Graph-Theoretical Classification of the Complexities of Planning Domains

Søren Bøg
052259

Kongens Lyngby 2012
IMM-MSC-2012-01

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

Automated planning is one of the cornerstones of modern Artificial Intelligence (AI). As with many of the fields within AI, automated planning is something we do naturally as humans yet is extremely hard to do computationally, this seems paradoxical. Therefore there is much interest in determining under what conditions automated planning is easy. This thesis examines the use of the state graph of planning problems to define conditions on automated planning. As a part of this two new cases are presented where planning is easier than full automated planning and in one of the cases actually tractable.

Resumé

Automatiseret planlægning er en af de fundamentale områder inden for moderne kunstig intelligens. Som så mange andre områder indenfor kunstig intelligens er automatiseret planlægning både noget vi, som mennesker, gør meget naturligt og samtidigt utroligt svært beregningsmæssigt. Der er derfor stor interesse for at afgøre under hvilke forhold automatiseret planlægning bliver nemt. Denne rapport klarlægger brugen af et planlægningsproblems tilstandsgraf som basis for at begrænse automatiseret planlægning. Som en del af dette introduceres to nye tilfælde, hvor automatiseret planlægning er nemmere end i det generelle tilfælde. I et af disse tilfælde er automatiseret planlægning endda håndterbart.

Preface

This thesis is a part of a masters project completed at the institute for Informatics and Mathematical Modelling, at the Technical University of Denmark, in accordance with the requirements of obtaining a masters degree in Computer Science and Engineering. This thesis documents the theoretical work done during the project and presents it within the context of other results in the field.

The following theorems, lemmata and corollaries, along with their associated proofs, have been developed as part of this project: 2.21 2.22 4.8 4.9 5.3 5.4 5.6 5.7 5.8 5.9 5.11 5.14 5.15 5.16 5.17 5.18 .

The following theorems, lemmata and corollaries are known from other literature, however, the presented proofs have been developed as part of this project: 2.19 2.20 .

Any other theorems, lemmata and/or corollaries, along with any associated proofs, are from a cited source or constitute a portion of the automated planning "folklore".

Kongens Lyngby, January 2012

Søren Bøg

Acknowledgements

Thanks must first and foremost go to my supervisor Thomas Bolander, without whom my interest in Automated Planning, Artificial Intelligence or even general Algorithmics might never have been sparked. His great guidance and patience are a virtue which I have without doubt tested during this project, particularly when things where not looking good.

Next I must thank my proof-reading and fact-checking team, in no particular order; Martin Holm Jensen, Morten Stöckel, Rasmus Ljungmann Pedersen, Mette Terp Petersen and Michael Faursby Ahlers. Without whom this thesis would without doubt contain many more errors and omissions.

I would also like to extend my thanks to, once again; Martin Holm Jensen, Morten Stöckel and Rasmus Ljungmann Pedersen, for many years of fantastic study and sparring.

Finally thanks go to the entire team of the DTUsat 2 project, which is way too big to list here. Without these guys I would have been bored out of my mind many years ago.

Contents

Chapter 1

Introduction

> Life is what happens to you while you're busy making other plans
>
> - John Lennon

Automated planning is one of the cornerstones of modern Artificial Intelligence (AI). It enables agents to know how they may affect their environment, in a rational manner, such that they may achieve some given goal. As with many central problems in AI, automated planning is extremely hard from a computational point of view. Even extremely restricted domains are intractable in the general case. Traditionally domains for automated planning have been restricted syntactically by limiting the number and/or type of conditions and effects.

These restrictions suffer from two issues; firstly they are easily cheated by domains which do not, at first sight, fulfil the restriction but may be rewritten such that they do. Also these restrictions are often too strict, such that even practically "trivial" problems fall outside the restrictions and into more general categories. This project therefore examines an alternate classification scheme based upon properties of the state graph instead. This approach is briefly mentioned by Bäckström and Jonsson (2011), however, they choose to continue working within the classical classification scheme. By basing the classification upon properties of the graph it is expected that the classification will be more robust against simple changes than pure syntactic classification.

1.1 Background

Automated planning, as mentioned earlier, is considered one of the cornerstones of modern AI. To fully appreciate the task of automated planning we should first understand its relation to AI. As a first approximation one might consider AI as the scientific pursuit of thinking machines, or agents as they are often called. However, merely thinking is not enough, it is also necessary to be able to express these thoughts. Therefore AI is often thought of as the pursuit of rationally acting agents, as expressed by Russell and Norvig (2003). However, there is more than one way for an agent to act rationally, and agents are often classified by the way in which they act or react to the environment. The class of agents we are interested in here is the class of proactive autonomous agents. A proactive agent is an agent which does not merely wait for some change in the environment, but actively seeks to manipulate the environment in order to achieve some desired goal. An autonomous agent is capable of acting by itself independent of any external controller, in this sense it is merely set in motion and then carries out its task on its own. Automated planning is a way to realise such agents by providing the proactive acting required, and by containing this system within the agent itself an autonomous agent is achieved.

Automated planning provides the means for agents to be proactive. This is accomplished by letting the agent perceive the surrounding environment, equipping the agent with a set of actions with which it can change the environment, and finally a goal which the agent attempts to achieve. Automated planning then attempts to plan which actions should be used upon the environment and in which order, so that the environment lives up to the goal of the agent.

1.2 Motivation

As with many other problems in the scope of AI, automated planning is a hard problem. In fact it is so hard that, theoretically, in many cases it is not possible to determine in any finite time whether a plan is possible. Despite this many planning problems are, in practice, easily solved. Attempts to explain this discrepancy have typically involved analysing different restrictions in the hopes of proving that certain subsets of automated planning are simpler than others and, most importantly, simpler than automated planning in general. Such results have actually been achieved, primarily by Bylander (1994) and Erol *et al.* (1995). However, as exposed in Bäckström and Jonsson (2011) these results are often considered lacklustre as the restrictions necessary to show the desired result are too restrictive to allow any practically interesting problems. In light of

this Bäckström and Jonsson (2011) decided to examine notions such as padded complexity and limited non-determinism.

Common for all these approaches is the use of syntactic limitations to create the necessary restrictions. Actually Bäckström and Jonsson (2011) mention another possibility, namely examining the state graph of the problem. They however, reject this based on work by Balcázar (1996) which failed to turn up any new restrictions which were simpler than those already known. The work of Balcázar (1996) was focused upon the notion of succinct representation, meaning that some information about the problem is thrown away at this point. This project therefore aims to examine the state graph of planning domains in order to find either completely new restrictions or refinements of previously known ones.

1.3 Structure of the Report

This report is structured in such a way that chapter 2 first introduces the two planning formalisms used and their associated notation. Chapter 3 then covers a set of known results regarding the computational complexity of automated planning when restricted syntactically, following this chapter 4 looks at similar results obtained by restricting the causal graph. Chapter 5 then moves on to examine what can be achieved by restricting the state graph. Finally chapter 6 will examine the broader consequences of the results obtained in the previous chapters.

All of this material assumes knowledge about at least automated planning, graphs, directed graphs and computational complexity. The reader should be familiar with these subjects.

Chapter 2

The Complexity of Planning

> Let your plans be dark and as impenetratable as night, and when you move, fall like a thunderbolt.
>
> - Sun Tzu

In order to discuss the complexity of automated planning, it is first necessary to define a planning problem and a planning domain. These two concepts are interrelated as every planning problem has an associated domain and every planning domain gives rise to a set of problems. As this report uses two planning formalisms, it is necessary to define both problems and domains for both formalisms. Thereafter the focus will shift to describe the computational problem which this report will deal with.

2.1 STRIPS Formalism

The Stanford Research Institute Problem Solver (STRIPS) formalism is the classical planning formalism, so much that it is also known as classical planning. In this report a restricted version of the formalism will be used. This restriction is called propositional STRIPS and disallows the use of variables in operators. These definitions are a variation of those from Ghallab *et al.* (2004).

Definition 2.1 A STRIPS planning domain is a two-tuple (P, O) where:

- P is a finite set of state propositions p_i.
- O is a finite set of operators $o = (b, e)$ such that:
  - $b = \text{pre}(o)$ is a set of propositions and negations thereof. These are the preconditions of o.
  - $e = \text{post}(o)$ is a set of propositions and negations thereof. These are the postconditions of o.
  - The preconditions b and postconditions e are disjoint:

$$b \cap e = \varnothing$$

(Ghallab *et al.*, 2004)

For each planning domain there exists a set of states, each of which is defined as follows:

Definition 2.2 A state s is a set of propositions, representing the propositions which are true in the state.

$$s \subseteq P$$

(Ghallab *et al.*, 2004)

Definition 2.3 The set of all states S is the powerset of the avaiable propositions.

$$S = 2^P$$

(Ghallab *et al.*, 2004)

A condition, such as those in the preconditions and postconditions is defined as follows:

Definition 2.4 A condition g is a set of propositions and negations of propositions. Such a condition is called satisfiable iff it does not contain both a proposition and its negation. A condition is satisfied by a state s if no positive propositions of g are missing in s and no negative propositions of g are in s.

$$s \models g \text{ iff } \{p | p \in P \wedge p \in g\} \subseteq s \wedge \{p | p \in P \wedge (\neg p) \in g\} \cap s = \varnothing$$

(Ghallab *et al.*, 2004)

The applicability and application of an operator may then be defined as:

Definition 2.5 An operator $o = (b, e)$ is applicable in a state s iff the preconditions of o are satisfied in the state s. The application of the operator o in the state s is then a new state $r = o(s)$ in which the positive propositions of the postcondition are added and the negative propositions removed.

$$r = o(s) = (s \cup \{p | p \in P \wedge p \in g\}) \setminus \{p | p \in P \wedge (\neg p) \in g\} \text{ iff } s \models b$$

(Ghallab *et al.*, 2004)

Using this a STRIPS problem is then defined as:

Definition 2.6 A STRIPS planning problem is a four-tuple (P, O, s_0, g) where:

- P, O is a STRIPS planning domain.
- s_0 is an initial state.
- g are the goal conditions (i.e. a set of propositions and their negations).

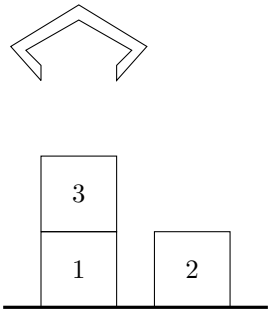
(Ghallab *et al.*, 2004)

Example 2.1 As an example of a STRIPS formalism of a planning problem. Consider the classical Blocks World problem. It consists of a number of blocks or boxes, a table and a gripper. A box may either be stacked upon another box, held in the gripper or stand directly upon the table. Upon each box at most one other box may be located. A couple of such situations may be seen in figure 2.1. In order to describe a blocks world instance it is first necessary to introduce the needed propositions, for n boxes these are:

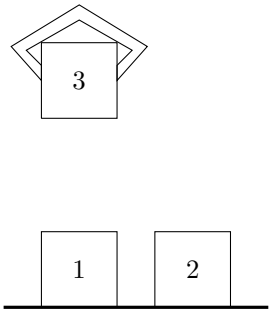
$$P = \bigcup \begin{array}{l} \bigcup_{i,j=1}^{n} \{\text{on}_{i,j}\} \\ \bigcup_{i=1}^{n} \{\text{held}_i, \text{free}_i, \text{on}_{i,\mathtt{t}}\} \\ \{\text{empty}\} \end{array} \tag{2.1}$$

The meaning of these propositions is as follows

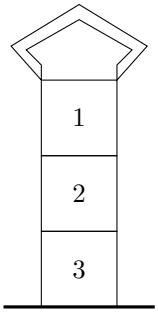
- $\text{on}_{i,j}$ - This proposition means that box i is placed on top of box j, unless j is `t` in which case i is placed on the table.
- held_i - This proposition means that box i is held by the gripper.
- free_i - This proposition means that no box is placed upon box i.



(a) The first (initial) state of the world.



(b) A second (resulting) state of the world.



(c) A third (final) state of the world.

Figure 2.1: Three simple planning states.

- empty - This proposition means that the gripper is empty.

The set of operators then becomes, for $i, j = 1, 2, \ldots n$:

PICKUP$_{i,j}$
Preconditions: $\text{free}_i, \text{on}_{i,j}, \text{empty}$
Postconditions: $\text{held}_i, \text{free}_j, \neg\text{free}_i, \neg\text{on}_{i,j}, \neg\text{empty}$

PICKUP$_{i,\mathtt{t}}$
Preconditions: $\text{free}_i, \text{on}_{i,\mathtt{t}}, \text{empty}$
Postconditions: $\text{held}_i, \neg\text{free}_i, \neg\text{on}_{i,\mathtt{t}}, \neg\text{empty}$

PUTDOWN$_{i,j}$
Preconditions: $\text{held}_i, \text{free}_j$
Postconditions: $\text{empty}, \text{free}_i, \text{on}_{i,j}, \neg\text{held}_i, \neg\text{free}_j$

PUTDOWN$_{i,\mathtt{t}}$
Preconditions: held_i
Postconditions: $\text{empty}, \text{free}_i, \neg\text{held}_i$

Each of these operators does as follows:

- PICKUP$_{i,j}$ - This picks up box i from box j or the table if j is $\mathtt{t}$.
- PUTDOWN$_{i,j}$ - This places box i on top of box j or the table if j is $\mathtt{t}$.

Using these propositions it is possible to describe the situation in figure 2.1a as the following state:

$$s_0 = \{\text{on}_{3,1}, \text{free}_2, \text{free}_3, \text{empty}\} \tag{2.2}$$

In this state it is possible to apply operator PICKUP$_{3,1}$, this results in the state seen in figure 2.1b. This state may be described by:

$$s' = \{\text{free}_1, \text{free}_2, \text{held}_3\} \tag{2.3}$$

Finally the goal state in figure 2.1c may be described by:

$$g = \{\text{on}_{1,2}, \text{on}_{2,3}, \text{empty}\} \tag{2.4}$$

Here it is possible to see how many propositions and operators may be necessary to describe a relatively simple problem.

2.2 SAS$^+$ Formalism

The Extended Simplified Action Structures (SAS$^+$) formalism is in many respects similar to the STRIPS formalism, however, instead of simple true/false propositions, SAS$^+$ makes use of multi-valued state variables. This introduces a set of domains from which the values for the state variables are drawn. The following definitions are a slight variation of those of Bäckström and Nebel (1995).

Definition 2.7 A SAS$^+$ planning domain is a three-tuple (V, D, O) where:

- V is a finite set $\{v_1, v_2, \ldots, v_n\}$ of state variables v_1, each with an associated finite domain D_i.
- D is the finite set of mutally exclusive finite domains of the variables. $D = \{D_1, D_2, \ldots, D_n\}$
- O is a finite set of operators $o = (b, e, f)$ such that:
  - $b = \mathrm{pre}\,(o)$ is a patial mapping of the state variables to their associated domains. These are the preconditions of o.
  - $e = \mathrm{post}\,(o)$ is a patial mapping of the state variables to their associated domains. These are the postconditions of o.
  - The preconditions b and postconditions e assign distinct values to shared variables:

$$\forall v \in (\mathrm{dom}\,(b) \cap \mathrm{dom}\,(e)) : b\,(v) \neq e\,(v)$$

(Bäckström and Nebel, 1995)

Some descriptions of the SAS$^+$ formalism include a third type of conditions, called prevailconditions, these are meant to a second set of preconditions such that the preconditions contain only variables also in the postcondition and the prevailconditions do not contain any variables also in the postconditions. The prevailconditions have been left out here as some definitions already leave them out, additionally they may trivially be recovered and are only a pain with respect to the notation in this report.

A state in SAS$^+$ may then be defined as a mapping:

Definition 2.8 A (total) state s is a mapping from each v_i to an element of its associated domain D_i. A partial state is a partial mapping, such that it maps a subset of the elements of V. (Bäckström and Nebel, 1995)

Definition 2.9 The set of all states S is the set of all total states.

$$S = D_1 \times D_2 \times \ldots \times D_n$$

(Bäckström and Nebel, 1995)

Conditions in SAS$^+$ are based upon partial states:

Definition 2.10 A condition g is a (partial) state. Such a condition is always satisfiable. A condition is satisfied by a given state s if s and g agree on every variable in the domain of g:

$$s \models g \text{ iff } \forall v \in \mathrm{dom}\,(g) : g\,(v) = s\,(v)$$

(Bäckström and Nebel, 1995)

The applicability and application of operators in SAS$^+$ then becomes:

Definition 2.11 An operator $o = (b, e, f)$ is applicable in a state s iff the preconditions of o are satisfied in the state s. The application of the operator o in the state s is then a new state $r = o\,(s)$ with the variables referenced in the postconditions changed.

$$r\,(v) = o\,(s)\,(v) = \begin{cases} e\,(v) & \text{iff } v \in \mathrm{dom}\,(e) \\ s\,(v) & \text{otherwise} \end{cases} \quad \text{iff } s \models b$$

(Bäckström and Nebel, 1995)

Using this we may define a SAS$^+$ problem:

Definition 2.12 A SAS$^+$ planning problem is a five-tuple (V, D, O, s_0, g) where:

- V, D, O is a SAS$^+$ planning domain.
- s_0 is a total initial state.
- g are the goal conditions.

(Bäckström and Nebel, 1995)

Example 2.2 To exemplify a SAS$^+$ planning instance, the Blocks World problem will be revisited. Again the situations of figure 2.1 will be used. For n boxes the needed variables and their domains are $i = 1, 2, \ldots n$:

$$\text{on}_i \in \{\texttt{none}, 1, 2, \ldots, n\} \tag{2.5}$$
$$\text{lowest}_i \in \{\texttt{yes}, \texttt{no}\} \tag{2.6}$$
$$\text{held} \in \{\texttt{none}, 1, 2, \ldots, n\} \tag{2.7}$$
$$\tag{2.8}$$

The meaning of these state variables is as follows

- on_i - The value of this state variable is the box which is placed on top of box i unless it is `none` in which case no box is on top of box i.
- lowest_i - The value of this state variable indicates whether the box i is directly on top of the table.
- held - The value of this state variable is the box which is held in the gripper unless it is `none` in which case the gripper is empty.

The set of operators then becomes, for $i, j = 1, 2, \ldots n$:

PICKUP$_{i,j}$
Preconditions: $\text{on}_i = \texttt{none}, \text{on}_j = i, \text{held} = \texttt{none}$
Postconditions: $\text{held} = i, \text{on}_j = \texttt{none}$

PICKUP$_{i,\texttt{t}}$
Preconditions: $\text{on}_i = \texttt{none}, \text{lowest}_j = \texttt{yes}, \text{held} = \texttt{none}$
Postconditions: $\text{held} = i, \text{lowest}_i = \texttt{no}$

PUTDOWN$_{i,j}$
Preconditions: $\text{held} = i, \text{on}_j = \texttt{none}$
Postconditions: $\text{held} = \texttt{none}, \text{on}_i = \texttt{none}, \text{on}_j = i$

PUTDOWN$_{i,\texttt{t}}$
Preconditions: $\text{held} = i$
Postconditions: $\text{held} = \texttt{none}, \text{on}_i = \texttt{none}, \text{lowest}_i = \texttt{yes}$

The function of these operators is the same as those of the STRIPS example. Using these variables and domains it is possible to describe the situation in figure 2.1a as the following state:

$$\begin{array}{rlrlrl} \text{on}_1 & = 3 & \text{on}_2 & = \texttt{none} & \text{on}_3 & = \texttt{none} \\ \text{lowest}_1 & = \texttt{yes} & \text{lowest}_2 & = \texttt{yes} & \text{lowest}_3 & = \texttt{no} \\ & & \text{held} & = \texttt{none} & & \end{array} \tag{2.9}$$

In this state it is possible to apply operator $\text{PICKUP}_{3,1}$, this results in the state seen in figure 2.1b. This state may be described by:

$$\begin{array}{rlrlrl} \text{on}_1 & = \texttt{none} & \text{on}_2 & = \texttt{none} & \text{on}_3 & = \texttt{none} \\ \text{lowest}_1 & = \texttt{yes} & \text{lowest}_2 & = \texttt{yes} & \text{lowest}_3 & = \texttt{no} \\ & & \text{held} & = 3 & & \end{array} \tag{2.10}$$

Finally the goal state in figure 2.1c may be described by:

$$\begin{array}{rlrlrl} \text{on}_1 & = \texttt{none} & \text{on}_2 & = 1 & \text{on}_3 & = 2 \\ \text{lowest}_1 & = \texttt{no} & \text{lowest}_2 & = \texttt{no} & \text{lowest}_3 & = \texttt{yes} \\ & & \text{held} & = \texttt{none} & & \end{array} \tag{2.11}$$

This example shows that it is possible to encode the same problem using fewer state variables in SAS^+ than STRIPS requires propositions.

One may quickly draw the conclusion that STRIPS planning is equivalent to SAS^+ planning with the domains of the variables limited to two values. This way every proposition in a STRIPS planning domain may be encoded in a SAS^+ variable with the domain $\{\texttt{True}, \texttt{False}\}$. This may lead to the idea that SAS^+ is more expressive than STRIPS. However, it has been shown by Bäckström (1995) that, in fact, STRIPS and SAS^+ have equal expressiveness.

2.3 Plans

Common to both of these formalisms is the concept of a plan:

Definition 2.13 A plan π is a sequence of operators.

$$\pi = \langle o_1, o_2, \ldots, o_m \rangle$$

(Folklore)

A complete plan may then be applied to a state:

Definition 2.14 A plan $\pi = \langle o_1, o_2, \ldots, o_m \rangle$ is applicable in a state s iff the first operator o_1 is applicable in s and the remaining plan $\langle o_2, \ldots, o_m \rangle$ is applicable in $o_1(s)$ (the empty plan is always applicable). The result of applying a plan to a state is a new state $r = \pi(s)$ being the result of applying each operator in turn.

$$r = \pi(s) = \langle o_1, o_2, \ldots, o_m \rangle (s) = \begin{cases} s & \text{iff } \pi = \langle\rangle \\ \langle o_2, \ldots, o_m \rangle (o_1(s)) & \text{otherwise} \end{cases}$$

(Folklore)

Definition 2.15 A plan π solves/satisfies a planning problem $\Pi = (\ldots, s_0, g)$ iff the result of applying the plan in the initial state satisfies the goal:

$$\pi \models \Pi \text{ iff } \pi(s_0) \models g$$

(Folklore)

2.4 Computational Problems

Based upon the above definition of a plan we may define two obvious decision problems for planning:

Definition 2.16 [PlanEx] The plan existence problem asks; given a planning problem Π, does there exist a plan π such that π solves Π. (Folklore)

Definition 2.17 [PlanLen] The plan length problem (occasionally called the bounded plan existence problem) asks; given a planning instance Π and a constant k, does there exist a plan π, of length at most k, such that π solves Π. (Folklore)

These two computational problems are valid for both STRIPS and SAS$^+$ planning, and are the main decision problems related to planning. They are therefore the main focus of the remainder of this report, with an emphasis upon the first problem. An interesting property of these two is the following, by Bäckström and Nebel (1995):

Theorem 2.18 *If there exists some polynomial time computable function $f(\Pi)$ which takes a planning instance as input and gives an upper bound on the shortest plan length, if a plan exists, then* PlanEx *is polynomial time reducable to* PlanLen.

$$\text{PlanEx} \leq_p \text{PlanLen}$$

(Bäckström and Nebel, 1995)

Proof. For a given planning instance Π if there exists a plan, there must exist a plan of length at most $f(\Pi)$. Therefore it is possible to reduce PlanEx(Π) to PlanLen($\Pi, f(\Pi)$). As $f(\Pi)$ is polynomial time computable, this is a polynomial time reduction. *Q.E.D.*

For both STRIPS and SAS$^+$ such a function may trivially be created:

Theorem 2.19 *There exists, for STRIPS planning, a polynomial time computable function upwards bounding the length of the shortest plan.*
(Bäckström and Nebel, 1995)

PROOF. Take the following function:

$$f(\Pi) = ||S|| = 2^{||P||}$$

For the first part the function is trivially polynomial time computable. For the second part, assume the shortest plan Π has length greater than $||S||$ then, by the pigeon hole principle, it must visit some state twice. This implies that the plan contains a cycle, by removing this cycle it is possible to create a shorter plan Π' which is a contradiction. Thus if a shortest plan exists, it must have a length $||\Pi|| \leq ||S||$ *Q.E.D.*

Theorem 2.20 *There exists, for SAS$^+$ planning, a polynomial time computable function upwards bounding the length of the shortest plan.*
(Bäckström and Nebel, 1995)

PROOF. Take the following function:

$$f(\Pi) = ||S|| = \prod_{i=1}^{n} ||D_i||$$

The rest of the proof is analogous to theorem 2.19. *Q.E.D.*

The complexity of an algorithm is measured against the size of the input given to the algorithm. For automated planning the size of the input is traditionally given in terms of the propositions/state variables and the operators in the problem domain, as an instance may be encoded in space polynomial in these.

Theorem 2.21 *A STRIPS planning problem $\Pi = (P, O, s_0, g)$ may encoded in $O(||P||\,||O||)$ bits.* *(Original)*

PROOF. A bitfield representation is used. First each proposition is a assigned a bit position. A state may then be encoded in $||P||$ bits by coding a 1 in the bit position for each proposition in the state. A condition may be encoded in $2\,||P||$ bits by dividing first coding the propositions in the condition as above, and then coding the negative propositions. To code the entire planning problem it is necessary to code the size of P, the size of O, the pre- and postconditions of the all operators, the initial state and the goal conditions. This requires $\lceil \lg(||P||) \rceil + \lceil \lg(||O||) \rceil + 2\,||O||\,||P|| + 3\,||P||$ bits. *Q.E.D.*

Theorem 2.22 *$S\,SAS^+$ planning problem $\Pi = (V, D, O, s_0, g)$ may encoded in $O\left(||V||\,||D||\,||O||\right)$ bits.* *(Original)*

Proof. Again a bitfield representation is used, however, each state variables may require more than one bit for representation. For each domain D_i let $f_i : D_i \mapsto \{0, 1, \ldots, ||D_i|| - 1\}$ be a bijection. This way the assignment of the variable v_i may be represented using $\lceil \lg\left(||D_i||\right)\rceil$ bits. A state may then be coded in $||V||\,\lceil \lg\left(||D_i||\right)\rceil$ bits. For conditions an additional don't care/not present marking is required for those variables which are not present in the domain of the conditions. Thus a condition is coded in $||V||\,\lceil \lg\left(||D_i|| + 1\right)\rceil$ bits. This way the entire problem instance may be coded in $\lceil \lg\left(||V||\right)\rceil + ||V||\,\lceil \lg\left(||D_i||\right)\rceil + 2\,||O||\,||V||\,\lceil \lg\left(||D_i|| + 1\right)\rceil + \lceil \lg\left(||D_i||\right)\rceil + \lceil \lg\left(||D_i|| + 1\right)\rceil$ *Q.E.D.*

It should be noted that, as the size of a planning problem is measured only in terms of the elements of the domain, it is possible to speak of the complexity of a planning domain. The remainder of the report will therefore focus on the complexity of planning domains, and not specific planning problems.

Chapter 3

Syntactic Restrictions

Tired minds don't plan well. Sleep first, plan later.

- Walter Reisch

Several complexity results are known for both the PlanEx (definition 2.16) and PlanLen (definition 2.17) problems for various restrictions of planning. This chapter will examine the complexity results which stem from syntactic restrictions of the basic planning problems. First the most recognised results for STRIPS planning will be examined. This will be followed by similar results for SAS^+.

Before any of that however, it is useful to know what is meant by a syntactic restriction. A syntactic restriction is, in essence, a restriction which may be imposed directly upon a planning instance and the objects it contains. This means that it is possible to restrict based upon things such as the predicates/state variables, operator conditions, the initial state and the goal conditions. It is not possible to restrict based upon any derived structures, such as the causal and/or state graphs covered later. In short if some information is not directly available in the planning instance, it is not possible to create a syntactic restriction based upon it.

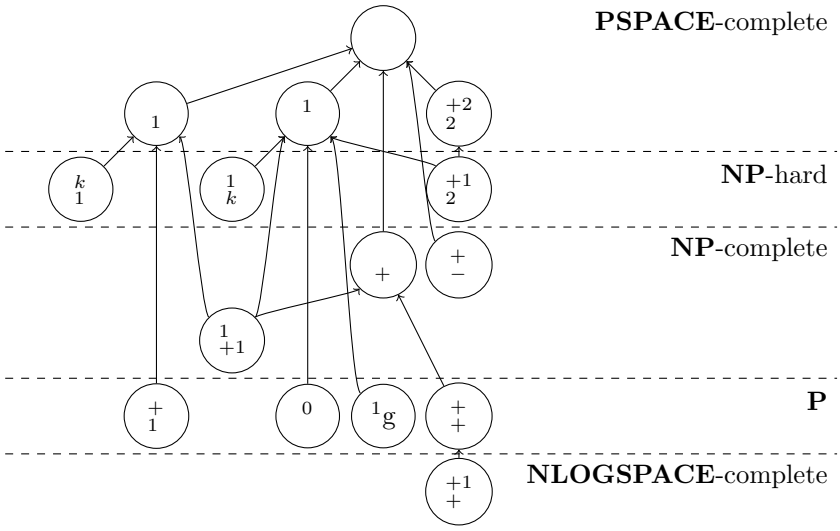


Figure 3.1: The relationship between the complexities of PlanEx for the various restrictions of STRIPS planning.
Arrows indicate that the head of the arrow is a generalisation of the restriction at the tail. Strictly speaking the **NP**-hard class has no upper bound, however, it is here shown to lie beneath **PSPACE**-complete. This is due to the fact that all of the problems show here are atmost **PSPACE**-complete.

Table 3.1: The complexity of PlanEx for the most common restrictions of propositional STRIPS.

Restriction	Pre	Post	Goals	Complexity
STRIPS				**PSPACE**-complete
STRIPS_1		1		**PSPACE**-complete
STRIPS^1	1			**PSPACE**-complete
STRIPS_2^{+2}	+2	2		**PSPACE**-complete
STRIPS_1^k	k	1		**NP**-hard
STRIPS_k^1	1	k		**NP**-hard
STRIPS_2^{+1}	+1	2		**NP**-hard
STRIPS_-^+	+	−		**NP**-complete
STRIPS_+		+		**NP**-complete
STRIPS_{+1}^1	1	+1		**NP**-complete
STRIPS_1^+	+	1		**P**
STRIPS^1g	1		k	**P**
STRIPS^0	0			**P**
STRIPS_+^+	+	+		**P**
STRIPS_+^{+1}	+1	+		**NL**-complete

3.1 STRIPS Restrictions

The majority of the results regarding the syntactic restriction of STRIPS planning are those of Bylander (1991), which have been slightly updated by Bylander (1994) himself, and those of Erol *et al.* (1995) most of which have been summarised by Ghallab *et al.* (2004). In general the restrictions limit the number of preconditions and postconditions of an operator, along with the mode of the predicates of those conditions. We will note a restriction of STRIPS using $\text{STRIPS}_{\gamma\beta}^{\delta\alpha}$ for a restriction where each operator has at most α (resp. β) preconditions (postconditions) which, optionally, are of mode δ (γ). If a k is present in place of α (β) then the number of precondition (postconditions) is merely bounded by a constant. Additionally the g suffix on one of the classes, means that the number of goal literals is bounded by a constant. Thus a STRIPS_1^{+2} problem is a restriction where each operator has at most two preconditions, all of which are positive, and a single postcondition of any mode. Starting with the PlanEx problem, these may be broken down as in table 3.1, and graphically in figure 3.1. Here it is possible to see both how they are distributed over the complexity classes as well as how they generalise each other.

For the PlanLen problem most of the complexities are the same due to theo-

Table 3.2: The complexity of PlanLen for the most common restrictions of propositional STRIPS.

Restriction	Pre	Post	Goals	Complexity
STRIPS^{+}_{1}	+	1		**NP**-complete
STRIPS^{+1}_{1}	+1	1		**NP**-complete
STRIPS^{0}	0			**NP**-complete
STRIPS^{0}_{2}	0	2		**NP**-complete
STRIPS^{0}_{1}	0	1		**P**
STRIPS^{0}_{+3}	0	+3		**NP**-complete
STRIPS^{0}_{+2}	0	+2		**P**

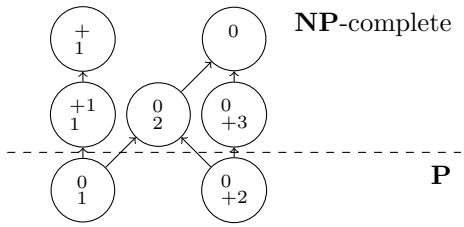


Figure 3.2: The relationship between the complexities of PlanLen for the various restrictions of STRIPS planning.
Arrows indicate that the head of the arrow is a generalisation of the restriction at the tail.

rem 2.18. However, the PlanLen problem is often harder than PlanEx, the restrictions in table 3.2 are all in **P** for PlanEx, but up to **NP**-complete for PlanLen.

Traditionally these complexity classes are split into two sets, those which are subsets of (or equal to) **P**, which are considered to be the tractable problems, and those which are (thought to be) strict supersets of **P**, which are considered intractable. The subset of problems which fall into the tractable catagory are the following:

A couple of things that are worth noting in figure 3.1. Firstly it is seen that the first single restriction to get planning to a lower complexity than **PSPACE**-completeness is to allow only positive postconditions. Not only that, but the only single restriction which is in fact tractable disallows any preconditions. This last indicates that preconditions are quite powerful, as it is necessary to either completely remove them, or combine them with other restrictions before planning becomes tractable. However, the general theme is that it necessary to heavily restrict planning before it even approaches tractability.

1. Plan existence
   (a) STRIPS_1^+
   (b) $\text{STRIPS}^1 g$
   (c) STRIPS^0
   (d) STRIPS_+^+
   (e) STRIPS_+^{+1}
2. Plan length
   (a) STRIPS_1^0
   (b) STRIPS_{+2}^0

These seven restrictions represent the points where planning, at least from an asymptotic point of view, becomes tractable. The majority of these are rather trivial cases. The only really interesting cases are the STRIPS_1^+ case and the $\text{STRIPS}^1 g$ case, as these include some interesting problems. Therefore it is interesting to examine what happens to the planning problems in these restrictions. The following sections will first review the existence of polynomial time algorithms for the STRIPS_1^+ case and then move on to examine the ways in which these restrictions may be "cheated".

3.1.1 STRIPS_1^+

By restricting planning to only positive preconditions and a single postcondition, it is possible to reduce the complexity of planning to be contained in **P**. A polynomial time algorithm for this restriction is given by Bylander (1994). The basic realisation needed is captured by the following theorem:

Theorem 3.1 *There exists a plan which solves a planning instance iff there exists a plan which solves the same planning instance which first applies all operators with positive postconditions and then applies any operators with negative postconditions.* *(Bylander, 1994)*

To prove this the following lemma is quite useful:

Lemma 3.2 *Given two adjacent operators in a plan it is possible to replace these operators with at most two operators, such that any operators with positive postconditions are applied first, followed by those with negative postconditions.* *(Bylander, 1994)*

Proof. Consider two adjacent operators o_1, o_2 in a plan. The only way these may violate lemma 3.2 is if o_1 has a single negative postcondition and o_2 has a single positive postcondition. Now consider applying these operators in a state s, resulting in a state r:

$$r = o_2\left(o_1\left(s\right)\right)$$
$$r = \left(s \setminus \text{post}_-\left(o_1\right)\right) \cup \text{post}_+\left(o_2\right)$$

If $\text{post}_-\left(o_1\right)$ and $\text{post}_+\left(o_2\right)$ are disjoint we may simply apply the operators in the opposite order:

$$r = \left(s \cup \text{post}_+\left(o_2\right)\right) \setminus \text{post}_-\left(o_1\right) \quad \text{assuming } \text{post}_+\left(o_2\right) \cap \text{post}_-\left(o_1\right) = \varnothing$$
$$r = o_1\left(o_2\left(s\right)\right)$$

If the effects are not disjoint they must be equal, as the sets have only a single member. In this case we may simply drop the first operator:

$$r = s \cup \text{post}_+\left(o_2\right) \qquad \text{assuming } \text{post}_+\left(o_2\right) = \text{post}_-\left(o_1\right)$$
$$r = o_2\left(s\right)$$

Thus given two adjacent operators o_1, o_2, either the original order (o_1, o_2), swapping the order (o_2, o_1) or dropping the first operator (o_2) satisfies lemma 3.2 and results in the same state. *Q.E.D.*

PROOF. The reverse direction of theorem 3.1 is trivial as any plan which first contains operators with positive postconditions and then any with negative postconditions is itself a plan. The forward direction of theorem 3.1 follows from lemma 3.2. Assume that π is a plan which solves the given planning instance, then by repeated application of lemma 3.2, it is possible to move all the operators with positive postconditions to the front of π thus giving a plan new plan π' which satisfies theorem 3.1 with the same resulting state as π. *Q.E.D.*

The result of theorem 3.1 is that it is only necessary to examine those plans which first apply operators with positive postconditions followed by applying operators with negative postconditions. Assume that π is such a plan. After applying the positive part of π to the initial state the planner would have reached a state $s_{\max}$, before and after which all states traversed will have fewer true prepositions. The algorithm of Bylander (1994) works by attempting to find such a state. This can be done in polynomial time.

3.1.2 Cheating the restriction

In order to fully appreciate why these syntactic restrictions are rather crude, the following will give some examples of constructions in the planning domain which may cause a given problem to fall outside a given sematic restriction despite being equivalent (at least with regard to PLANEX and PLANLEN) to a problem within the restriction. The first example given will attempt to cheat the positive precondition restrictions.

Example 3.1 Consider a planning domain with the following operators:

TURNON
Preconditions: ¬Backup, Off
Postconditions: ¬Off

TURNONBACKUP
Preconditions: Backup, Off
Postconditions: ¬Off

This kind of construction may arise naturally if there is more than one way to achieve the same effect. In this case there is a normal way to do something and then there is a backup in case the situation is not normal. It is clear that the above operators do not contain only positive preconditions due to the presence of the ¬Backup term, nor is there a way to replace these two operators to remove this term without making assumptions about the rest of the domain and/or introducing additional propositions. This means that any planning domain with these operators, can at most fall within the STRIPS_1 restriction

which is **PSPACE**-complete. However, for the PlanEx and PlanLen problems these may be replaced by a single operator as follows:

TurnOn∗
Preconditions: Off
Postconditions: ¬Off

This single operator clearly only has positive preconditions. This allows placing the planning domain within the STRIPS_1^{+1} restriction which is **NL**-complete, this is a very significant improvement. Not only that, but it is also obvious that this new operator is applicable iff one of the original operators is applicable. Also when it is applied it results in the same state as the original operator which is applicable in the original state. This implies a one-to-one relationship between plans with the new operator and plans with the original operators which preserves the length of the plans. This then implies that a plan of length k exists with the new operator iff there exists a plan of length k with the original operators.

While example 3.1 showed one construction which can affect the placement of a planning problem in the syntactic restriction hierarchy by affecting the mode of the preconditions, other natural constructions may have unfortunate effects on the complexity of a planning problem in similar ways.

Example 3.2 Now consider a planning domain with the following single operator:

TurnOn
Preconditions: ¬On
Postconditions: On

This may seem a very natural thing to do, if the thing is already on, there is no reason to turn it on again. However, the precondition causes the planning domain to fall within the STRIPS_{+1}^{1} restriction which is **NP**-complete. However, the following operator allows a better classification:

TurnOn′
Preconditions:
Postconditions: On

This single operator has no preconditions. This allows placing the planning domain within the STRIPS^0 restriction which is in **P**, again a significant improvement. The equivalence of these two operators, at least with respect to PlanEx and PlanLen, is trivial.

Of course these examples are designed to highlight the point in question and should not be taken as an indication that all planning domains can be rewritten to have lower complexity. Such a result would imply $\mathbf{P} = \mathbf{NP}$ or some other equally strong, but very unlikely, result. The point of these examples is to show how minor changes in the planning domain can have a huge effect on the complexity of automated planning.

3.2 SAS$^+$ Restrictions

The usual SAS$^+$ restrictions are somewhat more involved than the STRIPS restrictions. The set of restrictions which will be studied in this report are those of Bäckström and Nebel (1995). These are as follows:

P Post-uniqueness

For each possible state variable assignment, there is at most one operator whose post-condition includes this assignment.

$$\forall v_i \in V, x_i \in D_i : ||\{o \in O | \text{post}\,(o)\,(v_i) = x_i\}|| = 1$$

S Single-valuedness

For each state variable v_i there must exist some value $x_i \in D_i$ such that for all operators o, if v_i appears in the postcondition of o, it must either not appear in the preconditions of o or appear with the value x_i.

$$\forall v_i \in V : \exists x_i \in D_i : \forall o \in O :$$
$$(v_i \in \text{dom}\,(\text{post}\,(o))) \Rightarrow (v_i \notin \text{dom}\,(\text{pre}\,(o)) \wedge \text{pre}\,(o)\,(v_i) = x_i)$$

U Unariness

Every operator has exactly one postcondition.

$$\forall o \in O : ||\text{dom}\,(\text{post}\,(o))|| = 1$$

B Binariness

The size of the domain associated with each variable is exactly 2.

$$\forall D_i \in D : ||D_i|| = 2$$

To denote a SAS$^+$ planning instance with some restriction x applied this report will write SAS$^+$-x. If more than one restriction is applied, the specifications

Table 3.3: The complexity of the most common restrictions of SAS^+.

Restriction	Complexity	
	Plan existence	Plan length
SAS^+	**PSPACE**-complete	**PSPACE**-complete
SAS^+-P	Open	**NP**-hard
SAS^+-S	**PSPACE**-complete	**PSPACE**-complete
SAS^+-U	**PSPACE**-complete	**PSPACE**-complete
SAS^+-B	**PSPACE**-complete	**PSPACE**-complete
SAS^+-PS	Open	**NP**-hard
SAS^+-PU	Open	**NP**-hard
SAS^+-PB	Open	**NP**-hard
SAS^+-SU	**NP**-complete	**NP**-complete
SAS^+-SB	**PSPACE**-complete	**PSPACE**-complete
SAS^+-UB	**PSPACE**-complete	**PSPACE**-complete
SAS^+-PSU	**P**	**P**
SAS^+-PSB	Open	**NP**-hard
SAS^+-PUB	Open	**NP**-hard
SAS^+-SUB	**NP**-complete	**NP**-complete
SAS^+-PSUB	**P**	**P**

are simply appended such as SAS^+-xy. In this system a SAS^+-UB problem is a SAS^+ planning problem restricted to unariness and binariness. Unariness and binariness are rather natural restrictions whereas post-uniqueness and single-valuedness seem rather artificial. There is however, some reason behind the madness. A domain which is post-unique has at most one operator which sets a given state variable to a given variable. As a minimal example consider a single light switch connected to a light. There is only one way to turn on the light, flip the switch into the "on" position. This small domain is post-unique. Single-valuedness is somewhat related to the positive postcondition restrictions of STRIPS. However, single-valuedness does not require that operators which do not mention the variable in their preconditions do not change it to some odd value.

A systematic study of these restrictions was performed by Bäckström and Nebel (1995), the results of which are presented in table 3.3 and figure 3.3. Those plan existence problems which are marked as open are at least **NP**-hard and at most **PSPACE**-complete. Again it is seen that a great number of restrictions must be made before planning becomes tractable, as the planning domain must be at least post-unique, single-valued and unary.

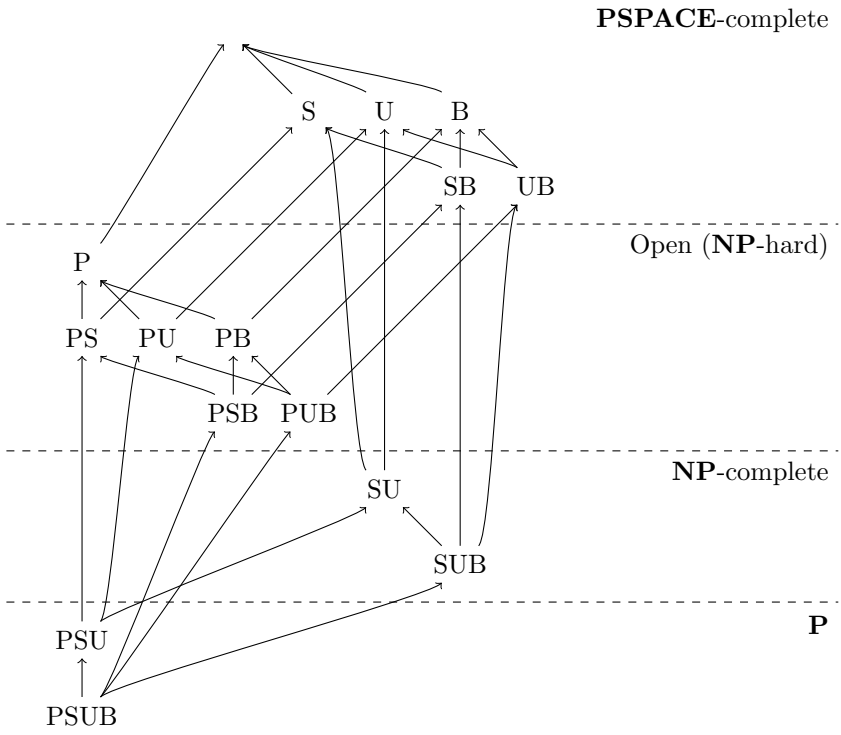


Figure 3.3: The relationship between the complexities of plan existence (plan length) for the various restrictions of SAS$^+$ planning.
Arrows indicate that the head of the arrow is a generalisation of the restriction at the tail.

From this it appears that post-uniqueness is a powerful restriction. Intuitively this power comes from the fact that, for every possible assignment of a state variable there is, at most, one operator which may create that assignment. This allows a "simple" backwards planning approach which allows tractable planning, when coupled with restrictions that guarantee polynomial length plans.

3.2.1 Cheating the Restriction

As for STRIPS planning, it is easy to devise constructions which cheat the various restrictions. The first example is similar to example 3.1 for STRIPS.

Example 3.3 Consider a planning domain, which for some $v_i, v_j \in V$ as well as all $x_i \in D_i$ and some $y_j \in D_j, y_j \neq x_i$ contains the following operator:

TRICK$_{x_i}$
Preconditions: $v_i = x_i$
Postconditions: $v_j = y_j$

As in the case of example 3.1, these may be replaced by a single operator while preserving plan existence and length:

NOTRICK
Preconditions:
Postconditions: $v_j = y_j$

The original set of operators prevents any problem instance including the operators from being post-unique as more than one operator includes the $v_j = y_j$ postcondition. As post-uniqueness is the most powerful restriction, this is unfortunate. Additionally if $v_i = v_j$ these operators also prevents the instance from being single-valued.

This shows how it is possible to cheat both the post-uniqueness and single-valuedness restrictions.

3.3 Results on Syntactic Restrictions

The preceding chapter has covered many of the syntactic restrictions which have been examined from a complexity theoretic point of view. As is rather obvious these restrictions are very restrictive and hardly allow any real-world problems

to be modeled. Even then, though they are severely restricted, very few of the restrictions actually achieve tractability.

Chapter 4

The Causal Graph

Plans are only good intentions unless they immediately degenerate into hard work.

- Peter Drucker

The first graph which will be examined for automated planning is the causal graph. As the work on causal graphs uses a state variable model, this chapter will use the SAS$^+$ formalism. The first step is to define the causal graph:

Definition 4.1 For a SAS$^+$ planning domain (V, D, O), the causal graph is the digraph $G_c = (V, A)$ st. $(u, v) \in A$ iff the following conditions hold:

$$u \neq v$$
$$\exists o \in O : v \in \mathrm{dom}\left(\mathrm{post}\left(o\right)\right) \wedge u \in \left(\mathrm{dom}\left(\mathrm{post}\left(o\right)\right) \cup \mathrm{dom}\left(\mathrm{pre}\left(o\right)\right)\right)$$

(Helmert, 2004)

The intuition behind the causal graph is this: There exists an arc between two variables iff a change in the variable at the tail of an arc may cause a change in the variable at the head of the arc. Causal graphs have originally been studied as a tool for heuristic search and for the generation of heuristics. However, some complexity results are known of planning problems with restricted causal graphs, the simplest of which are chain causal graphs.

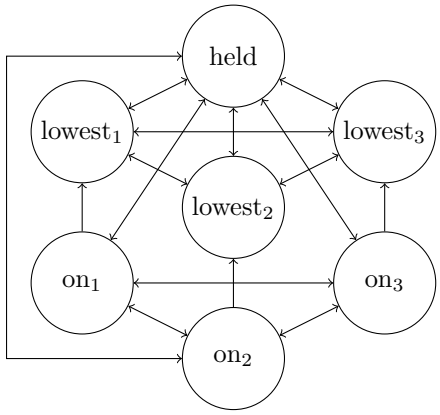


Figure 4.1: The causal graph for example 4.1.

Example 4.1 As an example, one might consider the blocks world planning domain from example 2.2 with 3 blocks. The causal graph for this domain is seen in figure 4.1.

4.1 Chain Causal Graphs

One class of causal graphs are is the class of chain causal graphs, denoted $\mathbb{C}_n$. The definition of chain causal graphs is as follows:

Definition 4.2 A planning problem Π is in the class $\mathbb{C}_n$ iff the causal graph of Π is a directed path. (Domshlak and Dinitz, 2001)

Theorem 4.3 *All planning problems in the class $\mathbb{C}_N$ are unary.* *(Helmert, 2004)*

From the definition of the class, the above corollary is obvious. This means that the earlier results from the unariness syntactic restriction also apply here. However, the problem is still not significantly simpler:

Theorem 4.4 PlanEx *for instances in the class $\mathbb{C}_n$ is* **NP***-hard.* *(Giménez and Jonsson, 2008)*

This has been shown by Giménez and Jonsson (2008) by way of a reduction from CNFSat thus showing **NP**-hardness. This leaves the planning problem

intractable, however, if one further restrict the planning problems, this time by the size of the variable domains, it is possible achieve a tractable class. Giménez and Jonsson (2009) define the class $\mathbb{C}_n^k$:

Definition 4.5 A planning problem Π is in the class $\mathbb{C}_n^k$ iff the following two conditions hold.

$$\Pi \in \mathbb{C}_n$$
$$\forall v_i \in V : ||D_i|| \leq k$$

(Giménez and Jonsson, 2009)

The complexity of planning problems in the class $\mathbb{C}_n^k$ varies from **P** to **NP**-hard depending on the k parameter, and includes two restrictions of which the complexity is, as yet, unknown.

Theorem 4.6 *The complexity of* PlanEx *for planning problems in the class* $\mathbb{C}_n^k$ *is:*

P	*for* $k = 2$
Open	*for* $3 \leq k \leq 4$
NP-*hard*	*for* $5 \leq k$

(Brafman and Domshlak, 2003),(Giménez and Jonsson, 2009)

Brafman and Domshlak (2003) demonstrate a polynomial time algorithm for planning in the class $\mathbb{C}_n^2$, while Giménez and Jonsson (2009) give a reduction from CNFSat to PlanEx for the class $\mathbb{C}_n^5$. This leaves open the complexity of the two intermediate classes. It is interesting here to note that any problem in the class $\mathbb{C}_n^2$ is implicitly both unary and binary, however, it does not require post-uniqueness or single-valuedness. This means that, although the general SAS$^+$-UB restriction is **PSPACE**-complete, the additional information in $\mathbb{C}_n^2$ allows it to be in **P**. This illustrates that the additional information in the causal graph allows a finer distinction between various restrictions.

4.2 Polytree Causal Graphs

Another class of causal graphs are those whose underlying undirected graph is acyclic. These are called polytree causal graphs. This class was first defined by

Brafman and Domshlak (2003) however, the definition here is due to Giménez and Jonsson (2009):

Definition 4.7 Let the underlying undirected graph $U(G) = U((V, A)) = (V, E)$, of a digraph g, be defined as follows:

$$E = \left\{(u, v) \in V^2 \middle| \{(u, v), (v, u)\} \cap A \neq \varnothing\right\}$$

(Giménez and Jonsson, 2009)

Definition 4.8 A planning problem is in the class $\mathbb{P}^1$ iff the undirected graph underlying the causal graph is acyclic. (Original)

In fact the class $\mathbb{C}_n$ is a subset of $\mathbb{P}$.

Theorem 4.9 *Every planning problem in $\mathbb{C}_n$ is also in $\mathbb{P}$.* *(Original)*

Proof. Consider a chain causal graph G i.e. a directed path. The undirected graph underlying this causal graph is merely a simple path and therefore acyclic. The implies that any problem in $\mathbb{C}_n$ is also in $\mathbb{P}$. *Q.E.D.*

Brafman and Domshlak (2003) have again shown a polynomial time algorithm for PlanEx for the case where the problem is binary and additionally the indegree is bounded. The **NP**-hardness result of Giménez and Jonsson (2009) trivially carries over. Additionally Giménez and Jonsson (2008) show that PlanEx is **NP**-complete for unbounded indegree.

4.3 Cheating the Causal Graph

Cheating the restrictions based on the causal graph, is a bit more difficult than for the simple syntactic restrictions of chapter 3. The example here is somewhat contrived and is based on the idea that the value of one variable can be inferred from another.

Example 4.2 Consider a SAS$^+$ planning domain with two variables:

$$\text{State} \in \{\texttt{Off}, \texttt{On}\}$$
$$\text{FullState} \in \{\texttt{Off}, \texttt{Low}, \texttt{High}\}$$

[1] This should not be confused with the complexity class **P**.

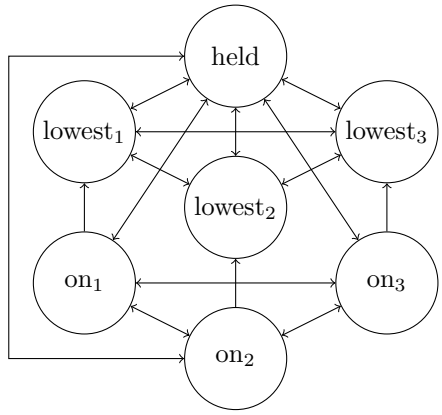


Figure 4.2: The causal graph for example 4.2.

Now in this case it is desired to maintain the following invariant:

$$\text{State} = \begin{cases} \texttt{Off} & \text{iff FullState} = \texttt{Off} \\ \texttt{On} & \text{otherwise} \end{cases}$$

In order to enforce this invariant it is necessary to change the value of State every time FullState is changed to or from `Off`. This implies that they both occur in the postcondition of the same operator, thus creating a cycle in the causal graph between these two variables. This prevents the problem from having a polytree causal graph. [TODO: Check this out!]

It is however, possible to remove the State variable from the domain. Every operator which contains State = `Off` as a precondition is replaced by one which has FullState = `Off` instead, and every which contains State = `On` as a precondition is replaced by two operators, one which has FullState = `Low` and one with FullState = `High`.

This shows that, while harder, it is still possible to cheat the restrictions based upon the causal graph.

Chapter 5

The State Graph

Adventure is just bad planning.

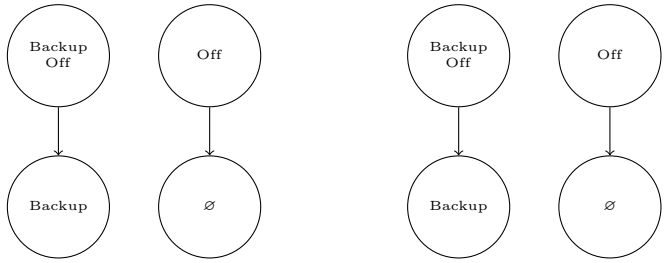
- Roald Amundsen

Chapter 3 examined the effect of syntactic restriction on the complexity of automated planning. It was shown that, apart from often being very restrictive, these restrictions can easily be cheated by constructions in the planning domain. Chapter 4 then went on to demonstrate some of the power of using the causal graph in order to restrict automated planning. These restrictions were somewhat harder to cheat than pure syntactic restrictions. This leads to the final graph which will be examined in the report, the state graph:

Definition 5.1 For some planning domain with states S and operators O, the state graph is a digraph $G_s = (V, A)$, such that:

$$V = S$$
$$A = \left\{(u, v) \in V^2 | : u \neq v \wedge \exists o \in O : v = o\left(u\right)\right\}$$

(Folklore)



(a) The state graph for the original operators of example 5.1.

(b) The state graph for the new operator of example 5.1.

Figure 5.1: The state graphs for example 5.1.

The definition applies to both STRIPS and SAS$^+$ formalisms. This chapter will proceed by first motivating the use of the state graph as a structure to express restrictions upon automated planning. It will then move on to grab some of the low hanging fruit in this setting before moving on to show a more involved result.

Example 5.1 Consider the planning problem consisting of the first the two original operators from example 3.1 and then the single new operator, along with the predicates $\{\text{Backup}, \text{Off}\}$. The state graphs for these are shown in figure 5.1. These are seen to be identical, this indicates that it should not be possible for a restriction, based upon the state graph, to distinguish between these two. This example illustrates that, using the state graph, it may be harder to cheat the restrictions compared to syntactic and causal graph based restrictions.

5.1 Basic Restrictions

The first restrictions which will be examined are simple graph theoretical restrictions which attempt to limit the number of nodes a search procedure must visit.

Property 5.2 *Planning a planning problem; if there exists a plan, then there exists a plan of polynomially (in the size of the planning problem) bounded length.*

Using just this single property, it is possible to reduce the complexity of both PLANLEN and PLANEX:

Theorem 5.3 PLANLEN, *for planning domains which satisfy property 5.2, is* **NP***-complete.* *(Original)*

PROOF. By property 5.2, if there exists a path then there exists a path of polynomial length. This path can then be found nondeterministically in polynomial time. Thus the problem is in **NP**.

To show that plan existence for this restriction is **NP**-hard we perform a polynomial many-one reduction from 3-CNFSAT to PLANLEN in a STRIPS planning domain which satisfies the above properties. Consider a 3-CNFSAT instance with clauses $C_1, \ldots, C_m$ and variables $x_1, \ldots, x_n$. For each clause C_i a predicate SAT_i is created to represent that clause C_i has been satisfied. For each variable two predicates T_i and F_i are created to represent that variable x_i has been assigned true or false respectively. For each variable x_i in the clause C_j one of the following operators is generated. The first operator is generated if x_i appears non-negated in C_j, representing that C_j is satisfied by making x_i true. Otherwise the second is generated, representing that C_j is satisfied by making x_i false.

ASSIGN$\mathrm{T}_{i,j}$
Preconditions: $\neg\mathrm{F}_i, \neg\mathrm{SAT}_j$
Postconditions: $\mathrm{SAT}_j, \mathrm{T}_i$

ASSIGN$\mathrm{F}_{i,j}$
Preconditions: $\neg\mathrm{T}_i, \neg\mathrm{SAT}_j$
Postconditions: $\mathrm{SAT}_j, \mathrm{F}_i$

The initial state s_0 is then the empty state, the goal condition is $\{\mathrm{SAT}_1, \ldots, \mathrm{SAT}_\mathrm{m}\}$ and the plan length bound is nm. This construction creates $m + 2n$ predicates and $3m$ operators, whereby it can be performed in polynomial time. The shortest path possible in this domain is of length at most nm. Assume that the shortest path has a length greater than nm then by the pigeonhole principle some operator must appear twice in the plan. The second application of this operator cannot have any effect on the state, as there are no negative postconditions in the domain. Thus a shorter plan can be created by leaving out the second application of the operator. This is a contradiction, thus there cannot exist a shortest plan of length greater than nm which is also polynomial, thus satisfying property 5.2.

If the original 3-CNFSAT instance accepts then there exists an assignment of true/false values to the variables $x_1, \ldots, x_n$. It is then possible to construct a plan from these assignments. If true (resp. false) is assigned to x_i then for every clause C_j in which x_i appears non-negated (negated) the operator ASSIGN$\mathrm{T}_{i,j}$ (ASSIGN$\mathrm{F}_{i,j}$) is added to the plan, if C_j has not already had an operator added to the plan. This plan is valid as only one of the operators is used for each variable x_i thus only one of the predicates T_i, F_i becomes true. This generates a plan of length m as only a single operator is applied for each clause, this trivially satisfies the nm bound. Also for each clause C_j exactly one ASSIGN$\mathrm{T}_{i,j}$ or ASSIGN$\mathrm{F}_{i,j}$ operator must be in the plan, thus each SAT_j predicates must be satisfied. Therefore if the original 3-CNFSAT instance accepts then the constructed PLANLEN instance accepts.

Any satisfying plan for the constructed plan existence problem must for each variable x_i use only ASSIGN$\mathrm{T}_{i,j}$ or ASSIGN$\mathrm{F}_{i,j}$ operators as the preconditions and effects ensure that these are mutually exclusive. Also there must be at least ASSIGN$\mathrm{T}_{i,j}$ or ASSIGN$\mathrm{F}_{i,j}$ operator for each C_j in order to satisfy the goal description. Therefore a satisfying assignment can be constructed from the final state s_g. If T_i (F_i) is in s_g then true (false) is assigned to x_i. If, for some x_i neither T_i nor F_i appear in s_g then it is assigned true or false arbitrarily as all clauses have satisfied without using the variable. Thus if the PLANLEN problem accepts then the original 3-CNFSAT instance accepts. Thus this is a valid many-one reduction.

By the combination of the problem being both in **NP** and **NP**-hard, the PLANLEN problem is **NP**-complete under property 5.2. *Q.E.D.*

Corollary 5.4 PLANEX*, for planning domains which satisfy property 5.2, is* **NP***-complete.* *(Original)*

This first property is one of the most essential in this chapter as almost every restriction from hereon relies on the fact that having plans of polynomially bounded length makes the problem at most **NP**-complete. The next property which will be examined is the following:

Property 5.5 *Given a state graph $G = (S, E)$. Every breadth first tree has polynomially bounded width.*

By ensuring both properties 5.2 and 5.5 it is possible to perform automated planning in polynomial time:

Theorem 5.6 PLANLEN*, for planning domains which satisfy properties 5.2 and 5.5, is in* **P***.* *(Original)*

PROOF. By property 5.2 it is only necessary to look for plans of a length up to some polynomial of the input size, let this polynomial be $p(x)$. To find the shortest plan, one simply performs a breadth first search, which terminates once the depth of the search has reached $p(x)$. By property 5.5 this search has polynomially bounded width, let this polynomial be $q(x)$. The search therefore at most examines $p(x)\,q(x)$ states, which is itself a polynomial. As breath first search is linear in the number of states visited, this algorithm works in polynomial time. Once the shortest path (if any) is found, this is compared to the bound given. *Q.E.D.*

Corollary 5.7 PLANEX*, for planning domains which satisfy properties 5.2 and 5.5, is in* **P**. *(Original)*

This restriction is a simple, if non-intuitive, way to realise polynomial time planning in a restriction based upon the state graph. Next some of the easier results about state graph restrictions will be shown.

5.2 Initial Restrictions

Consider the class of planning problems in which the state graphs consists of disjoint directed paths of, at most, polynomial length.

Theorem 5.8 PLANEX *and* PLANLEN *for planning problems whose state graph is the union of disjoint directed paths of, at most, polynomial length is in* **P**. *(Original)*

PROOF. Due to the polynomial length bound property 5.2 holds. Additionally, as each component of the graph is a directed path, all possible trees in the graph must be paths. This includes the breath first trees which then have width 1. This means that property 5.5 holds. Therefore the results of theorem 5.6 and corollary 5.7 hold. *Q.E.D.*

This is very much a toy example, as the only planning domains for which this holds are those where there, in each state, is at most one operator to choose, and there are never any cycles or exponentially long plans. However, it is possible to expand upon this. Consider a state graph which is the union of disjoint graphs, each of which is of polynomially bounded size.

Theorem 5.9 PLANEX *and* PLANLEN *for planning problems whose state graph is the union of disjoint subgraphs, each of, at most, polynomial size is in* **P**. *(Original)*

Proof. As each disjoint subgraph is of polynomial size, the shortest path between any two connected vertices must also be of polynomial length, thus property 5.2 holds. Additionally, as each component of the graph is of polynomial size, all possible trees in the graph are also of polynomial size and therefore, at most, polynomial width. This again includes the breath first trees, thus property 5.5 holds. Again the results of theorem 5.6 and corollary 5.7 therefore hold. *Q.E.D.*

This gives a somewhat wider restriction while remaining in **P**. One example of planning instances which fall within this restriction are those with very few fluents, in fact only logarithmically few fluents. A fluent is any STRIPS proposition which occurs in the postconditions of an operator.

Definition 5.10 A proposition p is called a fluent if it occurs in the postconditions of an operator:

$$p \text{ is a fluent} \Leftrightarrow \exists o \in O : p \in \text{post}(o) \wedge \neg p \in \text{post}(o) \tag{5.1}$$

(Ghallab *et al.*, 2004)

This is due to every non-fluent splitting the state graph into two disjoint subgraphs. This means that with only logarithmically few fluents, the state graph becomes a union of exponentially many disjoint subgraphs. As the vertices of these subgraphs differ only in the fluents, the subgraphs must be of polynomial size.

5.3 Ordering Restrictions

For the second set of restrictions it is necessary to introduce a total preordering $\preceq \subseteq S^2$ over the states of the planning domain. This preordering induces a equivalence relation $\asymp \subseteq S^2$ defined as follows:

$$a \preceq b \wedge b \preceq a \Leftrightarrow a \asymp b$$

Theorem 5.11 *$\asymp$ is an equivalence relation.* *(Original)*

Proof. For $\asymp$ to be an equivalence relation it must be reflexive, symmetric and transitive. It is trivially reflexive (resp. transitive) by the reflexivity (resp. transitivity) of $\preceq$. It is symmetric by the commutability of conjunction. *Q.E.D.*

Such an equivalence relation naturally gives raise to a set of equivalence classes. These are part of the restriction defined by the following two properties.

Property 5.12 *Given a state graph $S_g = (S, A)$ and a total preordering $\preceq$. It must be the case that:*

$$(u, v) \in A \Rightarrow u \succ v \Leftrightarrow \neg (u \preceq v)$$

Property 5.13 *Given a total preordering $\preceq$, it induces only polynomially many equivalence classes.*

One natural preorder, for STRIPS, which satisfies property 5.13 is the ordering of the size of the states.

Lemma 5.14 *Property 5.13 is satisfied by the total preorder:*

$$a \preceq b \Leftrightarrow ||a|| \leq ||b|| \quad \textit{for } (a, b) \in S^2$$

(Original)

Proof. First to prove that $\preceq$ actually is a total preorder it must be transitive and total. Both of these follow from the transitivity and totality of $\leq$. The size of a state s is bounded by the number of predicates in the problem domain:

$$0 \leq ||s|| \leq ||P|| \tag{5.2}$$

From this it follows that there are only $||P|| + 1$ possible sizes for a state, and thus that many possible equivalence relations. *Q.E.D.*

Corollary 5.15 *Property 5.13 is satisfied by the total preorder:*

$$a \preceq b \Leftrightarrow ||a|| \geq ||b|| \quad \textit{for } (a, b) \in S^2$$

(Original)

With these properties 5.12 and 5.13 it is possible to define another restriction of planning:

Theorem 5.16 PlanLen *for planning problems whose state graph satisfies property 5.12 using a total preorder which also satisfies 5.13 is* **NP***-complete.*
(Original)

PROOF. To show PLANLEN is in **NP** property 5.12 is used. This ensures that it is only possible to transition to states which precede the current state in the total order, by property 5.13 there are only polynomially many of these. Therefore property 5.2 is satisfied and the result of theorem 5.3 applies.

To then show that PLANLEN is also **NP**-complete the construction from theorem 5.3 is reused and shown to satisfy properties 5.12 and 5.13. The total order used is the one from corollary 5.15 which has been shown to satisfy 5.13. To show that property 5.12 is satisfied, it should be noted that the postconditions of all operators are positive. Thus whenever an operator is applied the state grows in size, thus satisfying property 5.12. *Q.E.D.*

Corollary 5.17 PLANEX *for planning problems whose state graph satisfies property 5.12 using a total preorder which also satisfies 5.13 is* **NP***-complete.* *(Original)*

Unlike the restriction of theorem 5.8 this restriction includes some of the restrictions from chapter 3, in this case the STRIPS_+ restriction (and, of course, any restrictions which are specialisations thereof).

Theorem 5.18 *The state graph of any STRIPS_+ problem satisfies property 5.12 using a total preorder which also satisfies 5.13.* *(Original)*

PROOF. Again the preorder used is that of corollary 5.15, it is therefore only necessary to show that property 5.12 holds. As the domain, by definition, only has positive postconditions, any transition is from a smaller state to a larger state. Therefore property 5.12 holds. *Q.E.D.*

5.4 Cheating the State Graph

The example require for cheating the state graph restrictions is the most involved example in this report. Essentially the state graph contains all information about the planning problem, therefore any change is visible in the state graph. However, the above restrictions are defined in such a way that they are independent of the actual operators which are applied and the contents of the states in which they are applied. This eliminates the possibility of simply splitting such as in examples 3.2, 3.1 and 3.3, or introducing redundant variables such as example 4.2. In order to cheat the state graph it is necessary to introduce redundant edges into the state graph which cause it to fall outside the above categories, such as a cycle. However, introducing a new edge means

that new transitions are available in the state graph, which means that entirely new states are reachable or old states are reachable in fewer steps. The following is one example of how such edges may be introduced, it is however, highly contrived.

Example 5.2 Consider a STRIPS planning domain with a special predicate x which is known a priori to never appear in the initial state or goal condition. In addition to this the following two operators exist and are the only place x is referenced:

TurnOn
Preconditions:
Postconditions: x

TurnOff
Preconditions:
Postconditions: ¬x

These two operators introduce a lot of cycles. This prevents at least the restriction of theorem 5.16 from working as no preorder can satisfy property 5.12. However, it is obvious that x, along with the two operators may simply be dropped from the domain as they serve no function.

Chapter 6

Discussion

A goal without a plan is just a wish.

- Antoine de Saint-Exupery

The previous chapters have shown that it is possible to create restrictions which limit the computational complexity of automated planning. It has also been shown how these restrictions may be cheated by constructions which cause a planning domain to fall outside any restrictions despite the existence of equivalent domains within a restriction. It has indirectly been postulated that such cheats are easiest to run into in the simple syntactic restrictions, while causal graph restrictions are harder to cheat and state graph restrictions are the hardest to cheat. This is exemplified by the example cheats becoming progressively more contrived and restrictive as the chapters have unfolded.

What has not been discussed is the ease with which a planning domain may be either determined and/or engineered to fulfil a certain restriction. Determining whether a given domain falls within some restriction is important for determining which planning algorithm should be used to solve a given instance. However, equally important, is being able to engineer a problem domain to fall within a given restriction. As especially seen for syntactic restrictions (chapter 3) the membership of a restriction may critically depend on how certain aspects of the domain are modelled. If tractable planning is desired, which it essentially always is, it is necessary to formulate the domain within one of the restrictions. This is greatly simplified by having restrictions which are intuitively understandable.

Of course once the planning domain is known, most planning problems, including PlanEx and PlanLen, become constant time computable (Ghallab *et al.*, 2004) simply by looking up the initial state and goal state in a (exponentially big) lookup table. Of course as such a lookup table is exponential in size, it is often not realistic to do so.

The previous chapters do give some indication as to the difficulty of developing an intuitive idea of the problems included in a given restriction. The first of these are found in the text surrounding the actual restrictions. Here it is clear that for some restrictions it is significantly easier to describe what kind of planning domains fall within a given restriction. In fact for syntactic restrictions this is practically self-evident, where, in the other extreme, the intuitive descriptions for the state graph restrictions are rather abstract. The other hint comes from the ease with which the cheating examples are constructed. The more construed these examples are, the harder, in general, it is to engineer a planning domain to fulfil the restriction. Both of these indicators point to the state graph restrictions being the hardest to work with, syntactic restrictions the easiest, and causal graph restrictions lying somewhere in between.

Another aspect which has not been discussed in the preceding chapters, is the use of these restrictions in heuristic planning. Both the original Heuristic Search Planner (HSP) (Bonet and Geffner, 2001) and the slightly later Fast Forward (FF) (Hoffmann and Nebel, 2001) planning systems rely on solving, or partially solving, a simplified version of the original problem to heuristically guide the search for a proper solution. In fact the causal graph was used by Helmert (2004) to develop a planning heuristic later used in the Fast Downward (FD) planning system (Helmert, 2006) based upon the FF planning system. The concept of these systems is to derive a heuristic for planning automatically by taking the original planning domain and the removing features such that the resulting domain is tractable. This process is called relaxing. A common approach to relaxing a problem is to drop any negative postconditions from STRIPS domains, this means that the resulting domain is in the STRIPS_+ which, though not quite tractable, is at least easier than unrestricted planning. It has not been examined whether the state graph restrictions herein, or found subsequently, may be used in a similar manner. However, it is conjectured here that this will be hampered by the state graph being exponential in size making it hard to computationally verify that the required restrictions are satisfied. Of course it may be possible to express restrictions based on attributes of the state graph which may be computed efficiently on a succinct representation of the state graph. This is in general beyond the scope of the project.

6.1 Further Work

From this point several interesting avenues of future research present themselves. Two of the obvious ones have been pointed out already above. The first is to find more restrictions based upon the state graph and the other is to determine whether some of these state graph restrictions may be used to guide a heuristic search.

Another area which may be studied are the so called Interaction Networks introduced recently by Chen and Giménez (2010). These networks are closely related to the causal graphs but carry some additional information. Already it is known that some problems which cannot be classified as tractable using causal graphs may be classified as tractable using interaction networks (Chen and Giménez, 2010) in particular STRIPS planning without preconditions. Here again it may be worthwhile to find additional restrictions and, given the success of the FD planning system and its causal graph based heuristic, examine whether it is possible to create a more accurate heuristic using the interaction network.

Chapter 7

Conclusion

He who fails to plan, plans to fail.

- Proverb

This project set out with the goal of expanding both the set of currently known restrictions of automated planning and the ways in which such restrictions are found. It was decided to do this by examining the state graph of planning domains. This had the advantage of the results being applicable to a large range of planning problems despite differences in formalism, as most formalisms have a natural mapping to a state graph. Not only this, but the resulting restrictions seem to be more robust against trivial changes to the problem domain's formulation.

The actual results consist of two abstract but not particularly useful properties which achieve **NP**-complete and **P** complexities. These two properties are then used to show two more concrete properties which also lead to restrictions with **NP**-complete and **P** complexities.

Thus it has been shown that not only is it possible to define restrictions of automated planning based upon the state graph, but that such restrictions have advantages compared to simple syntactic restrictions and causal graph restrictions.

List of Figures

List of Tables

Bibliography

Christer Bäckström. Expressive equivalence of planning formalisms. *Artif. Intell.*, 76(1-2):17–34, 1995.

Christer Bäckström and Peter Jonsson. All PSPACE-Complete Planning Problems Are Equal but Some Are More Equal than Others. In Daniel Borrajo, Maxim Likhachev, and Carlos Linares López, editors, *SOCS*. AAAI Press, 2011.

Christer Bäckström and Bernhard Nebel. Complexity Results for SAS+ Planning. *Computational Intelligence*, 11:625–656, 1995.

José L. Balcázar. The Complexity of Searching Implicit Graphs. *Artif. Intell.*, 86(1):171–188, 1996.

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artif. Intell.*, 129 (1-2):5–33, 2001.

Ronen I. Brafman and Carmel Domshlak. Structure and Complexity in Planning with Unary Operators. *J. Artif. Intell. Res. (JAIR)*, 18:315–349, 2003.

Tom Bylander. Complexity results for planning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, volume 1, pages 274–279, 1991.

Tom Bylander. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

Hubie Chen and Omer Giménez. Causal graphs and structurally restricted planning. *J. Comput. Syst. Sci.*, 76(7):579–592, November 2010. ISSN 0022-0000.

Carmel Domshlak and Yefim Dinitz. Multi-agent off-line coordination: Structure and complexity. In *Proceedings of the 6th European Conference on Planning*, pages 277–288, 2001.

Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, Decidability and Undecidability Results for Domain-Independent Planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, Amsterdam, 2004. ISBN 978-1-55860-856-6.

Omer Giménez and Anders Jonsson. The complexity of planning problems with simple causal graphs. *J. Artif. Intell. Res. (JAIR)*, 31(1):319–351, February 2008. ISSN 1076-9757.

Omer Giménez and Anders Jonsson. Planning over Chain Causal Graphs for Variables with Domains of Size 5 Is NP-Hard. *J. Artif. Intell. Res. (JAIR)*, 34:675–706, 2009.

Malte Helmert. A Planning Heuristic Based on Causal Graph Analysis. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *ICAPS*, pages 161–170. AAAI, 2004. ISBN 1-57735-200-9.

Malte Helmert. The Fast Downward Planning System. *J. Artif. Intell. Res. (JAIR)*, 26:191–246, 2006.

Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J. Artif. Intell. Res. (JAIR)*, 14:253–302, 2001.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, 2nd international edition edition, 2003. ISBN 9788129700414.