the T.

For CHCGP the domain is just like standard logistics. All states which are identical except for different fuel values are compressed into a single enhanced state.

4.9.2 Toll Chain. In this domain a car moves through a series of links in a chain. To move across each link, a resource "toll" must be paid. The toll is simply one unit of each resource. The goal is simply to get to the end of the chain. The difficult aspect of this problem is that the resources can be produced only at the first location of the chain. Two parameters can be used for this domain: the length of the chain (shown), and the number of resources, fixed at three in these experiments.

Figure 4.9 shows the performance of Metric-FF and CHCGP on the Toll Chain domain, as the length of the chain is increased. CHCGP does very well. At the beginning of the search, it creates a large enhanced state representing all values that could be reached by various production actions. At every step, the enhanced state is updated to take into account

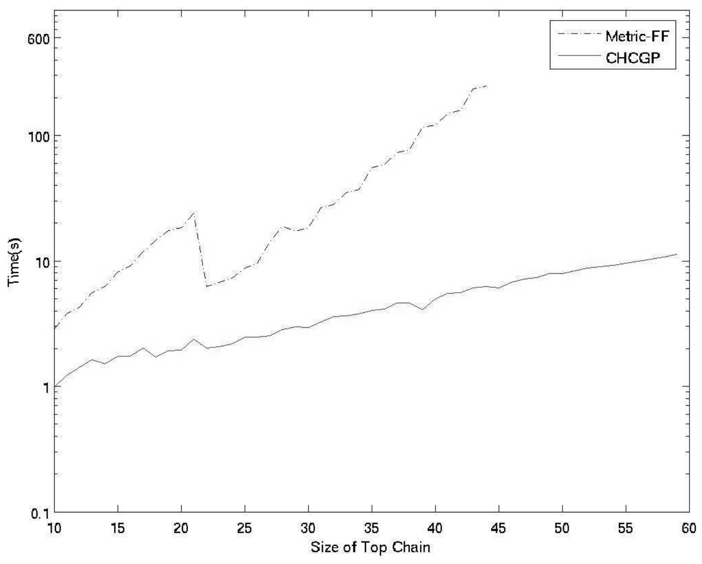

FIGURE 4.8. Performance curves for Metric-FF and CHCGP on the T Logistics problem. The parameter being modified is the width of the upper segment of the T.

the cost of the toll. Thus the search space is compressed into a number of enhanced states equal to the length of the chain.

Metric-FF, on the other hand, never knows how many resources should be obtained in advance, before leaving the initial location. It must search across the chain until it reaches a toll it cannot pay, then go all the way back up the chain.

4.9.3 Shortcut Trap. This domain again considers a chain of states. A "goal-silo" location exists at every other link in the chain. Each goal silo requires a certain number of resources to satisfy, and the goal of the problem is to satisfy the requirements for *one* goal silo. Resources can be produced, but after producing a resource the agent cannot move any further (in other words, it cannot produce some amount of a resource, then realize the amount is insufficient and move on). Also, there is a limit on how many resources can be produced. The resource requirements for goal silos are chosen randomly, but in such a way that the total required value is fixed. The first goal silo has a total requirement of 50, and the requirement for every subsequent silo is one less.

We call this domain "shortcut trap" because if the agent tries to take a shortcut by satisfying the requirements for an early goal silo, for which the resource requirement exceeds

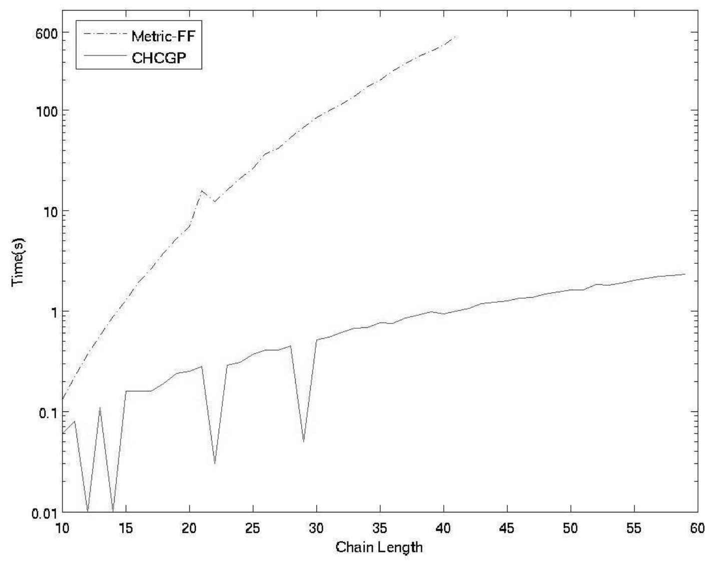

FIGURE 4.9. Performance curves for Metric-FF and CHCGP on the Toll Chain problem, as the length of the chain is increased.

the production limit, it will fall into a large dead end region (a trap). Figure 4.10 shows the performance of Metric-FF and CHCGP on this domain, as the production limit changes (note the right side of the graph shows low production limits).

CHCGP performs quite well on this domain. Because the production actions are pure translational, the dead end regions are collapsed into a single enhanced state. Therefore it moves along the chain, checking if the requirements for each goal silo can be satisfied, until it finds one that can be.

The performance of Metric-FF is quite interesting and can be understood as follows. On the easy problems, where the production limit is high, the optimizations of FF allow it to find a goal quickly. The seemingly hard problems with a low production limit are actually easier for Metric-FF, because the size of the dead end trap regions are much smaller. Therefore Metric-FF does the worst on the middle problems, where each dead-end region is very large. It should be noted that if the number of resource types is increased, Metric-FF breaks down badly. This is for a similar reason: the size of the dead end regions increase exponentially with more variables.

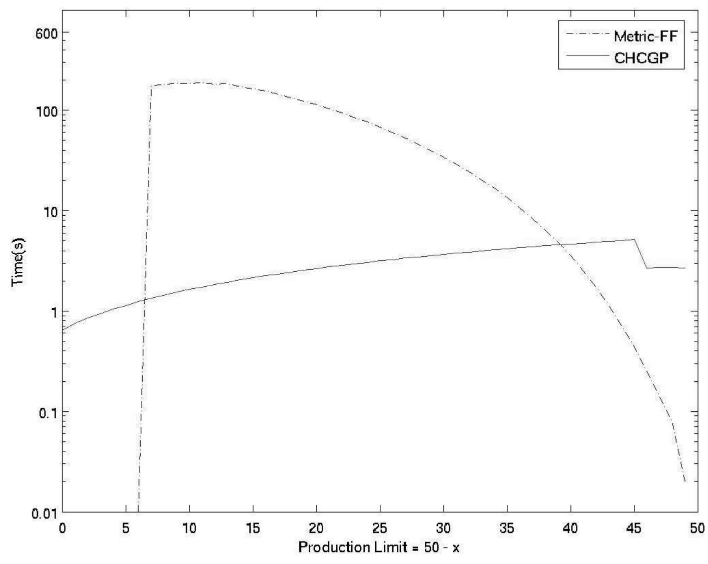

FIGURE 4.10. Performance curves for Metric-FF and CHCGP on the Shortcut Trap problem, as the production limit is decreased. Note that low production limit values are on the right of the graph.

4.9.4 Accumulation. In this domain there are several types of resources which can be traded evenly for one another. The initial state 120 total units distributed equally between N types (for these experiments N=4). The goal is to activate a set of goal flags, which can be done by achieving a certain threshold of one resource (the threshold varies in our experiments). Once the necessary level of some resource has been achieved and the goal flag set, it can be traded away for the next resource and so on until all the goal flags are obtained. The numeric version of h^+ does not reward states which have accumulated a large amount of one commodity at the expense of the others. Essentially this domain is a numeric version of the linked state logistics problem, where the heuristic does not discriminate between states along the chain.

The domain is quite easy in reality. CHCGP solves it after a number of state expansions equal to the number of variables in the domain. Metric-FF solves the problem quickly after accumulating the required amount of a given resource; the issue is just getting to that point. It can succeed on instances with small numbers of variables.

4.9.5 Pressurized Blocks World. Pressurized Blocks World is a simple variant of the standard domain. Each block has an associated variable called "pressure". The pressure

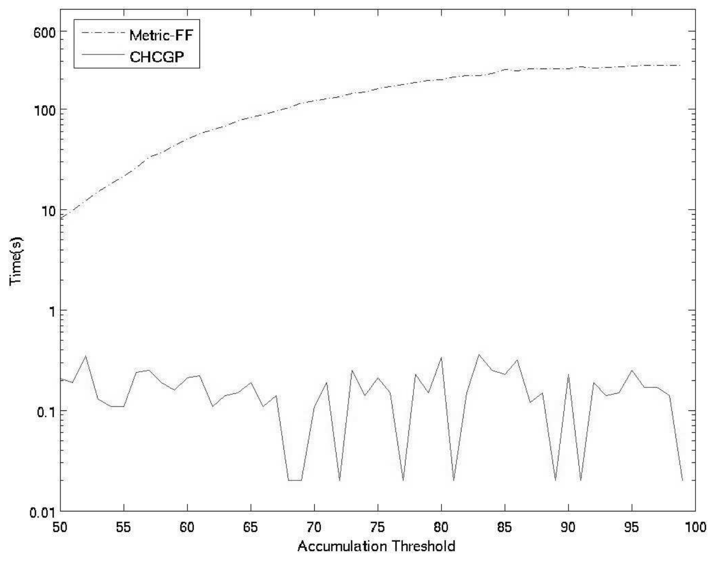

FIGURE 4.11. Performance curves for Metric-FF and CHCGP on the Accumulation problem as the threshold for goal flag achievement is increased.

variables all start at zero and can be increased (but not decreased) if the block is on the table and clear. To place one block on top of another, the one block must have greater pressure than the other. There are thus two natural variants. In the "simple" version of the problem, the higher block must have greater pressure. In the "hard" version, the lower block must have greater pressure. The hard version is hard because in order to successfully create a tower of blocks, the agent must reason about the future and pressurize a base block sufficiently to allow many other blocks to be stacked above it.

Unfortunately a comparison is not currently possible due to an implementation bug in Metric-FF. Note that in this domain the variables are not monotonic, and so inverted variables must be introduced (see Section 2.7). Additionally, Metric-FF's state domination scheme will only be able to prune a state if it is exactly identical to other states in the closed list.

On this domain, the limiting factor for CHCGP is the complexity of convex hull computations large numbers of blocks. The problem includes one numeric variable per block, so the run time starts to increase dramatically for problems with about 8 blocks. It is likely that refined techniques can decrease the required number of convex hull operations;

see Section 4.10 below. In Tables 4.1 and 4.2 we have listed the times to completion for the problems CHCGP was able to solve within 5 minutes. 13 out of 35 problems in the standard set were solved (note that the problem files have the same initial state and goals as those in the standard set - the only difference is the addition of pressure functions).

TABLE 4.1. Time to completion and number of hull points in final enhanced state for solved Simple Pressurized Blocks-World problems.

Problem	Time(s)	# Hull Pts
blocks4-1	0.12	5
blocks4-2	0.11	5
blocks5-0	0.24	6
blocks5-1	0.19	6
blocks5-2	0.57	6
blocks6-0	3.71	7
blocks6-1	0.35	7
blocks6-2	1.05	7
blocks7-0	4.36	8
blocks7-1	2.66	8
blocks7-2	5.31	8
blocks8-0	85.56	9
blocks8-1	3.76	9

TABLE 4.2. Time to completion and number of hull points in final enhanced state for solved Hard Pressurized Blocks-World problems.

Problem	Time(s)	# Hull Pts
blocks4-1	0.17	5
blocks4-2	0.16	5
blocks5-0	0.36	6
blocks5-1	1.15	6
blocks5-2	1.81	6
blocks6-0	5.22	7
blocks6-1	5.25	7
blocks6-2	6.58	7
blocks7-0	62.03	8
blocks7-2	7.64	8

4.10 Discussion

Planning in numeric domains raises a variety of issues, not all of which can be solved. In this Chapter we have identified two specific problems, the Curse of Poverty and the Curse of Affluence, and showed a technique for solving the latter in some cases. Any heuristic search planner that attempts to handle numeric domains will face these types of issues. The technique of reduced search with enhanced states is not tied to our implementation in CHCGP: it can be used in any heuristic search planner. The only requirement is to find a suitable modification to the state evaluation, so it can be applied to enhanced states. The experimental results demonstrate that in certain domains reduced search can produce significant performance improvements relative to traditional heuristic search.

We posit three criticisms which could be levelled at this approach:

- Over-reliance on pure translational operators.
- High computational costs associated with calculating convex hulls.
- Potential failure of post-processing step.

We address these in turn, starting with the first. It is certainly true that the technique relies strongly on the existence of pure translational operators. However, these operators can be used to represent quite natural structures. Consider the following two examples: **pooled variables** and **derived variables**.

The first type occurs when many numeric variables are all connected to one another and actually represent the same underlying resource. In this case, in order to increase one variable another must be decreased. A situation like this could occur in financial problems where the agent must allocate dollars to various projects. It could be the case that moving the money around does not change the rest of the problem state (i.e., the actions that move money have no propositional effects).

Derived variables occur when one resource can only be obtained by consuming some other resource. This might happen in a computer manufacturing problem, where in order to construct a computer a CPU, hard drive, motherboard, *etc* must be "consumed". It might also happen in purchasing problems where dollars can be converted into resources.

Regarding the second potential criticism: there is reason to believe that significant reductions in computational complexity can be made by refining the technique. For example, the representation of the reachable space does not need to include the minimal number of points possible. Extraneous points in the representation do not break the algorithm, they merely require more memory to maintain. Therefore it is not necessary to perform convex hull recomputations at every step. Optimizing the tradeoff between the cost of convex hull recomputations and increased memory cost of representing the hull could yield significant performance benefits. Furthermore, it may be that approximation techniques, in which one high-dimensional hull is broken apart into several lower-dimensional hulls, will allow significant performance improvements without giving up too much in terms of completeness.

With regards to the potential failure of the post-processing algorithm, a case can be made that the shortcoming is with the planning language and not the technique. While there are problems where operator applications are intrinsically discrete, there are also many problems where allowing continuous valued arguments would be more natural. For example in the Rovers domain, a rover can be recharged once for 20 units of energy, twice for 40 units, and so on, but it cannot be recharged by 37 units or 65.3 units in the standard PDDL formalism.

There are many comparisons that can be made between the reduced search method given in this section and related planning techniques. Broadly speaking, the technique can be considered a method for dimensionality reduction, and is thus related to other planning systems which apply such methods. It also involves a method for discarding dominated points. SGPlan [**16**], a recently successful planning system, involves discarding nodes based on the evaluation of a discrete space Lagrangian function. The domination criteria used in SGPlan differ significantly from ours. In SGPlan, the Lagrangian takes into account all constraints involved in the problem. If a node is not a local minimum of this function, it can be safely discarded in preference for a neighbor. The value of the Lagrangian changes as the search progresses, as some constraints begin to appear more difficult to satisfy than others. Our method uses only numerical constraints to discard nodes. However, this limitation allows us to discard entire *regions* of the search space, since every point in the convex hull of an enhanced state corresponds to many actual states.

CHAPTER 5

Conclusion

The purpose of this research has been to critically examine the concept of heuristic search planning, identify problems with the technique, and attempt to solve those problems.

5.1 Limitations of the Relaxed Plan Heuristic

The first half of this thesis dealt primarily with problems arising from a popular heuristic evaluation function called the relaxed plan heuristic (h^+). This evaluator works by calculating the length of the relaxed plan from the state under examination to the goal. Because in the relaxed problem operators have no delete effects, a relaxed plan can be found in polynomial time. This heuristic gives a good tradeoff between rapidity of evaluation and value of the information provided.

As with any heuristic, there are certain cases in which it gives systematically bad estimates. This was illustrated vividly in the simple domain called Push-Block (Figure 3.3). The problem in this domain (and many others) is actions that bring one goal condition closer push another goal condition further away. Thus the heuristic does not favor applying the action. Such conflicting goal conditions are called rival. Rival goal conditions create large regions of states with identical heuristic value. When such a region is encountered, the planner can only proceed by exhaustively searching the states.

5.1.1 RRT-Plan. The planning system described in Chapter 3 was inspired by the concept of Rapidly-exploring Random Trees (RRTs). This data structure and associated algorithm was originally intended to be used for mobile robot path planning. The RRT expansion algorithm is highly effective at growing the tree through the state space. Because of this rapid growth, the tree eventually comes close enough to the goal that a simple local search can succeed. There were several difficulties in translating the RRT concept into the discrete search space. It is problematic to find nearest neighbors, to sample randomly from the reachable space, and to connect a known point to a target point.

By using an alternate method of selecting random target states, much improved results were obtained. The reason for the improvement relates to the concept of artificial goal orderings. These are distinct from natural goal orderings in that the latter are necessary, while the former are acceptable: the planning problem can be solved by breaking it down into smaller pieces, and dealing with each piece in sequence. Often it is the case that many artificial goal orderings can be imposed on a given planning problem. This suggests the following analogy. In path planning, RRTs can succeed on large problems if solutions are abundant. In discrete planning, RRT-Plan can succeed on large problems in which acceptable artificial goal orderings are abundant.

Once such an artificial goal ordering is imposed, the problem becomes much easier, because rival goals no longer compete with one another. The heuristic function (often) gives information sufficient to achieve any individual goal, and so the full problem is reduced to solving several easy problems sequentially.

The results given in Chapter 3 show that RRT-Plan has good performance relative to the state of the art. On several domains (Blocks-World, DriverLog, Depot, Pipesworld and Push-Block) RRT-Plan achieves consistently better results than FF. RRT-Plan also does significantly better than LPG on at least two domains (FreeCell and Push-Block), though it also does worse on one domain (MPrime). An additional important point about RRT-Plan is that it can be modified to work with any heuristic search planner. Therefore new heuristic functions (such as the Causal Graph heuristic) can potentially be used to improve the performance of RRT-Plan as well.

5.2 Limitations of Heuristic Search in Numeric Domains

The second half of this thesis dealt with problems that arise when attempting to apply heuristic search techniques to numeric planning problems. Numeric planning is undecidable under very basic assumptions [**30**]. Thus it should not be imagined that the two issues discussed here are the only ones that must be faced by numeric heuristic search planners. However, they are general problems which occur in a variety of domains.

5.2.1 Two Curses. This work has identified two general issues for heuristic planning in numeric domains, which we call "curses". These pose significant difficulties for traditional heuristic search planners. The Curse of Poverty is caused by the problem of undetected resource shortfalls. The Curse of Affluence is caused by the blowup of the state space caused by extraneous resources.

If solving a numeric planning problem requires some resource to be conserved, in order to solve it efficiently a heuristic search planner must prune states that have an insufficient quantity of the resource. This could be done if a lower bound on the quantity required to reach the goal could be found. The Curse of Poverty is that such lower bounds are difficult to obtain. Part of the problem is that lower bounds are often not defined for single variables, but rather for combinations of variables. This is illustrated in by the Two Rovers problem (Section 4.3), in which the relevant lower bound is on the maximum of the two rovers' energy levels.

In some domains (e.g. Rovers and Settlers [**14**]) there are numeric operators that have no effect other than to increase the value of some variable. Thus in order to avoid turning a single state into a potentially infinite region of successor states, the agent must cap the number of times such an operator is applied. Again, this is hard to do because it is hard to estimate the amount of a resource that will be required to reach the goal. The Curse of Affluence refers to the difficulty of limiting the potentially infinite blowup in the state space caused by numeric variables.

5.2.2 Reduced Search and Enhanced States. As a way of addressing the Curse of Affluence, a new search scheme was presented in Chapter 4. This technique, known as reduced search, proceeds in the same way as normal search except that pure translational operators are removed. These operators are often responsible for situations in which the Curse of Affluence appears, because they can be applied an unlimited number of times.

In a reduced search, normal states are replaced by enhanced states. An enhanced state represents a region of numeric values which can be achieved by insertion of pure translational operators into the reduced plan leading to the state. Thus its size and shape is determined by the history of the reduced search.

As the reduced search proceeds, it is necessary to expand, update, and query the reachable space in various ways. An operator with numeric constraints is only applicable if some point in the reachable space satisfies the constraints. When applying a general operator with numeric effects, the reachable space must be appropriately updated. When new pure translational operators become applicable, the reachable space must be expanded to take these into account.

There are many ways to represent the reachable space. Under the assumption that effects and constraints are linear, the reachable space may be approximated very closely as a convex hull. For this representation, we need only maintain the extreme points of the hull. This representation satisfies the requirements for updating and querying the space in an intuitive way. Also, other researchers [**21, 22**] have developed advanced convex hull computation techniques which can be leveraged.

Of course, if the shape of the reachable space becomes complex, maintaining the representation can become computationally expensive. To be specific, computing the hull is exponential in $D/2$, where D is the number of dimensions of the space. In cases where the reachable space is large but relatively low-dimensional, the compression of the full state space thus achieved should more than make up for the additional costs. Furthermore, several additional techniques are employed to keep these costs to a minimum. Primarily, the notion of the relevant hull was introduced, which can significantly reduce the number of points that must be maintained by exploiting monotonicity on a per-variable basis.

The experimental results given in Chapter 4 demonstrate that there are problems that can be solved using the reduced search techniques of CHCGP, but cannot be solved by Metric-FF, which is a leading example of a heuristic search planner for numeric domains.

5.3 Future Work

5.3.1 Rapidly-exploring Dissimilarity Trees. At the outset of the development of RRT-Plan, the goal was to exploit the power of Rapidly-exploring Random Trees to solve large problems with many solutions. The initial idea was to choose target states by sampling choosing random sets of propositions. As the work progressed, a different approach began to look like it would be more effective. This was to choose a target state by sampling randomly from the goal atoms. While this yielded good results on many problems, a significant degree of the elegance and power of the original idea was lost.

This change of approach was motivated by several deep concerns. However, it may be possible to overcome these issues and create an implementation of RRT-Plan which is more faithful to the original RRT concept. To see how this might work, consider the process for selecting nodes for expansion in the RRT growth algorithm. A point is selected from the space at random, and its nearest neighbor in the tree is found. This strategy ensures that the nodes at the fringes of the tree are selected with the greatest probability. This can be stated in a more abstract way: the nodes that are selected for expansion are the ones which are maximally dissimilar from the rest of tree. This fact ensures that the tree grows rapidly throughout the state space.

Therefore, when translating the RRT algorithm into a new problem area, it should be sufficient to define a new measure of similarity. Given this measure, a simple process will suffice to expand the tree:

- Select the node in the tree with the least similarity to the other nodes,
- Expand outward from that node in the direction of the maximum dissimilarity.

Here, the expansion process is no longer random (a similar idea was introduced for continuous domains by [**52**]). However, it should retain the property of expanding throughout the space. Of course, everything depends on the quality of the similarity measure. A good measure would cause the tree to expand in interesting directions, while a bad measure would cause the tree to become constricted in one region of the space.

5.3.2 Generalized Reduced Search. Chapter 4 introduced the concept of reduced search. This technique depends on a particular definition of "simple" operators, that are then left out of the reduced search and inserted in the postprocessing stage. The definition used determines the form of the corresponding enhanced states. We discussed a specific form of reduced search, in which pure translational operators are simple, and enhanced states are represented by convex hulls.

Other definitions of operator simplicity can be explored. These definitions must be made with care, so that the resulting enhanced states can be represented compactly. Consider the following extreme examples. If we define every operator in the problem to be simple, then the first enhanced state created will be exactly the reachable space of the entire problem. On the other hand if we allow only operators with no preconditions and a single add effect, then the enhanced states will show that it is always possible for the corresponding proposition to be made true. While these examples are not particularly interesting, it is reasonable to believe that suitable definitions can result in significant performance improvement.

Helmert has recently incorporated this idea into Fast Downward [**23**]. His realization was that variables which were truly at the bottom of the Causal Graph, and which had fully connected DTGs (see Section 2.8), could be ignored in the search. Since every value for the variable can be achieved, a post-processing step can easily insert the correct set of operators into the full plan whenever a specific precondition is required. Helmert refers to this as *safe abstraction.* This technique corresponds to identifying simple operators as those that only have preconditions and effects on variables at the base of the causal graph.

Thus an interesting line of future work is to examine various ways of defining simple operators, and exploring the impact these have on the planning process. It may be that single operators will not be sufficiently simple, but sequences of operators can be found which result in simple effects (by canceling out most of each other's effects, but leaving something left over).

5.4 Summary

Discrete planning is a difficult enterprise, as demonstrated by the theoretical results of Bylander [**1**] and Helmert [**30**] discussed in Section 2.2. Because of this difficulty, it is clear that there are many domains which can never be efficiently solved by a planning system. However, it is also clear that there are wide categories of problems which *can* be solved.

Let $\mathcal{A}$ stand for the set of domains that are tractable in principle, i.e., the set of problems that can be solved in polynomial time by some algorithm (possibly one designed specifically for the domain). Let $\mathcal{B}$ stand for the set of domains that can be solved efficiently by current domain-independent planning systems. Then the goal of discrete planning research is to enlarge $\mathcal{B}$ to include as many domains in $\mathcal{A}$ as possible.

This thesis contributes to the above project in several ways. First, several simple and tractable problems were identified that could not be solved efficiently by standard techniques. These domains involved either the existence of rival goals, or the Curse of Affluence. We expanded $\mathcal{B}$ by developing two planning techniques. The first was an algorithm called RRT-Plan, which is very effective at finding artificial goal orderings. A problem is generally made easier by imposing an artificial goal ordering, assuming a valid one can be found. The second system was called CHCGP, which used the method of reduced search. Reduced search can effectively compress the search space when there are pure translational operators in the domain. While these techniques will not work on every problem, it can be hoped that they will be incorporated into the standard set of discrete planning tools.

One recurring theme in this research is the distinction between discrete valued (propositional) variables and continuous numeric variables. Intuitively, it would seem that numeric variables are more difficult to deal with, since adding them causes the search spaces to become infinite. However, we can observe that sometimes numeric variables are in fact easier to handle than discrete ones. The enhanced state techniques discussed in Chapter 4 illustrate one set of conditions under which this observation is true. Similarly, the continuous space RRT algorithm is more effective than our version of it for discrete planning.

These notes provide one possibility for future extensions of automated planning. Given the right set of assumptions, continuous planning might be a more promising and practical direction for the field to take.

APPENDIX A

Convex Hull Causal Graph Planner

A.1 CHCGP - Convex Hull Causal Graph Planner

The Convex Hull Causal Graph Planner (CHCGP) is a numeric planning system that employs reduced search as described in Chapter 4. It uses a modified version of the Causal Graph heuristic [**23**] to evaluate enhanced states. Note that the techniques described in Chapter 4 do not rely on this particular implementation or heuristic function.

A.2 Design Goals

The above observations motivated the development of the planning system described in this chapter. This system is based on Fast Downward by Helmert [**23**]. In modifying Fast Downward, three design goals were established:

- The performance of the new planner should not be meaningfully degraded relative to the original system on non-numeric problems.
- The system attempts to deal with the Curse of Affluence by conducting a reduced search through enhanced states. These enhanced states represent regions of reachable numeric values (Section 4.5).
- The system attempts to deal with the Curse of Poverty by tracking resource consumption within DTG traversals.

A.3 Virtual Purchase Flags

The causal graph heuristic is based on the recursive calculation of costs to change a variable from one value to another. For example $cost_v(d, d')$ gives the cost to change variable v from $d \rightarrow d'$. These costs are then propagated through to the costs of the CG successors of v. In order to modify this technique to take numeric variables into account, we must simply define some way of obtaining these cost functions.

To do this, for every numeric variable v_j, we introduce a pair of binary variables $\{v_j^+, v_j^-\}$. These variables represent the fact that a variable has been increased or decreased in the heuristic estimate. Thus the DTGs for these variables have exactly two states. Every operator which can potentially increase (decrease) v_j has a corresponding edge in the DTG of v_j^+ (v_j^-). These variables are called virtual purchase flags because they indicate whether a numeric variable has been "purchased", i.e., whether a cost has been paid to change it.

The purchase flag has a two-state DTG, and only contains transitions leading from 0 to 1. The number of transitions in this DTG is equal to the number of operators that could increase or decrease a given numeric variable (this is determined in the preprocessing stage, see Section 4.4). To take into account non-constant effects, and assignment effects that may or may not increase the variable, the purchase flag DTG is constructed dynamically depending on the current state (note that the current state is the one being evaluated, as opposed to the local states referred to in A.1).

A.4 Numeric Causal Graph Heuristic

Given the notion of virtual purchase flags as defined above, our modifications to the CG heuristic will be simple (Figure A.1, the original is given in Figure 2.8). There are three functions invoked by this modified heuristic that must be explained.

SATISFIES(...) merely checks if the given local state satisfies a constraint. The function is optimistic, in that it returns true if any point in the associated hull satisfies the constraint. Importantly, the algorithm checks only one constraint at a time. Determining if a convex hull satisfies one linear constraint can be done simply by checking each extreme point. The function APPLY_EFFECTS evaluates the effects of the given operator for each point in the hull, and makes the appropriate modifications. This is just the numeric equivalent of updating the local state to take into account the effects of the operator. The

```
**function** COMPUTE_COSTS($\Pi, s, v, d$)
  Let $V'$ be the set of immediate predecessors of $v$ in the pruned CG of $\Pi$.
  $DTG := CONSTRUCT\_DTG(v, s)$
  $cost_v(d, d) := 0$
  $cost_v(d, d) := \infty$ for all $d' \in D_v \setminus \{d\}$
  $local\_state_d := s$ restricted to $V'$
  $unreached := D_v$
  **while** $unreached$ contains a value $d' \in D_v$ with $cost_v(d, d') < \infty$ **do**
    Choose such a value $d' \in unreached$ minimizing $cost_v(d, d')$.
    $unreached := unreached \setminus \{d'\}$
    **for each** transition $t$ in $DTG$ leading from $d'$ to some $d'' \in unreached$ **do**
      $transition\_cost := 0$
      $cond\_set := cond(t)$
      **for each** constraint $C$ of $op(t)$ **do**
        **if not** SATISFIES($local\_state_{d'}, C$) **then**
          $cond\_set := cond\_set + (virtualvar(C), 1)$
        **end if**
      **end for**
      **for each** pair $v' = e'$ in $cond\_set$ **do**
        $e := local\_state(v')$
        **call** COMPUTE_COSTS($\Pi, s, v', e$)
        $transition\_cost+ = cost_{v'}(e, e')$
      **end for**
      **if** $cost_v(d, d') + transition\_cost < cost_v(d, d'')$ **then**
        $cost_v(d, d'') := cost_v(d, d') + transition\_cost$
        $local\_state_{d''} := local\_state_{d'}$
        **for each** pair $v' = e'$ in $cond\_set$ **do**
          $local\_state_{d''}(v') := e'$
        **end for**
        **for each** numeric effect $E$ of $op(t)$ **do**
          APPLY_EFFECT($local\_state_{d''}, E$)
        **end for**
      **end if**
    **end for**
  **end while**
```

FIGURE A.1. Modified Causal Graph heuristic for problems without multi-variable constraints. Compare to the original COMPUTE_COST algorithm, Figure 2.8.

subroutine CONSTRUCT_DTG returns the standard DTG if the variable is discrete, or the virtual purchase flag DTG if it is numeric.

The heuristic evaluator only takes into account the cost of achieving the preconditions of a single operator that increases or decreases a numeric variable, not the number of times it must be applied to satisfy the constraint. In this sense, the heuristic can be thought of as assuming that numeric operators are pure geometric, and calculating the distance to the goal in the *reduced* search, not the full search. This is in keeping with both the overall design of the planner and the idea that it is preferable to underestimate goal distances.

There are some subtleties about this technique that should be noted. The important points are that:

- If the local states satisfy a constraint, nothing is added to the *cond_set*. Therefore no cost is incurred, and the purchase flags are not set.
- Resource consumption is tracked through DTG traversals, through the action of APPLY_EFFECTS on the local states.
- In any given DTG traversal, the cost of increasing or decreasing a numeric variable is only paid once.

The second property above provides a loose lower bound on the resource consumption required to reach the goal. If the required consumption of a resource within one DTG traversal is greater than is available in the state, it will be assigned an infinite heuristic value and pruned (assuming the resource cannot be produced).

A.5 Multi-Variable Constraints

Figure A.1 shows the action of the heuristic in cases where all constraints in the problem involve single variables. This is often a valid assumption, but it is possible to provide for the general case.

The difficulty now is that constraints can be satisfied by changing different variables. This is analogous to an OR precondition the discrete variables, and it will be handled in basically the same way.

When constructing the initial DTGs at the outset of the search, N copies of of each transition with a multivariable constraint are made, where N is the number of variables in the constraint. Each copy corresponds to a different variable that is added to the *cond_set* in the DTG traversal. In other words, the $virtualvar(C)$ statement returns a different variable for each copy.

The intuition here is that the constraint can be satisfied by changing any one of the relevant variables. It does not need to be paid for multiple times. Thus the DTG algorithm will attempt each transition, and keep the one corresponding to the least expensive variable to modify.

One additional problem with multi-variable constraints is that they could cause cyclic causal graphs. Consider a discrete variable v_A the DTG for which has a multivariable constraint involving v_B and v_C. There are three possibilities. If v_A is above the others, then the constraint should be considered. If v_A is below the other two, the constraint should be ignored (attempting to satisfy it might result in an infinite loop). The third case, where v_A is between v_B and v_C in the CG, causes some puzzlement. It is possible to address the constraint by looking only at the lower variable. However, this would tend to make constraint harder to ameliorate, because there are fewer variables available to change. We decided on a simple fix: only consider the constraint if the highest variable it involves is below the variable being evaluated.

A.6 Inserting Numeric Variables in the Causal Graph

In the discrete case, a causal graph arc exists between two variables v, v' if some operator $o \in \mathcal{O}$ has an effect on v' and a precondition or effect on v. Note that the CG as defined in this way must then be pruned to be used in the heuristic calculation. The question now arises as to how to extend this definition of causal graph arcs to the numeric case. The simplest thing to do is merely expand the condition above to include constraints as well as preconditions and numeric effects as well as discrete effects. This will work for the most part.

However, consider the following case. In the Rovers domain there are many operators which have a numeric constraint and an effect involving energy, and a precondition and effect involving some other variable (say, the location of the rover). Here it seems more natural to put the energy variable lower than the Rover location variable in the CG, because it seems appropriate to think of the energy variable's role in the operator as primarily a constraint, and not as an interesting effect. The naïve definition we used above will not discriminate between the two variables: since both are involved in a precondition and both are involved in effects of the operator, the CG pruning algorithm will have no reason to place one above or below the other. To do this, we modify the pruning algorithm so that it detects resource variables like energy and pushes them upstream in the pruned CG.

A.7 Comments

Our attempt to handle the Curse of Poverty resulted in the technique described regarding resource consumption tracking within DTGs. While better than nothing, this technique is far from satisfactory. This method is able to prune states only when the required resource availability is significantly too large. This is because the method can only track consumption within a *single* DTG traversal.

In some cases, this will be sufficient. Unfortunately, in most cases the consumption of a resource will be spread out through multiple DTG traversals. Consider a Logistics problem with several packages and only one truck. Let V denote the location of the truck, and let $\{v_1, v_2, \ldots v_n\}$ be the values V must move through to satisfy the preconditions for moving around the packages. Furthermore let $\{r_1, r_2, \ldots r_{n-1}\}$ be the fuel costs associated with moving V through this set of values. Then the total fuel cost required is just the sum of the $\{r_i\}$. However, each DTG traversal only calculates one value of fuel consumption. Because the traversals are required to be isolated from one another (i.e., we cannot use information from the first traversal to modify the outcome of the second), the method described above prunes states only if:

$$fuel(S) < \max_i r_i$$

The reason information from the traversals must be kept isolated is that violating this rule would require many more calls of the COMPUTE_COSTS routine, and therefore dramatically degrade the performance of the heuristic evaluator.

APPENDIX B

Planning Domains

This Appendix describes several new domains that have been developed as part of this research, including PDDL definitions. Additionally, informal descriptions are given of the standard benchmark domains discussed in the document.

B.1 New Domains

B.1.1 Push-Block. This simple domain involves a set of blocks moving around on a grid. The block locations are represented by propositions such as *OCCUPIED*(x, y). The blocks can be pushed vertically or horizontally. The goal specifies a certain set of locations that must be occupied by blocks.

The interesting feature of this domain is that the goals are often rival. The case often occurs that the most direct way to achieve one goal requires moving a block away from another goal. This is the situation illustrated in Figure 3.3. The PDDL code for this domain is given in Figure 1.1. The problems described in Chapter 3 were generated by choosing starting locations and goal locations at random for N blocks, where N is between 1 and 20.

B.1.2 T-Logistics. This is the problem described in Figure 4.1. There is a “T” formation of locations. The single truck starts at the intersection of the T. The goal is to deliver packages to both edges of the T. The truck begins with a single package, and a second package is waiting at the base of the T. The truck requires fuel to move, and fuel can only be obtained at the initial location.

```
(define (domain push-block)
    (:requirements :strips :typing)
    (:types xcoord ycoord)

    (:predicates
        (occupied ?x - xcoord ?y - ycoord)
        (above ?ya - ycoord ?yb - ycoord)
        (right ?xa - xcoord ?xb - xcoord)
    )


(:action PUSH-LEFT
    :parameters    (?xa - xcoord ?xb - xcoord ?y - ycoord)
    :precondition
    (and
            (right ?xa ?xb)
            (occupied ?xa ?y)
            (not (occupied ?xb ?y))
    )
    :effect
    (and
        (occupied ?xb ?y)
        (not (occupied ?xa ?y))
    )
)

(...)

)
```

FIGURE B.1. The Push-Block domain. Only the Push-Left action is shown; other directions are exactly analogous. Predicates *above* and *right* encode adjacency relationships.

B.1.3 Accumulate. This domain has a set of N resources. Initially, a total of 120 units is divided equally between the various resources. The agent may trade resources one for one. There are a set of goal flags corresponding to each resource, which can be achieved when a threshold T of that resource is accumulated (T is the same for all resources). Once a goal flag is achieved, the corresponding resource can be traded away.

Because there are goals associated with each resource, and accumulating one type of resource requires trading away another, the goals are rival. At the outset, the modified h^+ heuristic does not discriminate between states with greater or lesser quantities of any resource. However, once one goal flag has been achieved, the corresponding resource is no longer needed and h^+ gives good goal estimates.

B.1.4 Toll Chain. In this domain a car moves through a series of links in a chain. To move across each link, a "toll" in the form of a resource cost is required. There are three resources. The toll is one of each resource (more complex tolls can be imagined and may have interesting implications, but for the current purposes simpler is better). The resources may be produced only at the initial state of the chain. The goal is to get to the end of the chain.

Two parameters of interest are the length of the chain, and the number of production actions that are allowed. If an insufficient number of production actions are available, the problem is unsolvable.

B.1.5 Shortcut Trap. This domain again considers a chain of states. A "goal-silo" location exists at every other link in the chain. Each goal silo requires a certain number of resources to satisfy, and the goal of the problem is to satisfy *some* goal silo. Resources can be produced, but after producing a resource the agent cannot move any further (in other words, it cannot produce some amount of a resource then realize the amount is insufficient for the current goal silo and move on to the next). The total resource requirements for goal silos decrease at a constant rate the further away they are from the initial state.

Therefore, in order to solve the domain efficiently, the agent must somehow know that the early goal silos require too much resources and move on without stopping. The amount of initial resource available is a parameter, as is the resource requirement for the first goal silo.

B.1.6 Pressurized Blocks-World. This domain is identical to traditional Blocks-World, with the addition of a single variable called "pressure" associated with each block. There is a single action called **PRESSURIZE** which increments the pressure value for a block. This operator is applicable when the block is **ON-TABLE** and **CLEAR**.

There are two varieties of this domain, referred to as "simple" and "hard". In the simple version, in order to *STACK*(A B), block A must have greater pressure than block B. In the hard version, the opposite must be true. The latter version is called hard because the planning system must somehow "know" in advance how high a stack of blocks will eventually become in order to adequately pressurize the block at the base.

B.2 Previous Domains

Note that Blocks-World and Logistics are discussed in Section 2.1. The following three domains are discussed in the document, so we describe them in some detail here. These domains are taken from IPC3 [**14**]. More recent planning competitions haven't introduced many new numeric planning domains, focusing instead on new aspects of the formalism.

B.2.1 DriverLog. This domain is a variant of Logistics. There are now drivers as well as trucks and packages, and all three types may have goal destinations. A truck must have a driver in order to move. There are also now two types of links between locations: highways which can be driven but not walked across, and paths which can be walked but not driven on.

B.2.2 Rovers. This domain seeks to model rover robots moving around on the surface of Mars. There are a collection of data types that must be obtained - soil samples, rock samples, and images. The data must be sent back to Earth for analysis through the lander. A rover can only have one soil or rock sample in its store at a time. Locations (called waypoints) can be visible from one another, and this visibility has implications for taking images and transmitting data.

The main interest of the domain for this work is that there is a numeric version in which each rover has a single energy variable (alternately fuel). Many of the actions performed by a rover require varying amounts of energy. There is also a **RECHARGE** operator which is applicable at certain locations. This operator is qualifies as pure translational (see Chapter 4).

B.2.3 Settlers. This rather complex domain involves gathering resources, and constructing buildings and vehicles. It is also similar to Logistics in that it involves moving resources around in vehicles, which can only traverse certain links. Locations have various properties which determine what kinds of resources can be gathered there. Some resources can be harvested from nature and others must be manufactured out of other resources. There are a set of buildings that can be built, with various costs. Some buildings allow resources to be manufactured.

Interestingly, many of the operators in this domain are pure translational as defined in Chapter 3. One example is **FELL-TIMBER** which requires a cabin (discrete variable) and then allows any amount of timber to be produced. Another example is **MAKE-IRON** which consumes ore and coal to produce iron, but has no propositional effects and so can be applied as many times as the ore and coal supplies allow.

```
(define
    (domain easyTLog)
    (:requirements :strips :fluents)
    (:predicates
            (link ?a ?b) (pos ?a) (package ?p) (truck ?t)
        (at ?t ?a) (in ?p ?t) (package-at ?a) (fuel-dump ?a)
    )

    (:functions (fuel ?t))

    (:action drive
    :parameters (?t ?a ?b)
        :precondition
        (and (at ?t ?a) (link ?a ?b) (truck ?t)
        (>= (fuel ?t) 1))
        :effect
        (and (at ?t ?b) (not (at ?t ?a)) (decrease (fuel ?t) 1))
    )

    (:action load-package
            :parameters (?t ?p ?a)
        :precondition
        (and
        (package ?p) (truck ?t) (at ?t ?a)
        (at ?p ?a) (package-at ?a)
        )
        :effect
    (and (in ?p ?t) (not (at ?p ?a)) (not (package-at ?a)))
    )

    (:action deliver-package
    :parameters (?t ?p ?a)
        :precondition
            (and (package ?p) (truck ?t) (at ?t ?a) (in ?p ?t))
        :effect
            (and (at ?p ?a) (not (in ?p ?t)) (package-at ?a)))

    (:action refuel
    :parameters (?t ?a)
        :precondition
            (and (truck ?t) (fuel-dump ?a) (at ?t ?a))
        :effect (and (increase (fuel ?t) 1) )
    )
)
```

FIGURE B.2. The T-Logistics domain. The main difference between T-Logistics and standard Logistics is that here the packages are identical, and there is also an unlimited fuel variable.

```
(define (domain vargame)
    (:requirements :strips :fluents)
    (:predicates
        (achieved ?goal)
        (numvar ?v)
        (goalvar ?g ?v)
    )
    (:functions
        (value ?var)
        (goalthresh)
    )
    (:action trade
        :parameters (?fr ?to)
        :precondition
        (and
            (numvar ?fr)
            (numvar ?to)
            (> (value ?fr) 0)
        )
        :effect
        (and
            (increase (value ?to) 1)
            (decrease (value ?fr) 1)
        )
    )

    (:action achieveGoal
        :parameters (?g ?v)
        :precondition
        (and
            (goalvar ?g ?v)
            (>= (value ?v) (goalthresh))
        )
        :effect (achieved ?g)
    )
)
```

FIGURE B.3. The Accumulate domain. The goal threshold and the number of variables are parameters that are modified to produce the different problems.

```
(define
    (domain tollChain)
    (:requirements :strips :fluents)
    (:predicates
        (at ?col)
        (good ?good)
        (canbuild ?col)
        (col ?colA)
        (link ?colA ?colB)
    )
    (:constants resA resB resC)
    (:functions
        (quant ?good)
        (numbuilds)
    )

    (:action move-horz
                :parameters (?colA ?colB)
                :precondition
            (and
            (at ?colA)
            (link ?colA ?colB)
            (>= (quant resA) 1)
            (>= (quant resB) 1)
            (>= (quant resC) 1)
            )
        :effect (and
            (at ?colB)
            (not (at ?colA))
            (decrease (quant resA) 1)
            (decrease (quant resB) 1)
            (decrease (quant resC) 1)
            )
    )

    (:action produce
        :parameters (?good ?col)
        :precondition
        (and
            (good ?good)
            (> (numbuilds) 0)
            (canbuild ?col)
            (at ?col)
            )
        :effect
        (and
            (increase (quant ?good) 1)
            (decrease (numBuilds) 1)
            )
    )
)
```

FIGURE B.4. The Toll Chain domain. The required values for each link are chosen randomly between one and three. Production of resources can only occur at the initial location.

```
(define
    (domain easyhTrap)
    (:requirements :strips :fluents)
    (:predicates
        (at ?loc) (link ?locA ?locB) (goal-silo ?loc)
        (goal-achieved) (location ?loc)
        (resource ?res) (move-phase)
    )

    (:functions (quant ?res) (valreq ?res ?loc) (numbuilds))
    (:constants resA resB resC resD resE)

    (:action move
        :parameters (?locA ?locB)
        :precondition
        (and (move-phase) (at ?locA) (link ?locA ?locB))
        :effect
        (and (at ?locB) (not (at ?locA)))
    )

    (:action manufacture
        :parameters (?res)
        :precondition  (and (not (move-phase)) (> (numbuilds) 0))
        :effect
        (and
            (not (move-phase))
            (resource ?res)
            (increase (quant ?res) 1)
            (decrease (numbuilds) 1)
        )
    )

    (:action sat-location
        :parameters (?loc)
        :precondition
        (and
            (goal-silo ?loc)
            (at ?loc)
            (>= (quant resA) (valreq resA ?loc))
            (>= (quant resB) (valreq resB ?loc))
            (>= (quant resC) (valreq resC ?loc))
            (>= (quant resD) (valreq resD ?loc))
        )
        :effect (goal-achieved)
    )
)
```

Figure B.5. The Shortcut Trap domain. The required values for the goal silos are chosen randomly but must add up to a predetermined total value, which decreases the further away one moves from the initial state. Goal silos occur at every other link on the chain.

```
(define (domain BLOCKS)
  (:requirements :strips :fluents)
  (:predicates (on ?x ?y) (ontable ?x) (clear ?x)
        (handempty) (holding ?x))

  (:functions (pressure ?block))

  (:action pick-up
             :parameters (?x)
             :precondition (and (clear ?x)
                (ontable ?x) (handempty))
             :effect
             (and (not (ontable ?x)) (not (clear ?x))
            (not (handempty)) (holding ?x)))

  (:action put-down
             :parameters (?x)
             :precondition (holding ?x)
             :effect
             (and (not (holding ?x)) (clear ?x)
            (handempty) (ontable ?x)))

  (:action pressurize
        :parameters (?x)
        :precondition (ontable ?x)
        :effect (and (ontable ?x) (increase (pressure ?x) 1))
  )

  (:action stack
             :parameters (?x ?y)
             :precondition
         (and (holding ?x) (clear ?y)
            (> (pressure ?x) (pressure ?y)))
             :effect
             (and (not (holding ?x)) (not (clear ?y))
            (clear ?x) (handempty) (on ?x ?y) )
  )

  (:action unstack
             :parameters (?x ?y)
             :precondition (and (on ?x ?y) (clear ?x) (handempty))
             :effect
             (and (holding ?x)
                   (clear ?y)
                   (not (clear ?x))
                   (not (handempty))
                   (not (on ?x ?y)))))
```

FIGURE B.6. The Pressurized Blocks-World domain. This figure shows the simple version. The complex version is obtained by flipping the direction of the inequality in the **STACK** action.

REFERENCES

[1] T. Bylander, “The computational complexity of propositional STRIPS planning,” *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.

[2] S. Trug, J. Hoffmann, and B. Nebel, “Applying automatic planning systems to airport ground-traffic control - a feasibility study,” in *Advances in Artificial Intelligence* (S. Biundo, T. Fruhwirth, and G. Palm, eds.), (Ulm, Germany), pp. 183–197, Springer-Verlag, 2004.

[3] R. L. Milidiu and F. dos Santos Liporace, “Plumber, a pipeline transportation planner,” in *Proceedings of the International Workshop on Harbour and Maritime Simulation (HMS)*, (Rio de Janeiro, Brazil), pp. 99–106, 2004.

[4] S. Edelkamp, “Promela planning,” in *Proceedings of Model Checking Software*, pp. 183–212, 2003.

[5] J. Hoffmann and B. Nebel, “The FF planning system: Fast plan generation through heuristic search.,” *J. Artif. Intell. Res. (JAIR)*, vol. 14, pp. 253–302, 2001.

[6] A. Gerevini and I. Serina, “LPG: A planner based on local search for planning graphs with action costs.,” in *AIPS* (M. Ghallab, J. Hertzberg, and P. Traverso, eds.), pp. 13–22, AAAI, 2002.

[7] H. A. Kautz and B. Selman, “Planning as satisfiability.,” in *ECAI*, pp. 359–363, 1992.

[8] P. Haslum and H. Geffner, “Admissible heuristics for optimal planning.,” in *AIPS*, pp. 140–149, 2000.

[9] M. Helmert, "A planning heuristic based on causal graph analysis," in *Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pp. 161–170, 2004.

[10] N. J. Nilsson, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Company, 1980.

[11] B. Bonet and H. Geffner, "Planning as heuristic search: New results," in *Proc. 5th European Conf. on Planning* (S. Biundo and M. Fox, eds.), (Durham, UK), pp. 359–371, Springer: Lecture Notes on Computer Science, 1999.

[12] D. V. McDermott, "The 1998 AI planning systems competition.," *AI Magazine*, vol. 21, no. 2, pp. 35–55, 2000.

[13] F. Bacchus, "The AIPS '00 planning competition.," *AI Magazine*, vol. 22, no. 3, pp. 47–56, 2001.

[14] D. Long and M. Fox, "The 3rd international planning competition: Results and analysis.," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 1–59, 2003.

[15] A. Gerevini, Y. Dimopoulos, P. Haslum, and A. Saetti, "The fifth international planning competition." Online at `http://zeus.ing.unibs.it/ipc-5/`, 2006.

[16] Y. Chen and B. W. Wah, "Automated planning and scheduling using calculus of variations in discrete space," in *Proc. of Intl Conf. on Automated Planning and Scheduling*, pp. 2–11, 2003.

[17] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics.* New York, NY: Cambridge University Press, 2000.

[18] J. Latombe, *Robot Motion Path Planning.* Boston, MA: Kluwer Academic Publishers, 1991.

[19] S. M. LaValle, *Planning Algorithms.* New York, NY: Cambridge University Press, 2006.

[20] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 995–1001, 2000.

[21] F. Preparata and M. Shamos, *Computational Geometry: An Introduction.* Springer-Verlag: Texts and Monographs in Computer Science, 1985.

[22] D. Avis, “lrs: A revised implementation of the reverse search vertex enumeration algorithm,” in *Polytopes - Combinatorics and Computation* (G. Kalai and G. Ziegler, eds.), pp. 177–198, Birkhauser-Verlag, 2000.

[23] M. Helmert, “The Fast Downward planning system,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[24] R. Fikes and N. J. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving.,” in *IJCAI*, pp. 608–620, 1971.

[25] E. P. D. Pednault, “ADL: Exploring the middle ground between strips and the situation calculus.,” in *KR*, pp. 324–332, 1989.

[26] M. Fox and D. Long, “PDDL2.1: An extension to PDDL for expressing temporal planning domains.,” *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 61–124, 2003.

[27] A. Gerevini and D. Long, “Preferences and soft constraints in PDDL3,” in *Proceedings of the ICAPS-2006 Workshop on Preferences and Soft Constraints in Planning*, pp. 46–53, 2006.

[28] C. Bäckström, “Expressive equivalence of planning formalisms.,” *Artif. Intell.*, vol. 76, no. 1-2, pp. 17–34, 1995.

[29] K. Erol, D. S. Nau, and V. S. Subrahmanian, “Complexity, decidability and undecidability results for domain-independent planning.,” *Artif. Intell.*, vol. 76, no. 1-2, pp. 75–88, 1995.

[30] M. Helmert, “Decidability and undecidability results for planning with numerical state variables.,” in *Proceedings Workshop Planen und Konfigurieren* (J. Sauer, ed.), (Freiburg, Germany), 2002.

[31] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1979.

[32] Y. Matiyasevich, “Enumerable sets are Diophantine,” in *Doklady Akademii Nauk SSSR*, pp. 279–282, 1970.

[33] A. Blum and M. Furst, "Fast planning through planning graph analysis," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pp. 1636–1642, 1995.

[34] B. Selman, H. J. Levesque, and D. G. Mitchell, "A new method for solving hard satisfiability problems.," in *AAAI*, pp. 440–446, 1992.

[35] B. Selman and H. A. Kautz, "Domain-independent extensions to GSAT: Solving large structured satisfiability problems.," in *IJCAI*, pp. 290–295, 1993.

[36] H. A. Kautz and B. Selman, "Unifying sat-based and graph-based planning.," in *IJCAI* (T. Dean, ed.), pp. 318–325, Morgan Kaufmann, 1999.

[37] J. Koehler and J. Hoffmann, "On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm," *Journal of Artificial Intelligence Research*, vol. 12, pp. 338–386, 2000.

[38] J. Hoffmann, "Local search topology in planning benchmarks: A theoretical analysis," in *Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, (Toulouse, France), Apr. 2002. 379-387.

[39] J. Hoffmann, "The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables.," *J. Artif. Intell. Res. (JAIR)*, vol. 20, pp. 291–341, 2003.

[40] M. Fox and D. Long, "Utilizing automatically inferred invariants in graph construction and search.," in *AIPS*, pp. 102–111, 2000.

[41] S. Edelkamp and M. Helmert, "The model checking integrated planning system (MIPS)," *AI Magazine*, vol. 22, no. 3, pp. 67–71, 2001.

[42] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning." TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[43] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots," in *Proc. 11th Intl Symp. of Robotics Research (ISRR 2003)*, November 2003.

[44] L. Kavraki, M. Kolountzakis, and J. Latombe, "Analysis of probabilistic roadmaps for path planning," in *Proc. IEEE Internat. Conf. Robot. Autom.*, pp. 3020–3025, 1996.

[45] S. Akl and G. Toussaint, "Efficient convex hull algorithms for pattern recognition applications," in *Proceedings of the Fourth International Joint Conference on Pattern Recognition*, pp. 483–487, 1978.

[46] A. Rosenfeld and A. C. Kak, *Digital Picture Processing.* Orlando, FL, USA: Academic Press, Inc., 1982.

[47] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," in *Info. Proc. Lett.*, pp. 18–21, 1973.

[48] E. Welzl, D. Avis, D. Bremner, and R. Seidel, "How good are convex hull algorithms?.," *Comput. Geom.*, vol. 7, pp. 265–301, 1997.

[49] R. E. Korf, "Macro-operators: a weak method for learning," *Artif. Intell.*, vol. 26, no. 1, pp. 35–77, 1985.

[50] C. A. Knoblock, "Learning abstraction hierarchies for problem solving," in *Proceedings of the Eighth National Conference on Artificial Intelligence* (T. Dietterich and W. Swartout, eds.), (Menlo Park, California), AAAI Press, 1990.

[51] A. Gerevini, A. Saetti, and I. Serina, "Planning with numerical expressions in LPG.," in *ECAI* (R. L. de Mántaras and L. Saitta, eds.), pp. 667–671, IOS Press, 2004.

[52] S. Lindemann and S. LaValle, "Steps toward derandomizing RRTs," in *Proc. of the Fourth International Workshop on Robot Motion and Control (RoMoCo'04)*, pp. 271–277, June 2004.

Document Log:

Manuscript Version 0
Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX — 29 January 2007

Daniel Burfoot

McGill University, 3480 University St., Montréal (Québec) H3A 2A7, Canada, *Tel.* : (514) 398-7071

E-mail address: `burfoot@cim.mcgill.ca`