Increasing the Efficiency and Feasibility of Automated Planning through the Specification of Domain-dependent Heuristic Information

Mikko Berggren Ettienne

Kongens Lyngby 2012
IMM-MSC-2012-137

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

A* with admissible heuristics is the leading approach to optimal planning. Pattern database (PDB) heuristics are admissible heuristics based on abstractions of the search space and have recently had a breakthrough as general heuristics for automated planning. The selection of appropriate abstractions is of paramount importance to the informedness of a PDB heuristic. Based on a combination of novel and well-known techniques, we show how to efficiently constrain abstractions which leads to increased informedness of PDB heuristics. State-of-the-art in PDB heuristics iteratively selects promising abstractions from the search space of all possible abstractions using modified local search techniques. We introduce an approach called *variable pruned mutex constrained extended pattern database generation* that has several theoretical advantages over the state-of-the-art approach. Some of which lead to more informed PDB heuristics, while others lead to reduced computation time without affecting informedness. Experimental evaluations show that our approach also improves state-of-the-art for PDB heuristics in practice.

Contents

Abstract i

1 Introduction 1
- 1.1 Structure of the thesis . . . 2

2 Planning 5
- 2.1 Planning Formalism and Notations . . . 6
- 2.2 Computational Problems . . . 11
- 2.3 Coping with Complexity . . . 15

3 Heuristics 17
- 3.1 Heuristics . . . 17
- 3.2 Abstraction Heuristics . . . 20
- 3.3 A General Abstraction Scheme . . . 20

4 Pattern Databases 23
- 4.1 Pattern Database Collections . . . 24
- 4.2 Mutex Constrained Pattern Databases . . . 31

5 Experiments 43
- 5.1 Implementation . . . 43
- 5.2 Configurations . . . 44
- 5.3 Results . . . 45

6 Future work 51

7 Conclusion 53

A Source 55

Chapter 1

Introduction

Automated planning (planning) is a major classical branch of *Artificial Intelligence* (AI), ultimately aiming at approaching human problem-solving ability and flexibility. Given a high-level description of the world, the planning problem is to determine how to achieve some desired goal.

Planning is a natural component in most systems comprised by intelligent and autonomous agents and has important applications in areas such as logistics, scheduling and space technology. The inherent level of abstraction also allows planning problems to easily model all kinds of abstract reasoning problems.

The expressivity and generality comes at the cost of high complexity and planning is intractable in the general case. Thus, making *one* solver (planner) that will perform sufficiently well on all planning problems is deemed to fail. Nevertheless, the research field of planning has experienced a major scalability breakthrough in the efficiency of general planners in the last decade, largely thanks to heuristic search.

General heuristics are not easy to design. The most successful approaches all build on the idea of solving an abstracted or relaxed version of the problem and using the solution length as the heuristic estimate.

We consider one such general heuristic called the pattern database (PDB) heuris-

tic. The PDB heuristic is very flexible but relies on a structured way to determine how to abstract planning problems in a general setting. So far, the most successful approach to this is called the *iPDB-method* which puts the PDB heuristic on par with other state-of-the-art heuristics for optimal planning. We present, analyse and experimentally evaluate several ideas for improving both the iPDB-method and the pattern database heuristic in general. Our main contribution is *Variable Pruned Mutex Constrained Extended Pattern Database Generation* which improves the iPDB-method both in theory and practice. The practical results are obtained by experiments on the benchmark set of planning problems provided by the latest International Planning Competition (2011 edition).

The International Planning Competition (IPC) is a biennial event where theoretical results and ideas for planning are put to the test and compared in practice on an ever growing benchmark set of well over 1000 diverse planning problems.

As it is customary in the planning community we will relate, compare and evaluate the practical efficiency of different approaches to solving planning problems in the frame of the benchmark set from the latest IPC. See Edelkamp *et al.* (2011) for an in-depth description and analysis of the IPC benchmark sets and how they are constructed.

We consider only fully ground planning in the sequential (single-agent) setting with full observability, determinism and static and discrete environments. The reader is expected to be familiar with these concepts as well as the remaining basic terminology of automated planning as found in Russell & Norvig (2010), Ghallab *et al.* (2004) or similar.

In this thesis definitions, theorems and lemmata that stem from elsewhere in the literature are marked with a reference. Results marked "original" are not found anywhere in the literature known by the author whereas unmarked definitions, theorems and lemmata either are variants or generalizations of the work of others or are expected to be so. All proofs are original by the author unless otherwise noted.

1.1 Structure of the thesis

Chapter 2 first introduces the relevant planning formalism and the associated notation. This is followed by a set of known results regarding the computational complexity of planning and a discussion on how they influence the practical applications. Chapter 3 examines the role of heuristics in planning and introduces

abstraction heuristics. In Chapter 4 we introduce the pattern database heuristic, the iPDB-method, present our contributions and consider the theoretical properties hereof. Chapter 5 moves on to evaluate these effects in practice and includes a brief discussion of the positive and negative results of the conducted experiments. Chapter 6 presents ideas for future work on PDB heuristics. Finally, Chapter 7 examines the broader consequences of the results obtained in the previous chapters.

Remark *The busy reader who is familiar with the complexity of automated planning may skip Section 2.2, as only already known complexity results are presented. However, it may help the reader reach a deeper understanding of the expressive power of automated planning and why it is very hard from a computational point of view. This is essential to understand both the power and limitations of automated planning. Furthermore, it serves in combination with Section 2.3 as a short primer for the following chapters.*

Chapter 2

Planning

This chapter defines the formalism in which we consider automated planning and presents some of the well known related complexity results. This is followed by a brief discussion on how these results affect our ability to solve planning problems in practice.

As a simple example of a planning problem consider the classical Gripper problem (McDermott, 2000) depicted in Figure 2.1.

Example 2.1 (gripper) *The problem consists of a number of balls, a robot with two grippers and two rooms. The robot can move between the rooms and pick up and drop balls with each of its grippers. Initially, the robot's grippers are empty and it is situated in one of the rooms along with all of the balls. The objective is to move all the balls to the other room.*

To automatically solve and reason about the complexity of solving such problems, we need a formal way of expressing them.

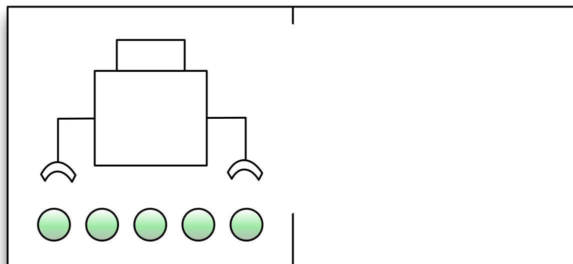

Figure 2.1: The initial state of a classical GRIPPER problem.

2.1 Planning Formalism and Notations

The Stanford Research Institute Problem Solver (STRIPS) is an automated planner developed at Stanford Research Institute in the early 1970s. STRIPS was also used to name the formal language accepted as input to the planner which has come to be defining for what we today know as *classical planning* or propositional STRIPS. *The Extended Simplified Action Structure (SAS$^+$)* formalism is an alternative to propositional STRIPS. Albeit sharing the representational power and computational complexity of propositional planning, the SAS$^+$ formalism allows a more compact representation and closer resembles how humans intuitively may think of a planning task. This resemblance is important when designing heuristics as it makes it easier to discover and exploit structural patterns in planning problems.

2.1.1 SAS$^+$ planning

The SAS$^+$-formalism is in many respects similar to the STRIPS formalism with the important difference that state variables may have non-binary (finite) domains. The following definitions differ slightly from those of Bäckström & Nebel (1995), Ghallab *et al.* (2004) and Helmert (2006) which we refer to for a more thorough definition of the formalism of multi-valued planning including extensions such as axioms, derived variables and non-unit-cost operators.

Definition 2.1 (SAS$^+$ planning tasks) A SAS$^+$ planning task Π is a 4-tuple $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ where:

- $\mathcal{V}$ is a finite set of the *state variables* v, each with an associated finite

domain D_v. If $d \in D_v$ then $v \doteq d$ denotes an *atom* of Π that is true iff v has the value d.

- s_0 is a *state* over $\mathcal{V}$ called the *initial state*. A state is a function s that maps each state variable $v \in \mathcal{V}$ into a value $s(v) \in D_v$. As is common in STRIPS planning, we represent and treat such functions as the set of atoms that they make true. For an atom x we write $var(x)$ for the variable associated with x and $val(x)$ for the value, for example if x is the atom $v \doteq d$, then $var(x) = v$ and $val(x) = d$.

- s_* is a *partial state* over $\mathcal{V}$ called the *goal*. A partial state s' is a state restricted to a subset of the state variables. $dom(s')$ is the subset of variables on which s' is defined and we say that v is a goal variable iff $v \in dom(s_*)$. We will often use the term state when it is clear from the context or does not matter whether the state is partial or complete. We say that a state s is a goal state iff $s \supseteq s_*$.

- $\mathcal{O}$ is a finite set of operators over $\mathcal{V}$. An *operator* o is a triple *(name, pre, eff)* where name is the unique name of the operator o separating it from other operators, *pre* and *eff* are partial states called preconditions and effects respectively.

We may refer to a SAS^+ planning task simply as a planning task and to state variables just as variables.

2.1.2 Semantics

The semantics of planning tasks are defined via labeled transition systems.

Definition 2.2 (labeled transition systems) A *labeled transition system* is a 5-tuple $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ where:

- S is a finite set of states.

- L is a finite set of *transition labels*.

- $A \subseteq S \times L \times S$ is a set of (labeled) *transitions*. A transition (s, l, s') represents the fact that there is a transition from state s to state s' with label l.

- $s_0 \in S$ is the *initial state*.

- $S_* \subseteq S$ is the set of *goal states*.

We may refer to a labeled transition system as a *transition system* as we only consider labeled transition systems throughout this thesis.

Definition 2.3 (reachability) Given a transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$, a state $s \in S$ is *reachable* iff there is a path from s_0 to s following the transitions of $\mathcal{T}$. If there is no such path, s is *unreachable*. The set of reachable states of $\mathcal{T}$ is denoted $r(\mathcal{T})$.

Definition 2.4 (transition systems for planning tasks) The transition system for a planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ is the labeled transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ denoted $\mathcal{T}(\Pi)$ such that:

- S is the set of states defined by all possible complete assignments of values to the variables in $\mathcal{V}$.
- $L = \{name \mid \textit{(name, pre, eff)} \in \mathcal{O}\}$
- $A = \{(s, n, s') \mid \; \textit{(n, p, e)} \in \mathcal{O} \wedge s \supseteq p \wedge s' \supseteq e\}$. We say that $o = (n, p, e)$ is applicable in state s and the result of applying it is s' iff there is a transition $(s, n, s') \in A$. We say that o achieves the goal from state s iff s' is a goal state and s is not and that o achieves an atom x iff $x \in e$.
- $S_* = \{s \mid s \in S \wedge s \supseteq s_*\}$.

A path from s_0 to S_* following the transitions of $\mathcal{T}(\Pi)$ is a *plan* for Π. A plan is optimal iff the length of the path is minimal. Π is *solvable* iff there is a plan for Π. Searching in the transition graph spanned by S and A is what we know as *state space search*.

An example of a Gripper planning task is given below, note that we have specified a somewhat unusual initial state for the sake of variation:

Example 2.2 (SAS$^+$ gripper) *Let* $\Pi_{gripper} = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ *such that:*

$$\begin{aligned}
\mathcal{V} &= \{ball_1, ball_2, ball_3, gripper_1, gripper_2, robot\} \\
&\quad where \\
D_{ball_1} &= D_{ball_2} = D_{ball_3} = \{\mathsf{room}_1, \mathsf{room}_2, \mathsf{gripper}_1, \mathsf{gripper}_2\} \\
D_{gripper_1} &= D_{gripper_2} = \{\mathsf{free}, \mathsf{occupied}\} \\
D_{robot} &= \{\mathsf{room}_1, \mathsf{room}_2\} \\
s_0 &= \{ball_1 \doteq \mathsf{room}_1, ball_2 \doteq \mathsf{room}_2, ball_3 \doteq \mathsf{gripper}_2 \\
&\quad robot \doteq \mathsf{room}_2, gripper_1 \doteq \mathsf{free}, gripper_2 \doteq \mathsf{occupied}\}
\end{aligned}$$

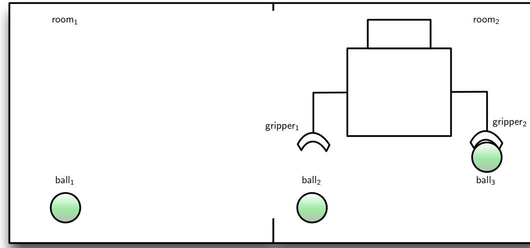

Figure 2.2: The initial state s_0 of the $\Pi_{gripper}$ task.

$$
\begin{aligned}
s_* = \{\ & ball_1 \doteq \mathsf{room_2}, ball_2 \doteq \mathsf{room_2}, ball_3 \doteq \mathsf{room_2}\} \\
\mathcal{O} = \{&(move_1, pre = \{robot \doteq \mathsf{room_1}\}, \mathit{eff} = \{robot \doteq \mathsf{room_2}\}), \\
&\quad (move_2, pre = \{robot \doteq \mathsf{room_2}\} \mathit{eff} = \{robot \doteq \mathsf{room_1}\})\} \cup \\
&\quad \cup_{i,j \in \{1,2\} \wedge k \in \{1,2,3\}} \\
&\{(pick_{i,j,k}, pre = \{ball_k \doteq \mathsf{room_i}, robot \doteq \mathsf{room_i}, gripper_j \doteq \mathsf{free}\} \\
&\quad \mathit{eff} = \{ball_k \doteq \mathsf{gripper_j}, gripper_j \doteq \mathsf{occupied}\}), \\
&\quad (drop_{i,j,k}, pre = \{ball_k \doteq \mathsf{gripper_j}, robot \doteq \mathsf{room_i}\} \\
&\quad \mathit{eff} = \{ball_k \doteq \mathsf{room_i}, gripper_j \doteq \mathsf{free}\})\}
\end{aligned}
$$

Operators, state variables and the values in their domains need not have meaningful names as in this example.

It may not be intuitively obvious why the gripper variables with the domain $\{\mathsf{free}, \mathsf{occupied}\}$ are needed, as their values are implied by the values of the ball variables. However, they are used to model the state of the grippers and the fact that a gripper can only carry one ball at a time. The definition above is not unique as there are other encodings that specify a planning task with the exact same properties. For example we could have defined the domain of the gripper state variables as $D_{gripper_x} = \cup_{i=1}^{3}\{\mathsf{ball_i}\} \cup \{\mathsf{free}\}$ and modified the effects of the $pick_{i,j,k}$ operators accordingly. This would increase the domain size of the *gripper* variables but would still encode the problem as intended.

In general, there is no single way of translating a problem, from natural language or even from the STRIPS formalism to the SAS$^+$ formalism. It is simply a matter of modeling. Modeling choices may increase the complexity of the problem,

sometimes dramatically and often unintended. We will return to this intricacy in Section 2.2. The STRIPS inspired *Planning Domain Definition Language* (PDDL; Edelkamp & Hoffmann, 2004) is currently the de facto language used in planning but polynomial-time algorithms for automatic translation between PDDL planning tasks and SAS^+ planning tasks exist (Helmert, 2008).

Having introduced the SAS^+ formalism we move on to consider the search and decision problems it gives rise to and their computational complexity; How hard is it to determine if a plan exists and how hard is it to generate (optimal) plans in the general case.

2.2 Computational Problems

When analyzing the computational complexity of planning we must distinguish the problem of generating a plan from the initial state to a goal state, from the problem of deciding whether a plan exists.

We consider the following problems:

Definition 2.5 [PLAN] is the following search problem: Given a planning task Π with initial state s_0 and goal state s_*, compute a path in $\mathcal{T}(\Pi)$ from s_0 to S_* or prove that none exists.

Definition 2.6 [PLANEX] is the related decision problem: Given a planning task Π with initial state s_0 and goal state s_*, does $\mathcal{T}(\Pi)$ contain a path from s_0 to S_*.

Definition 2.7 [PLANOPT] is the following optimization problem: Given a planning task Π with initial state s_0 and goal state s_* compute an optimal path in $\mathcal{T}(\Pi)$ from s_0 to S_* or prove that no path exists.

Definition 2.8 [PLANLEN,K] is the related decision problem: Given a planning task Π with initial state s_0 and goal state s_* and a constant k, does $\mathcal{T}(\Pi)$ contain a path of length at most k from s_0 to S_*.

It is well known that planning in general is decidable and intractable (Bylander, 1994) which holds for any of the problems above[1].

Decidability is trivial for all problems, as the number of states of a SAS$^+$ planning task is the set of all possible complete variable assignments. As each variable has a finite domain, the number of states must also be finite. In fact, for a planning task Π, the number of states is $\prod_{v \in \mathcal{V}} |D_v|$. In the following we will consider and prove some of the hardness results and discuss how the problems are related.

2.2.1 Complexity

The complexity of an algorithm is measured against the size of the input given to the algorithm. The size of a planning problem is traditionally given in terms

[1] We consider problems that are not polynomial-time solvable to be intractable and assume that P $\neq$ NP

of the variables and the operators in the problem, as an instance can trivially be encoded in space polynomial in these.

Trivially, search problems are always as hard as their corresponding decision problems, but the opposite is not always the case. In particular not for the problems in definitions 2.5-2.8, simply because writing down a plan may take more time than determining its existence.

Theorem 2.9 *For all $m > 0$ there is some solvable instance Π_m of the* PLAN *problem s.t. Π_m is of size $O(p(m))$ for some polynomial p and all solutions to Π are of length $O(2^m)$.*

(Bylander, 1994)

PROOF. Let $m > 0$ and $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ such that:

- $\mathcal{V} = \{v_1, \ldots, v_m\}$
- $D_v = \{0, 1\}$ for all $v \in \mathcal{V}$.
- $s_0 = \{v_1 \doteq 0, \ldots, v_m \doteq 0\}$.
- $s_* = \{v_1 \doteq 1, \ldots, v_m \doteq 1\}$
- $\mathcal{O} = \{(inc_1, pre = \emptyset, eff = \{v_1 \doteq 1\})\} \cup$ $\cup_{i=2}^{m} \{(inc_i, pre = \{v_1 \doteq 1, \ldots, v_{i-1} \doteq 1\},$ $eff = \{v_1 \doteq 0, \ldots, v_{i-1} \doteq 0, v_i \doteq 1\})\}$

Obviously Π_m is of size $O(m^2)$. We prove by induction that there is a shortest plan of length $2^m - 1$ for the above instance:
For $m = 1$ the single operator inc_1 achieves the goal in $2^1 - 1 = 1$ step.
The hypothesis is that the minimal plan for Π_i has length $2^i - 1$.
For $m = i + 1$. Only inc_{i+1} achieves $v_{i+1} \doteq 1$ thus it must be part of any plan. But inc_{i+1} is only applicable in states where $v_1 \doteq 1, \ldots, v_{m-1} \doteq 1$ hold. The sequence of operators inducing a path to such a state also induces a plan for Π_i which by the induction hypothesis has minimal length $2^i - 1$. Executing inc_{i+1} leaves us in the state $\{v_1 \doteq 0, \ldots, v_{m-1} \doteq 0, v_m \doteq 1\}$. But from that state, the minimal plan to reach the goal state is again a plan for Π_i with minimal length $2^i - 1$. Thus the total length is $2(2^i - 1) + 1 = 2^{i+1} - 1 = 2^m - 1$. □

(adapted from Bylander, 1994)

As there are planning tasks where all solutions are of exponential length, any algorithm for the PLAN and PLANOPT problems may require exponential time. The results for the decision problems are only slightly better.

Theorem 2.10 PLANEX *(*PLANLEN,K*) is in PSPACE.* *(Bylander, 1994)*

PROOF. We can guess a (length k) plan one operator at a time and verify it step by step using polynomial space, thus the problem is in NPSPACE. As NPSPACE = PSPACE by Savitch's theorem (Savitch, 1970), the problem is also in PSPACE. □

Theorem 2.11 PLANEX *is PSPACE-hard.* *(Bylander, 1994)*

PROOF. We refer the reader to Bylander (1994) and Bäckström (1992) for the straight forward but rather long proof. The proof shows how Turing machines whose space is polynomially bounded can be reduced to PLANEX in polynomial time.

Trivially PLANEX can be polynomially time reduced to PLANLEN,K by setting $k = \prod_{v \in \mathcal{V}} |D_v|$ for some problem instance with planning task Π. Thus PLANEX $\leq_p$ PLANLEN,K which in conjunction with Theorem 2.10 and 2.11 shows that both PLANEX and PLANLEN,K are PSPACE-complete.

We realize that the decision problems are only intractable if P $\neq$ PSPACE whereas the search problems are provably intractable. Furthermore we have planning problems where plan existence is provably easy while generating plans remains intractable. Consider for instance the Towers of Hanoi problem for which the length of the shortest plan can be found in low-order polynomial time whereas the plan itself is of exponential length and thus requires exponential time to produce.

The situation is somewhat different if we limit the problem space to the planning tasks that have plans of polynomial length.

Definition 2.12 (admitting short plans) A set of planning tasks S, *admits short plans* if there exists a polynomial p such that for all solvable planning tasks $\Pi \in S$, the optimal plan(s) for Π has length at most $p(|\Pi|)$. The polynomial p is called the *length bounding polynomial* for S.

We denote the related computational problems by subscripting them with the length bounding polynomial p.

Theorem 2.13 PLANLEN,K$_p$ *is in NP.* *(Kautz & Selman, 1996)*

PROOF. We can guess a length k plan one operator at a time and verify it step by step using polynomial time, thus the problem is in NP. □

Theorem 2.14 PLANLEN,K$_p$ *is NP-hard.* *(Kautz & Selman, 1996)*

PROOF. The proof is by polynomial reduction from 3-SAT. Let $\mathcal{F} = C_1 \wedge \ldots \wedge C_n$ be a propositional formula belonging to 3-SAT and let $X_1, \ldots, X_m$ be the variables used in $\mathcal{F}$. An equivalent PLANLEN,K$_p$ problem can be constructed as follows: Let $k = m + n + 1$ and $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ such that:

- $\mathcal{V} = \{A, X_1, \ldots, X_m, C_1, \ldots, C_n\}$
- $D_v = \{0, 1\}$ for all $v \in \mathcal{V}$
- $s_0 = \cup_{v \in \mathcal{V}} \{v \doteq 0\}$
- $s_* = \{C_1 \doteq 1, \ldots, C_n \doteq 0\}$
- $\mathcal{O} = \{(inc_A, pre = \emptyset, eff = \{A \doteq 1\})\} \cup$
  $\cup_{i=1}^{m} \{(inc_{X_i}, pre = \{A \doteq 0\}, eff = \{X_i \doteq 1\})\} \cup$
  $\cup_{i=1}^{n} \{(inc_{C_1^i}, pre = \alpha_1^i, eff = \{C_i \doteq 1\}), \ldots,$
  $(inc_{C_j^i}, pre = \{A \doteq 1\} \cup \alpha_{j_i}^i, eff = \{C_i \doteq 1\})\}$

where $\alpha_1^i, \ldots, \alpha_{j_i}^i$ are all possible (truth) assignments to the variables in clause C_i, such that clause C_i is satisfied (each $\alpha_{l_i}^i$ is a set of three atoms). There is a constant number of such possible variable assignments for each clause C_i and thus the number of operators is linear in $n+m$ and the reduction is polynomial.

Now as long as $A \doteq 0$ each X_i can change its value from 0 to 1 and no C_i can change its value. After A is changed to 1 no X_i can change its value (the assignment on the formula's variables is locked) and each C_i can change its value only if the current assignment to $X_1, \ldots, X_m$ satisfies clause C_i. But then there is a plan for $\mathcal{T}(\Pi)$ iff there is a satisfying assignment for $\mathcal{F}$. If there is a plan, then clearly there is a plan of length no more than $k = m+n+1$ which is linear in $|\Pi|$. □

(adapted from Brafman & Domshlak, 2003).

Similar results hold for PLANEX$_p$ and thus PLANEX$_p$ and PLANLEN,K$_p$ are NP-complete. More interesting, we have the following property of PLANOPT$_p$

Theorem 2.15 $\textsc{PlanOpt}_p$ *is NP-equivalent*[2].

PROOF. Let $O_{planLen}$ be an oracle that solves $\textsc{PlanLen},\textsc{k}_p$ in polynomial time, let p be the polynomial bounding plan length and let Π be task of the problem instance. $\textsc{PlanOpt}_p$ trivially solves the $\textsc{PlanEx}_p$ problem and is therefore NP-hard. We now show that it is also NP-easy. First, determine the length of the optimal path by calling $\mathrm{O}_{PlanLen}(\Pi, l)$ for $l \in \{0, \ldots p(|\Pi|)\}$ until a positive answer is received. If no such answer is received, we have proved that no optimal plan of length at most $p(|\Pi|)$ exists. Otherwise, let m denote the length of the shortest path. Now, use a greedy search in $\mathcal{T}(\Pi)$ with the heuristic function h defined as follows: $h(s) = 1$ if $\mathrm{O}_{PlanLen}(\Pi(s), m - g(s))$=YES and 0 otherwise, where $\Pi(s)$ is like Π except that s_0 is replaced by s and $g(s)$ is the distance from s_0 to s. The search will produce a path of length m. The search ends after k node expansions and each node has a polynomial amount of neighbors. Thus the search will consider only a polynomial amount of nodes. Also, the oracle is only invoked a polynomial amount of times. $\square$

Similar can be shown for $\textsc{Plan}_p$ and we see that the search and decision problems are more tightly related for problems with polynomially bounded plan lengths. We may conclude that some of the hardness of especially plan generation stems from the fact that shortest plans may be exponentially long.

However, if P$\neq$NP, as is widely believed, both generating plans and deciding their existence remains intractable even for problems admitting short plans.

2.3 Coping with Complexity

The complexity results presented above are somewhat discouraging for practical applications of planning. We cannot hope to solve arbitrary instances of intractable problems within reasonable time. However, the results give an upper bound on the worst case complexity of the formalism, namely the hardest problems encodable in SAS$^+$. Trivially, all problems of lower complexity are also encodable in this formalism and many of the real world problems that occur in practical applications are in fact tractable (Bäckström, 1995). Still, we cannot guarantee that an automated planner for an intractable formalism, can solve

[2] The NP-complete term only applies to decision problems. A search problem is NP-easy if it is polynomial time solvable with an oracle for some decision problem in NP, NP-hard if some NP-complete problem can be reduced to it and NP-equivalent if it is both NP-easy and NP-hard.

a tractable problem in polynomial time. To this end, many researchers (Bäckström, 1995; Bäckström & Nebel, 1995; Bylander, 1994; Brafman & Domshlak, 2003) have investigated how to restrict planning formalisms, both syntactically and structurally, to subclasses where planning becomes easier. Common to their results is, that the identified tractable subclasses all restrict the formalisms to such extent that even simple real world problems, that are provably tractable, are not easily expressible.

Example 2.3 (modeling "easy" tasks) *Consider the class of problems given by a generalization of the* GRIPPER *task from Example 2.2 with any even number of balls. These problems are completely characterized by the number of balls so there is no difference between worst-case and average case problems. The problem is in P (Helmert, 2003) and we immediately identify the problem of generating an optimal plan as intuitively "easy": Move to* room_1*, pick up any two balls, move to* room_2*, drop both balls, repeat until all balls are in* room_2*. However, this class of problems is far from trivial, if even possible, to model within any of the restricted subclasses of* SAS^+ *guaranteeing polynomial time planning (Bäckström, 1995; Bäckström & Nebel, 1995).*

As long as no P-complete subclass of a planning formalism has been identified, the identification of tractable subclasses is really only of theoretical use, as we have no reason to believe that any non-trivial polynomial-time solvable planning problems can be expressed within the required restrictions. Thus, it will neither help us identify nor solve such problems in practice.

So far, the most successful approach to solving planning problems in practice is heuristic forward state-space search which we will consider in the following chapters.

Chapter 3

Heuristics

This chapter gives a short introduction to heuristic functions and a discussion on what we can expect from this approach in terms of complexity. Hereafter we introduce abstraction heuristics which lie the ground for what we will consider in the remaining chapters.

Heuristic forward state-space search is so far the most successful approach to automated planning judging from the results of the latest IPC where all participating planners relied on heuristic search techniques.

3.1 Heuristics

Heuristic functions, sometimes called heuristic estimators or just *heuristics* are the most important part of heuristic search planners and the only thing separating them from blind search. A *heuristic* for a transition system $\mathcal{T}$ is a function $h : r(\mathcal{T}) \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$ where $h(s)$ is an estimate of the length of a shortest path from $s \in r(\mathcal{T})$ to S_*. The perfect heuristic h^* maps every state $s \in r(\mathcal{T})$ to the length of the shortest path from s to S_* or infinite if no such path exists. A heuristic h is *admissible* when $h(s) \leq h^*(s)$ for all states $s \in r(\mathcal{T})$. It is *consistent* when, for every two states $s, s' \in r(\mathcal{T})$, $h(s) \leq h(s') + c(s, s')$ where $c(s, s')$ is the length of the shortest path from s to s'. Consistency implies admissibility

but the converse is not true. All heuristic search algorithms for optimal planning require admissible heuristics and can avoid re-expanding nodes when using a consistent heuristic.

If h and h' are admissible heuristics and $h(s) \geq h'(s)$ for all $s \in r(\mathcal{T})$, we say that h *dominates* h'. Clearly, h is superior or equal to h' in terms of heuristic quality which has provable consequences for the number of nodes expanded by an optimal heuristic search algorithm. However, using h instead of h' will not necessarily provide better performance when measuring the performance of a planning system. h might be more complex to compute so that the gain in information is outweighed by the computational overhead resulting in worse overall performance of h than of h'. It is not uncommon for blind search to outperform many heuristics on complex planning tasks. Therefore, the general approach is to use a combination of two measures when comparing different heuristics.

Comparing the number of node expansions done by the search algorithm gives a correct measure of the informedness of the heuristics. Comparing computation time for the heuristic provides a measure of the computational complexity of the heuristic. Comparing the total *CPU time* used to solve a planning task provides a measure of the practically important relationship between informedness and computation time of the heuristic.

The rest of this thesis will consider only admissible heuristics and thus we will limit our focus to optimal planning.

3.1.1 Heuristic Search for Optimal Planning

For optimal planning, A* and variants hereof are the de facto search algorithms. With a consistent heuristic, A* has been proven to be admissible, complete, and optimally effective (Dechter & Pearl, 1985) in the sense that it never expands a node that can be skipped by an algorithm with access to the same heuristic information and that it always finds the optimal solution (if a solution exists).

Korf *et al.* (2001) has shown that A* will do a linear number of node expansions given a heuristic h with constant absolute error, i.e. $h(s) = \max(h^*(s) - c, 0)$ for some $c \in \mathbb{N}_1$ only under the following conditions:

- The branching factor of the search space is constant across inputs.
- There is only a single goal state

- The search space contains no transpositions.

All of these conditions are violated in all common benchmark tasks in planning (Helmert, 2003). Furthermore, we are not very likely to obtain polynomially-time computable heuristics with constant absolute error for any NP-equivalent class of planning problems, as this would imply P=NP (Helmert *et al.* , 2006).

Even if we accomplished to do so, Helmert & Röger (2008) show that for an APX-equivalent subset of the common benchmark tasks and a heuristic with a constant absolute error of 1, A* expands an exponential number of nodes, both in theory and practice.

So why do we study heuristic functions, when they seem to be deemed unsuccessful? The answer to this question is dependant on what perspective we choose to answer it from. If we are concerned with general problem solving in its most genuine form, then blind search will do as good as any other approach. To this end, planning has the advantage of the flexible formalisms making it easy to quickly model any problem. Heuristic search will not do us any good.

If we are concerned with problem solving for some predefined set of problems that are tractable or just tractable in the average case, it is likely that well designed heuristics will outperform blind search. According to Bäckström (1995), many application problems in structured environments like the industry are inherently tractable. But finding and exploiting the underlying structure causing tractability may be non-trivial. In some cases, the tractability may only hold for certain values of parameters which are not easily bounded, but can be assumed to have reasonable values.

In these cases, heuristic search has the potential to perform better than blind search by an arbitrary amount when measuring the time taken to solve a problem instance. Thus, the intention of studying and designing heuristic functions is to provide a general solver that is practically effective for a set of problems of interest.

One such is the ever growing set of benchmark problems from IPC ranging more than 1000 problem instances developed by the automated planning research community. For this set of problems, the research in heuristic search methods has succeeded in increasing the effectiveness of planning systems year after year.

3.2 Abstraction Heuristics

Abstraction heuristics map the transition system of a planning task into an abstract transition system where some information is ignored so that the task becomes less complex and typically smaller. The distance from the current state to a goal state in the abstract task is then used as an estimate for the cost of reaching a goal state from the current state in the original task. The abstraction preserves paths in the graph spanned by the transition system, intuitively by merging states, which makes it suitable for defining admissible heuristics.

3.3 A General Abstraction Scheme

Definition 3.1 (abstraction mapping) Given a transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$, an *abstraction mapping* for $\mathcal{T}$ is a function $\alpha : S \rightarrow 2^S$ such that $s \in \alpha(s)$ for all $s \in S$.

Definition 3.2 (α-abstraction) Let α be an abstraction mapping for $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$. Then the *$\alpha$-abstraction* of $\mathcal{T}$ is the transition system $\mathcal{T}^\alpha = \langle S^\alpha, L, A^\alpha, \alpha(s_0), S_*^\alpha \rangle$ such that:

- $S^\alpha = \{\alpha(s) \mid s \in S\}$
- $A^\alpha = \{\langle \alpha(s), l, \alpha(s') \rangle \mid \langle s, l, s' \rangle \in A\}$
- $S_*^\alpha = \{\alpha(s) \mid s \in S_*\}$

The size of an abstraction is the number of states in the transition system it induces and thus $|\mathcal{T}^\alpha| = |S^\alpha|$. The abstraction mapping α is a homomorphism from $\mathcal{T}$ to its α-abstraction $\mathcal{T}^\alpha$. Furthermore, if α is injective, then $\mathcal{T}$ and $\mathcal{T}^\alpha$ are isomorphic.

Lemma 3.3 *Let $\mathcal{T}$ be a transition system and let α be an abstraction mapping for $\mathcal{T}$. For every two states s, s' in the state space of $\mathcal{T}$, if there is a path from s to s' of length n following the transitions of $\mathcal{T}$, then there is a path of length n from $\alpha(s)$ to $\alpha(s')$ following the transitions of $\mathcal{T}^\alpha$.*

Proof. Follows directly from Definition 3.2. □

Definition 3.4 (abstraction heuristic) Let $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ be a transition system and let α be abstraction mapping for $\mathcal{T}$, the *abstraction heuristic* h^α for $\mathcal{T}$ is the function which assigns to each state $s \in r(\mathcal{T})$ the length of the shortest path in $\mathcal{T}^\alpha$ from $\alpha(s)$ to a goal state of $\mathcal{T}^\alpha$ or ∞ if no such path exists. (Helmert *et al.* , 2007)

Theorem 3.5 *h^α is admissible and consistent.*

PROOF. Admissibility follows directly from Lemma 3.3. Assume to reach a contradiction, that h^α is not consistent. Then there must exist a transition system $\mathcal{T}$ with abstraction mapping α and states $s, s' \in r(\mathcal{T})$ such that $h^\alpha(s) > h^\alpha(s') + c(s, s')$ where $c(s, s')$ is the length of the shortest path from s to s' in $\mathcal{T}$. By Lemma 3.3 there is a path of length $c(s, s')$ in $\mathcal{T}^\alpha$ from $\alpha(s)$ to $\alpha(s')$ and thus by Definition 3.4 it must be the case that $h^\alpha(s) \leq h^\alpha(s') + c(s, s')$ which is a contradiction. □

To illustrate how we can use abstractions to solve planning tasks, we consider a simplified version of the GRIPPER task from Example 2.2 with only one gripper and two balls below:

Example 3.1 (abstractions) *Consider a simplified* GRIPPER *planning task with the 4 variables $robot, ball_1, ball_2, gripper$ and the usual variable domains, where operators initial state and goal states are simplified accordingly.*

Figure 3.1 shows the transition system of this task. The abstraction aggregates states by assigning them to the same abstract state iff they agree on the status of the robot and on the number of balls in each room. Thus the abstraction does not distinguish the upper solution path (transporting $ball_1$ first) from the lower one (transporting $ball_2$ first). This does not affect solution length and therefore h^α is perfect. The same can be done for GRIPPER *tasks with two grippers and any number of balls yielding perfect heuristics and polynomial sized abstractions.* (*adapted from Nissim* et al. , *2011*)

The question is how to choose and represent an abstraction mapping that induces a good abstraction heuristic in general? The following chapters consider an approach for this based on pattern databases.

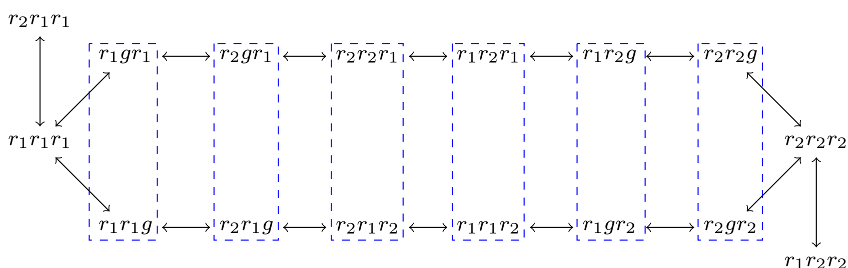

Figure 3.1: An abstraction of a simplified GRIPPER task. States are shown as a triple giving the abbreviated values of the $robot, ball_1, ball_2$ variables in this order (omitting transition labels, unreachable states and the value of *gripper* which is implied in all reachable states). For example, in the state marked r_2gr_1, the robot is in room_2, ball_1 is in the $\mathsf{gripper}$ and ball_2 is in room_1. The abstraction is indicated by the (blue) dashed boxes.

CHAPTER 4

Pattern Databases

This chapter presents pattern database (PDB) heuristics and the state-of-the-art approach for using PDBs as general heuristics for automated planning. We consider constrained abstractions and present and discuss the implications of a series of original results related to state-of-the-art of PDB heuristics.

Pattern databases (Culberson & Schaeffer, 1998) is the most effective heuristic currently known for solving big instances of combinatorial problems optimally, including Rubik's Cube, the Sliding Puzzle (Felner *et al.* , 2004) and Towers of Hanoi (Korf & Felner, 2007). It is also one of the most promising approaches to general admissible heuristics for planning.

Pattern databases are based on a particular type of abstractions called projections defined as follows differing slightly from the definition of Helmert *et al.* (2007):

Definition 4.1 (projections) The *projection* of an *equivalence relation* $\sim$ on a set X is the function $\pi : X \to X/\sim$ defined by $\pi(x) = [x]_\sim$ where the *equivalence class* $[x]_\sim = \{y \in X \mid x \sim y\}$.

For a transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ the projection of an equivalence relation on S is by Definition 3.1 an abstraction mapping for $\mathcal{T}$.

Definition 4.2 (pattern) Any subset of the variables of a planning task Π is called a *pattern* for Π.

Definition 4.3 (π-abstraction) Let Π be a planning task with transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ and let V be a pattern for Π. The abstraction mapping given by the projection of the equivalence relation $\sim$ on S defined such that $s \sim s'$ iff $s(v) = s'(v)$ for all $v \in V$ is denoted π_V. We say that the π_V-abstraction of $\mathcal{T}$ denoted $\mathcal{T}^V$ is a projection of $\mathcal{T}$ onto V.

Definition 4.4 (pattern database heuristic) Let Π be a planning task with transition system $\mathcal{T}$ and let V be a pattern for Π. V induces the abstraction heuristic h^{π_V} called a *pattern database heuristic* for $\mathcal{T}$. We use h^V as shorthand for h^{π_V}.

Definition 4.5 (pattern database) Let Π be a planning task with transition system $\mathcal{T}$ and let $\mathcal{T}^V$ with abstract state set S^V be the abstraction of $\mathcal{T}$ induced by the pattern V. The *database* for pattern V denoted $DB(V)$ is a precomputed table storing the length of the shortest path for every state $s \in S^V$ to a goal state of $\mathcal{T}^V$. The number of states in S^V is the number of different complete assignments to the variables in V thus $|DB(V)| = O(\prod_{v \in V} D_v) = |\mathcal{T}^V|$.

When discussing the size of a PDB we always refer to the size of the database.

A database for a pattern is build by exhaustive breadth-first regression search in a preprocessing of the task. Given a transition system $\mathcal{T}$ and a pattern V for a planning task Π and a state $s \in r(\mathcal{T})$, $h^V(s)$ is computed by determining the image of s under π_V followed by a constant-time lookup in the precomputed table $DB(V)$. The general approach is to store such tables in memory allowing very fast lookups.

Including all state variables of a planning task Π with transition system $\mathcal{T}$ in a pattern P, would make $\mathcal{T}^P$ isomorphic to $\mathcal{T}$. It would yield the perfect heuristic $h^P = h^*$, but computing it would be equivalent to solving the original planning task. Instead we may construct a collection of PDBs, each for a different small subset of the variables in Π, and then use the maximum or in some cases the sum of the heuristic estimates in the collection as a heuristic estimate.

4.1 Pattern Database Collections

This section presents and discusses the theoretical background of the state-of-the-art implementation of the general PDB heuristic in short. We will refer to

this as the *iPDB-method*. The implementation is a simplification of the approach described in detail in Haslum *et al.* (2007) which builds on the work and ideas of Edelkamp (2001), Haslum *et al.* (2005), and Edelkamp (2006) whereas details of the implementation and simplifications can be found in Sievers *et al.* (2012).

Given two PDB heuristics h^A and h^B for a planning task Π with transition system $\mathcal{T}$, such that A and B are subsets of its variable set, clearly h^A dominates h^B if $A \supseteq B$. As both h^A and h^B are admissible, the heuristic $h = \max(h^A, h^B)$ is also admissible and dominates both h^A and h^B. If the set of operators that affects some variable in A is disjoint from the set of operators that affect some variable in B, the heuristic $h = h^A + h^B$ is also admissible.

Proposition 4.6 *Given a planning task* $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$, $h = h^A + h^B$ *is an admissible heuristic for* $\mathcal{T}(\Pi)$ *if:*

$$\{o = (name,\ pre,\ eff) \in \mathcal{O} \mid dom(eff) \cap A \neq \emptyset\} \cap \\ \{o = (name,\ pre,\ eff) \in \mathcal{O} \mid dom(eff) \cap B \neq \emptyset\} = \emptyset$$

in which case we say that h^A *and* h^B *are* additive.

Proof. See Edelkamp (2001).

Note that the condition is sufficient and not necessary but has the advantage of being easy to check. A set of PDB heuristics is additive iff all PDB heuristics in the set are pairwise additive. Clearly, $h^A + h^B$ dominates $\max(h^A, h^B)$ and $h^{A \cup B}$ dominates both the sum and the maximum.

For some collection of pattern $C = \{P_1, \ldots, P_k\}$ where the additivity condition holds only between some of the induced PDB heuristics, there is a unique way of combining these into an admissible and consistent heuristic function that dominates all others. Haslum *et al.* (2007) calls this the canonical heuristic function for the pattern collection and denotes it h^C:

Theorem 4.7 *Let* $C = \{P_1, \ldots, P_k\}$ *be a collection of patterns for a planning task* Π *and let* A *be a collection of all maximal (w.r.t. set inclusion) additive subsets of* C*: The* canonical heuristic function *of* C *is:*

$$h^C(s) = \max_{S \in A} \sum_{P \in S} h^P(s)$$

where s *is a reachable state in* $\mathcal{T}(\Pi)$.

(Haslum et al. *, 2007)*

Proof. It follows directly from the definition that h^C dominates all other admissible combinations of the heuristics induced by the patterns in C. □

The sum over additive sets $S \in A$ can be removed from h^C if there is $S' \in A$ such that for every pattern $P_i \in S, P_i \subseteq P_j$ for some $P_j \in S'$ as the sum over S will always be dominated by the sum over S'. Yet, in the worst case, the number of non-dominated additive subsets may be exponential. The problem is similar to finding all maximal (w.r.t. set inclusion) cliques in a graph, for which there are algorithms that run in time polynomial in the number of maximal cliques (e.g. Tomita *et al.* , 2006). This has proven to be efficient enough for computing the canonical heuristic function in practice (Haslum *et al.* , 2007).

Even though h^A dominates h^B iff $A \supseteq B$, it might still be meaningful to include B in the collection for the canonical heuristic function since it may be additive with some patterns that A is not. In a collection C, a pattern P is redundant if $h^C = h^{C-\{P\}}$. This is the case whenever P does not appear in any of the non-dominated sums in the canonical heuristic function. A pattern containing no variable mentioned in the goal state is always redundant.

Below we give an example of a PDBs and a PDB collections over the simplified version of the Gripper task from Example 3.1 with a single gripper and two balls.

Example 4.1 (pattern databases and collections) *Consider the simplified* Gripper *planning task* Π *with the 4 variables:* $robot, ball_1, ball_2, gripper$ *and the usual variable domains, operators, initial state and goal states*

Figure 4.1 shows the abstract transition system induced by the pattern $P = \{robot, ball_1\}$.

We represent states in $\mathcal{T}(\Pi)$ *using the same abbreviated notation as in Figure 3.1 where the order of the variables is* $robot, ball_1, ball_2, gripper$. *Projected states are represented similarly where "free" variables are represented by* $\star$, *e.g. the abstract state* $t = \{r_2r_1gf, r_2r_1go, r_2r_1r_2f, r_2r_1r_2o\}$ *is represented by the 4-tuple* $r_2r_1\star\star$. *Multiple transitions between the same projected states are merged and self loops and transition labels are omitted. Goal states are subscripted with * and the initial state with 0.*

As an example, for the state $s = r_2r_1r_2f$ *the projection is* $\pi_P(s) = r_2r_1\star\star$ *and* $h^P(s) = h^*(s) = 4$ *as the heuristic recognizes that the robot must move to* room_1, *pick up* $ball_1$, *move to* room_2 *and drop* $ball_1$.

To see how PDB collections can increase the estimates we realize that for the

$$r_2 r_1 \star\star \leftrightarrow r_1 r_1 \star\star_0 \longleftrightarrow r_1 g \star\star \longleftrightarrow r_2 g \star\star \longleftrightarrow r_2 r_2 \star\star_* \leftrightarrow r_1 r_2 \star\star_*$$

Figure 4.1: The abstract transition system of a simplified GRIPPER task induced by the pattern $\{robot, ball_1\}$.

state $s' = r_2 r_2 r_1 f$, $h^P(s') = 0$ *while* $h^*(s') = 4$ *(move to* room$_1$*, pick up ball*$_2$*, move to* room$_2$ *and drop ball*$_2$*). For the initial state* $s_0 = r_1 r_1 r_1 f$, $h^P(s_0) = 3$ *while* $h^*(s_0) = 7$ *(e.g. pick up ball*$_1$*, move to* room$_2$*, drop ball*$_1$*, move to* room$_1$*, pick up ball*$_2$*, move to* room$_2$ *and drop ball*$_2$*). These estimates are rather bad and occur because the heuristic does not distinguish states that differ on the value of ball*$_2$*.*

Figure 4.2 shows the abstract transition system induced by the pattern $P' = \{ball_2\}$*. If we consider the collection* $C = \{P, P'\}$ *which is additive, we obtain a better estimate of both of these states:* $h^C(s') = h^P(s') + h^{P'}(s') = 0 + 2$ *and* $h^C(s_0) = h^P(s_0) + h^{P'}(s_0) = 3 + 2$*. Thus* h^C *underestimates* s_0 *and* s'' *by 2 whereas* h^P *underestimates them by* 4*.*

Furthermore the time taken to precompute the databases for the patterns in C *is linear in the time taken to precompute* $DB(P)$*. Thus we obtain a noticeable increase of heuristic informedness with a linear increase in computation time.*

$$\star\star r_1 \star_0 \longleftrightarrow \star\star g \star \longleftrightarrow \star\star r_2 \star_*$$

Figure 4.2: The abstract transition system of simplified GRIPPER task induced by the pattern $\{ball_2\}$.

The main problem with using PDB heuristics as a general heuristic for planning, is how to select which variables to include in the patterns such that the informedness of the heuristic high, while the resources needed to precompute and store the databases remain effectively tractable.

4.1.1 Automated Variable Selection

The iPDB-method is concerned with pattern selection and considers the optimization problem of finding the best collection of patterns within some given memory limit in an iterative manner. Finding the true optimum within the memory limit would require an exhaustive search through the search space of

pattern collections. The method instead finds a local optimum using simple hill climbing. The starting point is a collection consisting of one pattern for each goal variable for a planning task Π. From a given collection of patterns $C = \{P_1, \dots, P_k\}$ an expanded collection can be constructed by selecting a pattern $P_i \in C$, a variable $v \notin P_i$ and adding the new pattern $P_{k+1} = P \cup \{v\}$ to the collection, which defines the search neighborhood of C. Note that the heuristic quality is non-decreasing as every neighbor of C contains C. Starting from the initial collection of patterns with a single goal variable in each, the search repeatedly evaluates the neighbourhood of expanded pattern collections and selects the best neighbor to be the current collection in the next iteration ending either when no significant improvement can be obtained or when the memory limit has been reached.

4.1.1.1 Evaluating the Neighborhood

Haslum *et al.* (2007) consider different approaches to evaluating the neighborhood of a collection. They all build on a simplification of the following formula developed by Korf *et al.* (2001) which predicts the number of nodes expanded by a tree search (IDA*) with a given cost bound c, using an admissible and consistent heuristic h for pruning:

$$\sum_{k=0,\dots,c} N_{c-k} P(k) \tag{4.1}$$

where N_i is the number of nodes whose accumulated cost (distance from the starting node) equals i and P is the equilibrium distribution of h: $P(k)$ is the probability that $h(n) \leq k$, where n is a node drawn uniformly at random from the search tree up to cost bound c. N_i, P and c can be estimated where P is influenced by the pattern collection under consideration. In case of graph search the node n is drawn uniformly at random from the search space, i.e. all states reachable from the initial state. Their experimental results show the most promising results for the coarsest simplification of Equation 4.1 which they call the *counting approximation*.

Definition 4.8 (counting approximation) Given a sample $\bar{n}$ containing m nodes uniformly drawn at random from the search space within cost bound of the solution length, a pattern collection C and a pattern collection C' in the neighborhood of C, the *counting approximation* is a relative estimate of how much $h^{C'}$ improves upon h^C obtained by:

$$\frac{1}{m} \left| \{ n_i \mid h^C(n_i) < h^{C'}(n_i) \} \right| \tag{4.2}$$

As the sample $\bar{n}$ is not easily obtainable, an efficiently obtainable approximation is used instead. The solution length is estimated by the current heuristic

multiplied by two because the heuristic is underestimating. The sample $\bar{n}$ is then constructed by m random walks of binomially distributed length with a mean value of the estimated solution length. Each walk starts from the initial state and restarts if a state identified as a deadlock by $h^{C'}$ is reached.
(Haslum *et al.* , 2007)

4.1.1.2 Limiting Collection Search Space

Recall that a neighbor of a pattern collection C is constructed by extending some pattern $P \in C$ with a new variable $v \notin P$.

Definition 4.9 (irrelevance) Given a planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$, a variable $v \in \mathcal{V}$ and a pattern P for Π, v is *irrelevant* to P if we can prove that it will not increase the informedness of the induced heuristic, i.e. $h^P = h^{P \cup \{v\}}$. If we have no such proof, then we consider v to be *possibly relevant* to P.

It follows from Definition 4.9 that $h^C = h^{C'}$ for collections C and C', whenever C' is a neighbor of C constructed by adding an irrelevant variable to a pattern in C and thus C' does not improve upon C according to Equation 4.2. The causal graph is a common tool for analyzing causality between variables in a planning task and can help identify irrelevant and possibly relevant variables for a pattern.

Definition 4.10 (causal graph) The causal graph for a planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ denoted $CG(\Pi)$ is a directed graph with vertex set $\mathcal{V}$ where (v', v) is an edge in $CG(\Pi)$ iff $v \neq v'$ and there is an operator $(name, pre, eff) \in \mathcal{O}$ such that the following condition is true:

$$v \in dom(eff) \wedge v' \in dom(pre)$$

If P is a pattern for Π then the set of direct ancestors for all variables in P is denoted $da(P) = \{v \in \mathcal{V} \mid (v, v') \in CG(\Pi) \ \wedge \ v' \in P\}$.
(Helmert, 2006)

The iPDB-method considers all variables $v' \notin P$ irrelevant to P iff $v' \notin da(P)$. Neighbors constructed by adding irrelevant variables to a pattern in a collection C are therefore not considered when evaluating the neighborhood.

There is an immediate weakness of the notion of irrelevance considered by the iPDB-method which we identify by the following proposition.

Proposition 4.11 *Let $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ be a planning task, let P be a pattern for Π and let $v \in \mathcal{V} \backslash P$ and $v \notin pd(P)$. If v is a goal variable, then v is possibly relevant for P.*

(original)

Proof. Consider the planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ with transition system $\mathcal{T}$ where $\mathcal{V} = \{a, b\}, D_a = D_b = \{0, 1\}, s_0 = \{a \doteq 0, b \doteq 0\}, s_* = \{a \doteq 1, b \doteq 1\}, \mathcal{O} = \{(inc_a, pre = \emptyset, \mathit{eff} = \{a \doteq 1\}), (inc_b, pre = \emptyset, \mathit{eff} = \{b \doteq 1\})\}$. Now let P be the pattern $\{a\}$. Clearly $CG(\Pi)$ contains no edges and thus $b \notin dp(a)$. Now $h(s_0)^P = 1$ as a goal state can be reached in one step in $\mathcal{T}^P$. Yet $h(s_0)^{P \cup \{b\}} = 2$ as every path to a goal state contains at least two transitions in $\mathcal{T}^{P \cup \{b\}}$. Therefore v is possibly relevant to P according to Definition 4.9.□

Proposition 4.11 shows that the iPDB-method ignores neighbor collections that are constructed by adding possibly relevant variables to the current collection.

Observation 4.12 *Adapting the iPDB-method to Proposition 4.11 is not guaranteed to effectively improve it. Recall that the iPDB-method starts from a collection with a pattern for each goal variable and iteratively extends this collection. If there is and remains additivity between all pairs of patterns in the collection, then extending a pattern with a goal variable will not improve the heuristic induced by the collection, as the variable is already part of another pattern and thus the possible positive effect is totally captured by additivity. However, such strong additivity is only guaranteed for planning tasks that are completely decomposable, which is seldom the case (Helmert, 2004). Therefore, under the assumption that iteratively choosing the neighborhood collection that offers the best improvement according to the counting approximation, we expect that adaption to Proposition 4.11 will improve upon the iPDB-method. As this assumption is the basis for the iPDB-method which is state-of-the-art for PDB heuristics, it seems unlikely that it should not be the case.*

We denote the adapted iPDB-method given by refining the notion of irrelevance so that goal variables considered possibly relevant by *extended pattern database generation* (EPDBG). We evaluate the performance experimentally in Chapter 5.

Below is an example of how the iPDB-method would proceed on the simplified version of the Gripper task from Example 3.1 with a single gripper and two balls.

Example 4.2 (iterative variable selection) *Consider again a simplified* Gripper *planning task with the 4 variables $robot, ball_1, ball_2, gripper$ and the usual*

variable domains, operators, initial state and goal states.

The iPDB-method constructs the initial collection $C = \{\{ball_1\}, \{ball_2\}\}$ with a pattern for each goal variable. The neighborhood collections are

- $C_1 = \{\{ball_1\}, \{ball_1, gripper\}, \{ball_2\}\}$
- $C_2 = \{\{ball_1\}, \{ball_1, robot\}, \{ball_2\}\}$
- $C_3 = \{\{ball_1\}, \{ball_2\}, \{ball_2, gripper\}\}$
- $C_4 = \{\{ball_1\}, \{ball_2\}, \{ball_2, robot\}\}$

The induced heuristics $h^{C_1}, \ldots, h^{C_4}$ are evaluated on the states resulting from a series of random walks from the initial state and the best one is determined according to Equation 4.2. If the size of the database for pattern in a collection exceeds some given limit, the collection is not considered. Say for instance C_2 is chosen as the best collection. The procedure then repeats by constructing and evaluating the neighborhood collections of C_2. The repetition stops when the improvement in the neighborhood collections is estimated to be below some given limit or when the total size of the databases in the current collection exceeds some given limit.
The above description also holds for the EPDBG-method with the only difference that the collection $\{\{ball_1\}, \{ball_2\}, \{ball_1, ball_2\}\}$ would also be in the neighborhood of C_1.

4.2 Mutex Constrained Pattern Databases

In many cases, the variables of a planning task cannot take any values within their respective domains independent of each other. Consider for example the variables of the balls in the Gripper task from Example 2.2. Clearly, no two ball variables can take the value $\mathsf{gripper}_1$ at the same time as a gripper can only hold a single ball at a time. We know that this is the intended semantics of the Gripper problem and we can verify it by observing that it holds in any reachable state of the transition system for the task. Properties that are true in all reachable states of a planning task are called *invariants* of the planning task.

Definition 4.13 (mutually exclusive atoms) Let Π be a planning task with transition system $\mathcal{T}$. Any two atoms $v = d, v' = d'$ of Π are *mutually exclusive* iff $\{v = d, v' = d'\} \not\subseteq s$ for all states $s \in r(\mathcal{T})$.

Definition 4.14 (mutex set) Let Π be a planning task with transition system $\mathcal{T}$. A set M of atoms of Π is a *mutex set* for Π iff it contains at least two atoms and any two atoms in the set are mutually exclusive, i.e. $|M| > 1$ and $|M \cap s| \leq 1$ for all states $s \in r(\mathcal{T})$. M is a *maximal mutex set* iff there is no other mutex set M' for Π, such that $M \subset M'$. We say that a mutex set is an "at-most-one-atom" invariant for Π.

Definition 4.15 (mutex collection) The *mutex collection* for a planning task Π denoted $M(\Pi) = \{M_1, \ldots M_n\}$ is the set of all maximal mutex sets for Π.

Abstraction heuristics may underestimate the perfect heuristic by an arbitrary amount as they aggregate states which may result in "shortcuts" in the abstract transition systems. First we note by the following observation that abstractions preserve reachability:

Observation 4.16 *Let $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ be a transition system and let α be an abstraction mapping for $\mathcal{T}$. If $s \in r(\mathcal{T})$ is reachable in $\mathcal{T}$ then $\alpha(s)$ is reachable in its α-abstraction $\mathcal{T}^\alpha$.*

PROOF. Follows directly from Lemma 3.3. □

However, the converse does not hold, as shown by the following observation:

Observation 4.17 *There exists a transition system $\mathcal{T} = \langle S, L, A, s_0, S_* \rangle$ with abstraction mapping α and a state $s \in S$ such that $\alpha(s)$ is reachable in $\mathcal{T}^\alpha$ but s is unreachable in $\mathcal{T}$.*

PROOF. Let s, s' and s'' be states in S such that $s \in r(\mathcal{T})$ and $s', s'' \notin r(\mathcal{T})$ and $\langle s', l, s'' \rangle \in A$ and let α be defined such that $\alpha(s) = \alpha(s')$ and $\alpha(s'') = \{s''\}$. But then $\langle \alpha(s), l, \alpha(s'') \rangle \in A'$ by definition. As $s \in r(\mathcal{T})$ it follows from Lemma 4.16 that $\alpha(s)$ is reachable in $\mathcal{T}^\alpha$ and therefore $\alpha(s'')$ must also be. □

As shown by Observation 4.17, abstractions may aggregate unreachable and reachable into a reachable abstract state. From such a state, we may be able to reach an abstract states that is an aggregation of only unreachable states through a transition between two unreachable states in the original task. Clearly, this can lead to shorter paths, and we intuitively say that the abstraction violates invariants of the original task.

Definition 4.18 (abstract mutex state) Let Π be a planning task with transition system $\mathcal{T}$ and let α be an abstraction mapping for $\mathcal{T}$. If t is an abstract state in the state set of $\mathcal{T}^\alpha$ and $|s \cap M| > 1$ for all $s \in t$ where $M \in M(\Pi)$ then t is an *abstract mutex state*.

Lemma 4.19 *Let $\mathcal{T}$ be a transition system and let α be an abstraction mapping for $\mathcal{T}$. If $s, \ldots, s^n$ are states on a path from s to s^n in $\mathcal{T}$ such that $s, \ldots, s^n \in r(\mathcal{T})$, then $\alpha(s), \ldots, \alpha(s^n)$ is a path in $\mathcal{T}^\alpha$ where none of $\alpha(s), \ldots, \alpha(s^n)$ are abstract mutex states.*

(original)

Proof. $s \in \alpha(s), \ldots, s^n \in \alpha(s^n)$ and as all of $s, \ldots, s_n$ are reachable, it follows by Definition 4.18 that none of $\alpha(s), \ldots, \alpha(s^n)$ are abstract mutex states. □

Definition 4.20 (constrained abstraction) Let Π be a planning task with transition system $\mathcal{T}$ and let α be an abstraction for $\mathcal{T}$. The *constrained α-abstraction* for Π is a restriction of $\mathcal{T}^\alpha$ where all transitions (if any) to and from abstract mutex states are removed from $\mathcal{T}^\alpha$. We denote the heuristic induced by the constrained α-abstraction by h^α_c.

Theorem 4.21 *Let Π be a planning task with transition system $\mathcal{T}$ and let α be an abstraction for $\mathcal{T}$. $h^\alpha(s) \leq h^\alpha_c(s) \leq h^*(s)$ for all $s \in r(\mathcal{T})$.*

(adapted from Haslum et al., 2005)

Proof. $h^\alpha(s) \leq h^\alpha_c(s)$ for all $s \in r(\mathcal{T})$ follows by definitions 3.4 and 4.20 as no shorter paths are introduced by constraining $\mathcal{T}^\alpha$ which only removes transitions. To see why $h^\alpha_c(s) \leq h^*(s)$, recall that $h^*(s)$ is the length of the shortest path from s to a goal state of $\mathcal{T}$. By Lemma 4.19 there is a path of the same length from $\alpha(s)$ to a goal state of $\mathcal{T}^\alpha$ consisting of states that are not abstract mutex states. Thus by Definition 4.20, this path is preserved in the constrained α-abstraction and therefore it must be the case that $h^\alpha_c(s) \leq h^*(s)$. □

Recall that, given a planning task Π with transition system $\mathcal{T}$ and a pattern V for Π, $DB(V)$ is constructed by exhaustive breadth-first regression search in the transition system $\mathcal{T}^V$. Constrained abstractions may therefore not only increase the accuracy of the heuristic estimates as of Theorem 4.21 but may also decrease the construction time as removing transitions from $\mathcal{T}^V$ may reduce the search space by an arbitrary amount. A PDB is *mutex constrained* when it is build from a constrained abstraction.

Below we give an example of a constrained abstraction on the well known simplified GRIPPER task from Example 3.1 with a single gripper and two balls.

Example 4.3 (constrained abstraction) *Consider again a simplified* GRIPPER *planning task* Π *with the 4 variables* $robot, ball_1, ball_2, gripper$ *and the usual variable domains, operators, initial state and goal states and transition system* $\mathcal{T}$. $M = \{ball_1 \doteq \mathsf{gripper}, ball_2 \doteq \mathsf{gripper}, gripper \doteq \mathsf{free}\}$ *is a mutex set for* Π.

Now consider the pattern $P = \{robot, ball_1, ball_2\}$ *that induces an abstract transition system* $\mathcal{T}^P$ *depicted in Figure 4.3 using the same notation as Example 4.1. Clearly, for the initial state of* Π, $s_0 = r_1r_1r_1f$, $h^P(s_0) = 5$ *as there is a path to a goal state of length* 5 *along the thick lines in Figure 4.3. However,* $h^P_c(s_0) = 7$ *as the red lines are removed in the constrained abstraction leaving us with two shortest path to a goal state of length* 7*. Thus, we see a significant increase of informedness as the result of mutex constraining.*

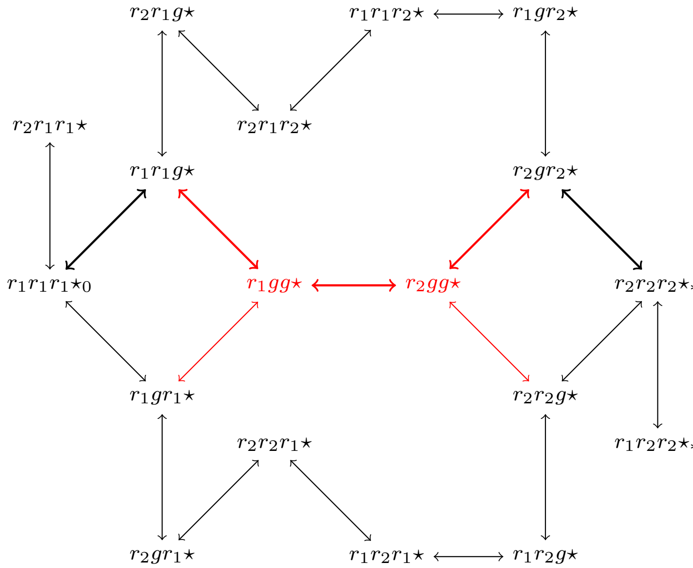

Figure 4.3: The abstract transition system of simplified GRIPPER induced by the pattern $\{ball_1, robot\}$.

The idea of mutex constraining PDBs is by no means new; Haslum *et al.* (2005) were the first to consider the idea. Their experiments are very sparsely analysed but are generally negative and their heuristic performs poorly compared to state-of-the-art heuristics at that time. Haslum *et al.* (2007) use mutex constrained

PDBs with an iterative variable selection procedure comparable to the iPDB-method but provide no evaluation of the improvement achieved with mutex constrained PDBs.

The above sources also make no mention of implementation details which can be of critical importance when measuring the practical effectiveness of a heuristic.

4.2.1 Efficient Mutex Constraining

Given a planning task Π, a sound approximation of $M(\Pi)$ can be computed in time polynomial in the size of Π (Bonet & Geffner, 1999). This approximation can be used to detect abstract mutex states "on-the-fly" when building the database for a pattern for Π. To make the approach feasible in practice, we use a novel idea based on a successor generator data structure very similar to decision trees.

Note that the transition graphs spanned by the transition systems defining the semantics of planning tasks are implicitly defined by the operators, variables and variable domains of a planning task. In practice, transitions and states are generated on-the-fly during search by considering which operators are applicable to the current state and what the resulting states are. For this reason, the successor generator is defined on the planning task and not on its transition system. To this end, a pattern for a planning task is in practice, a simplified version of the same planning task where only the variables in the pattern appear and the remaining variables are removed from the preconditions and effects of all operators. Abstract states are aggregations of states that agree on the value of the projected variables and are thus efficiently represented by a single atom for each of these variables.

Definition 4.22 (successor generators) A *successor generator* for a planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ is a rooted tree consisting of *selector nodes* and *generator nodes*. A selector node is an internal node of the tree and has an associated variable $v \in \mathcal{V}$ called the *selection variable*. Moreover, it has $|D_v| + 1$ children accessed via labeled edges, one edge labeled $v \doteq d$ for each value $d \in D_v$ and one edge labeled $\top$. The latter edge is called the *don't care edge* of the selector. A generator node is a leaf node of the tree and has an associated set of operators from $\mathcal{O}$ called the set of *generated operators*. Each operator $o \in \mathcal{O}$ occurs in exactly one generator node, and the set of edge labels leading from the root to this node (excluding don't care edges) equals the precondition of o. Given the successor generator for Π and a state s we can compute the set of applicable operators in s by traversing the successor generator as follows, starting from the root:

- At a selector node with selection variable v, follow the edge $v \doteq s(v)$ and the don't care edge.

- At a generator node, report the generated operators as applicable.

We refer to Helmert (2006) for further details along with the algorithm for initializing such data structures.

When building the database for a pattern, we obtain what corresponds to a constrained abstraction by slightly modifying the successor generator such that no operators are reported when evaluating an abstract mutex state. This is achieved by inserting one special operator in the successor generator for each pair of mutex atoms that are relevant to the pattern. The preconditions of the operator is the pair of mutex atoms it represents. Whenever such a special operator is encountered, the recursion is stopped and an empty set of operators is reported.

To efficiently represent successor generators and speed up the retrieval of applicable operators, only nodes and edges that lead to non-empty sets of operators are inserted upon construction. Thus we might have to insert new nodes and edges when inserting our special operators.

The space used by the successor generator for a PDB is in practice negligible compared to the space used by the PDB itself. The same holds for the time taken to modify the successor generator compared to the time taken to build the PDB. More interesting is the possible overhead incurred in traversing the new edges of the successor generator for each node during exhaustive search. Our hope is that this is accounted for by the possible decrease in the number of states to consider. Note that the newly inserted special operators has exactly two preconditions in all cases. Thus we will at most consider $O(1)$ new edges and nodes for each mutex pair. This follows as all selector nodes that does not lead to any generated operators can be removed from the tree (Helmert, 2006).

4.2.2 Mutex Constrained Pattern Database Generation

Mutex constraining the abstract transition system induced by some pattern affects what variables are relevant to the pattern. To this end we need to extend what we consider to be possibly relevant variables if we are to use the iPDB-method or EPDBG-method with mutex constrained PDBs

Observation 4.23 *There exists a planning task* $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ *and a pattern* P *for* Π*, a variable* $v \in \mathcal{V}$*,* $v \notin P \cup da(P)$ *and a mutex set* $M \in M(\Pi)$ *with atoms* $x, x' \in M$ *such that* $var(x) = v$*,* $var(x') \in da(P)$ *and* v *is possibly relevant for* P *if we consider only constrained abstractions.*

Proof. Consider the planning task $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ with transition system $\mathcal{T}$ where:

- $\mathcal{V} = \{a, b, c\}$ where $D_a = \{0, 1, 2\}, D_b = D_c = \{0, 1\}$
- $s_0 = \{a \doteq 0, b \doteq 1, c \doteq 1\}$
- $s_* = \{a \doteq 2\}$
- $\mathcal{O} = \{$

$$(inc0_a, pre = \{a \doteq 0, b \doteq 0\}, eff = \{a \doteq 1, b \doteq 1\}),$$
$$(inc1_a, pre = \{a \doteq 1\}, eff = \{a \doteq 2\}),$$
$$(inc_c, pre = \{b \doteq 0, c \doteq 0\}, eff = \{b \doteq 1, c \doteq 1\}),$$
$$(dec_c, pre = \{b \doteq 1, c \doteq 1\}, eff = \{b \doteq 0, c \doteq 0\})\}.$$

Now let P be the pattern $\{a\}$. Clearly $da(P) = \{b\}$ and $\{a \doteq 1, b \doteq 0, c \doteq 1\}$ is a mutex set for Π. Figures 4.4 and 4.5 show the abstract transition systems induced by the patterns P and $P \cup \{c\}$ respectively. The abstract states are represented by their equivalence classes where an element is a state in $\mathcal{T}$ represented by a triple giving the value for the variables a, b and c. (transitions labels are omitted). The initial state is subscripted with 0 and goal states with $*$. Clearly, $h^p(s_0) = 2$ and we see that there is a path along the thick red lines in Figure 4.5 of length two, thus $h^{P \cup \{c\}}(s_0) = 2$. But the state marked in red is an abstract mutex state by Definition 4.18 and thus all edges marked in red are removed in the constrained abstraction. But then, the shortest path is of length three and thus $h_c^{P \cup \{c\}}(s_0) > h^P(s_0)$ and v is possibly relevant to P. □

$$000, 010, 001, 011_0 \longrightarrow 100, 110, 101, 111 \longrightarrow 200, 210, 201, 211_*$$

Figure 4.4: The abstract transition system induced by the pattern P

However, mutex constraining the abstract transition system induced by some pattern actually allows us to limit the collection search space. Binary state

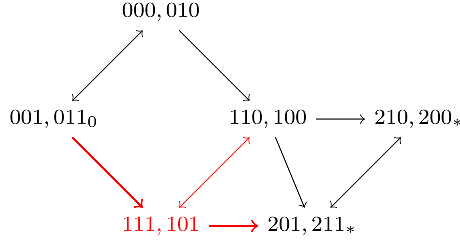

Figure 4.5: The abstract transition system induced by the pattern $P \cup \{c\}$

variables in a planning task may have the sole purpose of enforcing, that "at-most-one invariants" of a planning task are not violated. Such state variables become redundant when mutex constraining the abstraction ensures that the invariant is not violated.

Definition 4.24 (redundant variables) Let $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ be a planning task. A variable $v \in \mathcal{V}$ is *redundant* to Π if we can choose a variable $v' \in \mathcal{V}$ and $v' \neq v$ such that $h^{P\cup\{v\}}(s) \leq h^{P\cup\{v'\}}(s)$ for all reachable states s in $\mathcal{T}(\Pi)$ and all possible patterns P for Π.

Lemma 4.25 *Let $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ be a planning task with transition system $\mathcal{T}$ and let P be a pattern for Π. Every transition on every shortest path in $\mathcal{T}^P$ is induced by an operator $o = (n, p, e) \in \mathcal{O}$ such that dom$(e) \cap P \neq \emptyset$.*

Proof. See Helmert (2006)

Definition 4.26 (invariant securing variables) Let $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ be a planning task. A variable $w \in \mathcal{V}$ is *invariant securing* to Π if $|D_w| = 2$ and the following conditions hold *assume without loss of generality that* $D_w = \{0, 1\}$:

1 $w \doteq 1$ is an atom in some mutex set M for Π.

2 w is not a goal variable.

3 Exactly one atom in M is true in every reachable state of $\mathcal{T}(\Pi)$.

4 For every operator $(n, p, e) \in \mathcal{O}$, $w \doteq 0 \notin p$.

5 For $(n, p, e) \in \mathcal{O}$, if $v \doteq d \in e$ for some atom $v \doteq d \in M$ then $v' \doteq d' \in p$ for some atom $v' \doteq d' \in M$ where $v' \neq v$, $d \in D_v$, $d' \in D_v$ and if $v \neq w$ there is an operator $(n', p', e') \in \mathcal{O}$ such that $p = p'$ and $e' = e \backslash \{v \doteq d\} \cup \{w \doteq 1\}$

i.e. an operator that adds an atom x to M has a precondition on another atom x' in M (that is therefore removed from M) Also, if the atom added is not $w \doteq 1$ then there is an equivalent operator with the only difference that it adds $w \doteq 1$ to M instead of x

6 For every operator $o = (n, p, e) \in \mathcal{O}$, $w \doteq 1 \in p$ iff $e = \{w \doteq 0, x\}$ for some $x \in M$.

Theorem 4.27 *Let $\Pi = \langle \mathcal{V}, s_0, s_*, \mathcal{O} \rangle$ be a planning task and $w \in \mathcal{V}$ be an invariant securing variable for Π then w is redundant to Π if we consider only constrained abstractions.*

(original)

Proof. *(all references to conditions are to the conditions of Definition 4.26)* Trivially w is redundant to all patterns P for Π that satisfies $w \in P$. Let P be an arbitrary pattern for Π such that $w \notin P$, let $P' = P \cup \{w\}$. Recall that abstractions preserve transitions and that every transition is induced by an operator. Let s_1 be an arbitrary reachable state in $\mathcal{T}$ and let $t_1, \dots, t_n$ denote the sequence of states on a shortest path from $\pi_P(s_1) = t_1$ to a goal state of $\mathcal{T}^P$. Let $o_1, \dots, o_{n-1} \in \mathcal{O}$ denote a sequence of operators that induces the transitions between these states.

Claim 4.28 *By condition 5 and Lemma 4.25, we may assume for any $o_j = (p_j, n_j, e_j)$ in this sequence that if $x \in e_j$ for some atom $x \in M$ then $var(x) \in P$ and either $x' \in p$ for some $x' \in M$ and $var(x') \in P$ or $w \doteq 0 \in e_j$.*

Observation 4.29 *For every state t_j we have exactly two states in $\mathcal{T}^{P'}$ which we denote u_j^1 and u_j^0 such that $s \in u_j^1$ iff $s(w) = 1$ and $s \in u_j^0$ iff $s(w) = 0$ for every $s \in t_j$. Also, as w is not a goal variable by condition 2, u_n^1 and u_n^0 are goal states of $\mathcal{T}^{P'}$.*

The transition from t_j to t_{j+1} induced by the operator $o_j = (n_j, p_j, e_j)$ exists because there is some $s \in t_j$ such that $p_j \subseteq s$ and some $s' \in t_{j+1}$ such that $e_j \subseteq s'$. Also, for both of these states, there is another state that only differs on the value of w. Therefore one of the following three cases must hold for o_j:

1 $w \notin dom(e_j)$ and therefore o_j induces transition from u_j^1 to u_{j+1}^1 and from u_j^0 to u_{j+1}^0. In this case we say that o_j is a type 1 operator.

2 $w \doteq 0 \in e_j$ and thus $w \doteq 1 \in p_j$ by condition 6. Then o_j induces a transition from u_j^1 to u_j^0. In this case we say that o_j is a type 2 operator.

3 $w \doteq 1 \in e_j$ and by condition 4 o_j a transition from u_j^0 to u_{j+1}^1 and from u_j^1 to u_{j+1}^1. In this case we say that o_j is a type 3 operator.

By condition 3 one of the following must be the case:

- case 1 $w \doteq 1 \in s_1$
- case 2 $x \in s_1$ for some atom $x \in M$ and $var(x) \in P$
- case 3 $x \in s_1$ for some atom $x \in M$ where $var(x) \neq w$ and $var(x) \notin P$

If the path induced by the operators $o_1, \dots o_{n-1}$ in $\mathcal{T}^P$ is also a path from $\pi_{P'}(s_1)$ to a goal state of $\mathcal{T}^{P'}$ then clearly $h^P(s_1) = h^{P'}(s_1)$.

Consider case 1 and assume to reach a contradiction that the sequence of operators does not induce a path in $\mathcal{T}^{P'}$: The only way to achieve this is if the (sub)sequence of operators $o_k, \dots, o_m$ for $0 < k < m \leq n$ exists such that o_k is a type 2 operator and the first of such in the sequence $o_1, \dots, o_{n-1}$, o_i for $k < i < m$ are type 1 operators and o_m is a type 2 operator.

By condition 6 and because $o_k = (n_k, p_k, e_k)$ is of type 2 it must be the case that $x \in p_k$ and $x \in M$ and by Lemma 4.25, $var(x) \in P$. Thus $x \in s$ for all $s \in t_{k+1}$. Now, by Claim 4.28 and condition 3, there must be some atom $x \in M$ and $x \in P$ and $x \in s$ for all $s \in t_m$ and because o_i for $j < i < m$ are of type 1, $x \neq w \doteq 1$. By condition 6, we also have for $o_m = (n_m, p_m, e_m)$ that $x' \in e_m$ for some $x' \in M$ and by Lemma 4.25 $var(x) \in P$. If $x = x'$ then $t_{m-1} = t_m$ which contradicts that the sequence $t_1, \dots, t_n$ is a shortest path. If $x \neq x'$ then $\{x, x'\} \subseteq s$ for all $s \in t_m$ and thus t_m is an abstract mutex set which contradicts that it is part of a shortest path. In any case we reach a contradiction and thus $h^P(s_1) = h^{P'}(s_1)$ in case 1.

case 2 is completely analogous to case 1.

Consider case 3 where $x \in s_1$ for some atom $x \in M$ where $var(x) \neq v$ and $var(x) \notin P$: Now let $P'' = P \cup \{var(x)\}$. By case 2, we know that $h^Q(s) = h^{Q \cup \{v\}}(s)$ for a pattern Q and any reachable state s if $x \in s$, $x \in M$ and $var(x) \in Q$. But then we can take $Q = P''$ and $s = s_1$ and thus $h^{P''}(s_1) = h^{P'' \cup \{v\}}(s_1)$. But trivially $h^{P'' \cup \{v\}}(s_1) \geq h^{P'}(s_1)$ because $P' = P \cup \{v\} \subset P'' \cup \{v\}$ and thus it must also be the case that $h^{P''}(s_1) \geq h^{P \cup \{v\}}(s_1)$.

But as $P \subset P''$ we have $h^{P''}(s_1) \geq h^{P \cup \{v\}}(s_1)$ in all three cases and as P and s_1 were chosen arbitrarily, with the only assumption that s_1 is reachable, w is redundant to Π also if $w \notin P$ which concludes the proof. □

Observation 4.30 *Observe that Theorem 4.27 does not only apply to a projection induced by a pattern but readily generalizes to all kinds of mutex constrained abstractions that distinguishes states that differ on the value of redundant variables. The consequence is that we never have to consider a variable that is redundant by Theorem 4.27 when constructing an abstraction mapping because we are guaranteed an unchanged level of informedness.*

Redundant variables are clearly irrelevant to every pattern for a planning task if we consider constrained abstractions. Invariant securing redundant variables are easily identifiable by the conditions in Definition 4.26 and can be removed from the iterative variable selection procedures described in Section 4.1.1. This reduces the neighborhood collection search space and does not decrease the quality of the final heuristic by Theorem 4.27.

We again consider an example over the Gripper task.

Example 4.4 (redundant variables) *Consider a* Gripper *task with n balls and two grippers. The gripper variables are redundant by Theorem 4.27 when we consider mutex constrained abstractions. For any collection C considered by variable selection procedure of the iPDB-method we can avoid considering neighborhood collections that are constructed by adding a gripper variable to a pattern in C. If C is the initial collection, then C contains a pattern for each ball and thus the size of the neighborhood is reduced by $2n$ by ignoring the gripper variables. Assume that the neighbor C' constructed by adding the robot variable to some pattern in C gets chosen in the first iteration. Thus, C' has $n + 1$ patterns all without gripper variables, and the size of the neighborhood of C' is reduced by $2n + 2$ if we ignore the gripper variables, etc.*

In many planning tasks modeling transportation problems of real world environments, binary state variables are used to encode the state of locations that can be either free or occupied. Under the right conditions all such variables can be ignored which can result in a dramatic speedup of the variable selection procedure of the iPDB-method.

Using mutex constrained abstractions and adapting the notion of possibly relevant to Observation 4.23 we arrive at what we call *mutex constrained iPDB* and *mutex constrained EPDBG*. If we on top of that apply Theorem 4.27 we obtain

the approaches *variable pruned mutex constrained iPDB* and *variable pruned mutex constrained EPDBG*. We evaluate the performance of the different approaches in Chapter 5.

4.2.3 Limitations of Pattern Databases

As seen in Example 3.1 we can represent the perfect heuristic for the GRIPPER tasks using a sophisticated abstraction mapping.

However, projections are too coarse to mimic the required level of sophistication and thus we cannot compute and represent the perfect heuristic for the class of gripper tasks with a varying number of balls in polynomial time and space using PDB heuristics (Helmert *et al.* , 2007).

To this end, the Merge & Shrink heuristic (M&S; Nissim *et al.* , 2011) strictly generalizes the PDB heuristic by allowing more fine grained abstraction mappings than projections. This enables the computation of the perfect heuristic in polynomial time for some tractable benchmarks tasks where the PDB heuristic provably cannot (Helmert *et al.* , 2007). However, the theoretical power of M&S relies on the ability to make perfect decisions about how to design abstractions and in practice, the PDB heuristic is competitive with M&S. M&S is the currently leading heuristic for optimal planning and we will include it for comparison when evaluating the modified PDB heuristics in the following chapter.

We note that the huge number of permutations in the GRIPPER tasks is the main reason for the exponential sized search spaces. There is complete symmetry between the balls, i.e. the length of the optimal plan is uninfluenced by which order we choose to move the balls in (also it does not matter in which order we use the grippers). State-of-the-art methods for symmetry breaking in planning (Domshlak *et al.* , 2012) reduce the search space of such problem to polynomial size in polynomial time. Thus we need not be too concerned with the limitation of PDB heuristics when it comes to highly symmetrical planning tasks.

CHAPTER 5

Experiments

This chapter discusses the implementation details of our results and gives an experimental evaluation of their practical application.

Encouraging theoretical results do not necessarily imply similar results in practice, in particular not for heuristics as we have already indicated in earlier chapters. To evaluate the practical usefulness of our heuristics, we have conducted an experimental study on the benchmark set of the latest IPC consisting of 280 planning tasks which are known to be hard for optimal planners. The benchmark set is composed of problems from 14 different categories with 20 tasks in each category. Tasks in the same category are usually tightly related. For instance, GRIPPER tasks with a varying number of balls comprised a category in one of the earlier IPC editions.

5.1 Implementation

Our implementation is built on top of the FastDownward planning system (FD; Helmert, 2006). Many state-of-the-art heuristics are implemented within FD which took the first, second, fourth, and fifth place in different configurations at the latest edition (2011) of IPC. To this end, FD is a great framework

for testing new heuristics as it facilitates an unbiased comparison where both implementation details of search algorithms and translation between formalisms is abstracted away, leaving only the heuristic estimates and their computation time as the reason for varying performance.

FD is a complex planning system comprised by more than 500 source files with more than 100.000 lines of C++ and Python code. The documentation is sparse, but the implementation is very modular allowing new heuristic methods to be implemented and added to the system with ease. Thus, our implementation is mostly limited to modifications and extensions to the already implemented iPDB-method as well as the successor generator data structure. But also the causal graph implementation has been modified and redundant variable identification has been implemented.

All planning tasks in the IPC benchmark set are specified in (first-order) PDDL. Internally, FD works with fully instantiated SAS$^+$ tasks and thus, it has a translator part that translates planning tasks from PDDL to SAS$^+$ (Helmert, 2008). As part of the translation, a sound approximation of the mutex collection for the SAS$^+$ task is also computed. We have chosen to let this serve as the basis for mutex constraining our PDBs as it was readily available.

5.2 Configurations

We have experimented with all relevant combinations of the approaches discussed in the previous chapter as well as blind search and the M&S heuristic, to relate the results to other approaches. The configurations and the abbreviations we will refer to them by are:

- “i” The iPDB-method which is the best known general PDB heuristic known by the author, see Section 4.1.
- “mc-i” Mutex constrained iPDB, see Section 4.2.
- “mc-i-p” Variable pruned mutex constrained iPDB, see Section 4.2.2.
- “E” The extended version of the iPDB-method called EPDBG, see Section 4.1.
- “mc-E” Mutex constrained EPDBG, see Section 4.2.
- “mc-E-p” Variable pruned mutex constrained EPDBG, see Section 4.2.2.
- “M&S” The currently leading heuristic for optimal planning which is also based on abstractions, see Nissim *et al.* (2011).

"blind" Blind search, i.e. A* with a heuristic function that is zero for goal states and one for all other states.

The configuration space is rather big for all of the above approaches except for blind search. To this end, we have performed several experiments with each approach and different configuration values. However, the variation in the results was very little and for brevity and to facilitate an unbiased comparison we only present the results using the configuration performing best with the iPDB-method for all PDB heuristics and the best performing configuration for the M&S heuristic.

All PDB heuristics use 1000 samples and requires a minimum improvement of 1% when evaluating collection neighborhoods. Each PDB is limited to a maximum of 2.000.000 states and the PDB collection is limited such that the total number of states is no more than 20.000.000 states. Due to the efficient implementation (Sievers *et al.* , 2012), this can be stored in approximately 80 megabytes of ram. The M&S heuristic use the *DFP-bop* configuration described in Nissim *et al.* (2011) with an abstraction size limit of 200.000 states.

All experiments where performed on quad-core ProLiant SL2x120z G6 machines with each planner instance running on a single core with a time limit of 30 minutes and a memory limit of 2 GB for each task.

5.3 Results

Table 5.1 shows the coverage data for each configuration, i.e. the number of solved instances in each problem category of the benchmark set within the time and memory limit. Coverage is a function of the trade-off between the informedness of a heuristic and the effort needed to compute it and has so far been used to determine the winner of IPC. It gives a measure of how many tasks a heuristic can solve with limited CPU time, which is preferable as we do not have resources to compare CPU time on all tasks as some optimal plans simply take to much time to find.

All heuristics improve performance compared to blind search. Furthermore, the iPDB-method is on par with the M&S with only a single task less solved in total. Comparing the three iPDB configurations, we see that constrained abstractions and variable pruning does neither increase nor decrease performance. However, we see a significant improvement when moving to EPDBG. Here, constrained abstraction gives a little bump in performance on the parking category

coverage	i	mc-i	mc-i-p	E	mc-E	mc-E-p	M&S	blind
barman	4	4	4	4	4	4	4	4
elevators	16	16	16	**18**	**18**	**18**	10	9
floortile	2	2	2	2	2	2	**7**	2
nomystery	16	16	16	**20**	**20**	**20**	18	8
openstacks	13	13	13	13	13	13	13	13
parcprint.	7	7	7	7	7	7	**13**	6
parking	5	5	5	5	**7**	**7**	0	0
pegsol	0	0	0	0	0	0	**19**	17
scanalyzer	**10**	**10**	**10**	**10**	**10**	**10**	9	9
sokoban	**20**	**20**	**20**	**20**	**20**	**20**	19	16
tidybot	**14**	**14**	**14**	**14**	**14**	**14**	0	8
transport	6	6	6	**11**	**11**	**11**	7	6
visitall	**16**	**16**	**16**	**16**	**16**	**16**	9	9
woodwork.	2	2	2	0	0	0	**4**	2
sum (280)	131	131	131	140	**142**	**142**	132	109

Table 5.1: Comparison of solved task over 14 IPC benchmark problem categories with 280 tasks in total. Best results are highlighted in bold.

suggesting that mutex relations occur between goal variables for tasks in this category.

Note the difference in performance between all PDB heuristics and M&S in the pegsol and tidybot categories. For all pegsol tasks, the variable selection procedure for the PDB heuristics does not end before the 30 minutes time limit because all tasks contains unusually many goal variables causing an excessive amount of pattern collections to be considered. There are many solutions to this and a simple one is to put a time limit on the variable selection procedure. If we set this limit to 15 minutes, the PDB heuristics solve from eighteen to twenty pegsol tasks. This is also what causes the difference between the iPDB and EPDBG configurations on the woodwork tasks and we expect that something similar occurs with M&S for the tidybot tasks.

Table 5.2 shows the number of abstract mutex states encountered in the tasks solved by all mutex constrained configurations (variable pruning does not change the numbers, thus we leave out mc-i-p and mc-E-p). The first row indicates that the same abstractions are used by both configurations for tasks in the barman category.

For categories where the entries are zero, we can conclude that constrained abstractions cannot lead to increased informedness and that variable pruning has no effect. Unfortunately, this is the case in 7 of the 13 problem categories with solved tasks. The reason is either that mutex constraints do not occur

mutex states	**mc-i**	**mc-E**
barman	336060	336060
elevators	0	0
floortile	0	0
nomystery	0	0
openstacks	0	0
parcprinter	0	0
parking	0	**1772538**
scanalyzer	40430408	**104681469**
sokoban	165925	**240909**
tidybot	0	0
transport	0	0
visitall	0	0
woodworking	87	0
SUM	40932480	**107030976**

Table 5.2: Comparison of the number of abstract mutex states encountered in the solved tasks of the benchmark sets grouped by problem categories. Best results are highlighted in bold.

between the variables we consider to be important in the remaining tasks or more likely, because our approximation of the mutex collections is to weak to capture them.

For categories with entries greater than zero, we might see increased informedness as a result of using constrained abstractions. Furthermore we may see either a decrease or an increase in computation time due to the fact that it introduces some overhead in the successor generator data structure, but also has a chance of decreasing the size of the search space. Note that the only place we see a correspondence between Table 5.2 and 5.1 is for the parking category tasks where we have more solved tasks as an effect of mutex constraining.

Table 5.3 shows the total time spent in the variable selection procedure for tasks solved by all configurations. Recall that this is the only computation time needed for a PDB heuristic. We see how mutex constraining incurs an overhead of 10-14% and variable pruning incurs a reduction of 6-16% on computation time. Note how much more time the EPDBG configurations spend on variable selection than what is spend by the iPDB configuration.

Table 5.4 shows the total number of states evaluated by A* for tasks solved by all configurations (variable pruning does not affect this measure so we leave out mc-E-p and mc-i-p). We have left out configuration i-mc because it agrees with configuration "i" on all numbers. This means that we observe no positive

computation time	**i**	**mc-i**	**mc-i-p**	**E**	**mc-E**	**mc-E-p**
SUM (203)	**9425**	10348	9678	15824	17820	15006

Table 5.3: Comparison of the total computation time for the tasks that are solved by all PDB configurations. Best results are highlighted in bold.

evaluations	**i**	**E**	**mc-E**
barman	17454343	17454343	17454343
elevators	47129902	**18975485**	**18975485**
floortile	787060	787060	787060
nomystery	10124698	**292071**	**292071**
openstacks	41761969	41761969	41761969
parcprinter	594624	**505**	**505**
parking	11798851	11798851	**10628797**
scanalyzer	31338699	23647995	**23627228**
sokoban	5698964	5698964	5685169
tidybot	3247287	3247287	3247287
transport	3231779	**807045**	**807045**
visitall	2059400	**2018761**	**2018761**
SUM	175227576	126490336	**125285720**

Table 5.4: Comparison of the number of nodes evaluated by A* during search for the tasks that are solved by all PDB configurations, grouped by problem categories. Best results are highlighted in bold.

effect of using constrained abstractions with the iPDB-method, not even in the categories where we detect mutex states.

However, we see that the EPDBG heuristic is more informed than iPDB in five out of twelve categories, three of which has better coverage results in Table 5.1. Furthermore, mutex constraining leads to an even more informed heuristic for the parking and scanalyzer problems. For parking we see a reduction of approx. 10% in the number of expanded nodes. The EPDBG configuration expands approx. 30% fewer nodes than iPDB in total. We have not included the M&S heuristic in Table 5.4 as this would require us to exclude the tasks that are not solved by all configurations to maintain sensible numbers. However, M&S has fever evaluations in four out of ten categories where all configurations have solved tasks. Yet, it still evaluates approx 60% more states than the iPDB-method in total.

5.3.1 Summary of Results

The analysis of the data obtained by our experiments supports our claims and expectations based on our theoretical derivation and analysis of constrained abstractions, redundant variable pruning, and extended pattern database generation. Below we summarize the most important points of the experimental results:

- Mutex constraining does not decrease informedness of PDB heuristics. In some cases it increases the informedness of PDB heuristics significantly as it affects coverage positively.
- Mutex constraining PDBs incurs an overhead in computation time.
- Modifying the successor generator is a viable implementation approach to mutex constraining PDBs. The overhead in computation time is insignificant, as it causes no reduction in coverage.
- Redundant variable pruning leads to a decrease in computation time without affecting the informedness of mutex constrained PDB heuristics. The decrease is considered insignificant as it does not affect coverage.
- EPDBG increases both computation time and informedness compared to iPDB.
- EPDBG is superior to iPDB as it leads to a significant increase in coverage.

We stress that the value of the experimental data and therefore also the above statements should be interpreted with precaution. Most of the results may be subject to change if we consider other limitations for time and memory for the search and different configuration parameters for the PDB heuristics. Furthermore, the data and associated analysis only relates to the benchmark set under consideration.

However, the analysis serves the intended purpose of evaluating the practical usefulness of our theoretical results and observations. And for this particular benchmark set, created by the automated planning community involved with IPC, this chapter documents that our results lead to a better heuristic with increased performance.

Chapter 6

Future work

As this project has shown, there is still room for improvement in the area of general heuristics. While state-of-the-art heuristic methods are still evolving, the return-on-investment is gradually lowering as heuristics become better (Helmert & Röger, 2008).

However, abstraction heuristics such as the M&S heuristic and PDB heuristics has much in common and one may benefit from the same observations that improved the other. Considering constrained abstractions and redundant variable pruning for the M&S heuristic would be an interesting road to follow. This has, to our knowledge, not yet been investigated. The practical effect of mutex constraining were few while our theoretical analysis showed more potential. This may be because the approximation of mutex collection computed by FD is too poor. Investigating if a better approximation is obtainable within reasonable computation time would be a natural next step.

Even though we have made improvements to the variable selection procedure of iPDB, it still has several weaknesses that are not easily overcome. For instance, we have no guarantee that two or more variables that are all irrelevant to a pattern, would not improve the induced heuristic, if they were added to the pattern simultaneously. The selection technique is an obvious area for future work.

Most of the general heuristics that perform well on the IPC benchmark test set rely on some abstraction or relaxation strategy that is of immense importance for their performance, yet not straight forward to generalize while preserving good heuristic estimates in general. For the PDB heuristic, the variable selection is vital to the effectiveness of the heuristic. The iterative approaches discussed in this thesis works well, but are not competitive with problem specific solvers based on pattern databases (Felner *et al.* , 2004; Korf & Felner, 2007). To this end, investigating a structured way of providing problem specific heuristic information to planning systems presents itself as an interesting avenue for future research. Furthermore, succeeding with this could turn the general problem solvers into cost-effective solvers for real-world problems.

We have done preliminary experiments that show great potential for a method we call *structural pattern database generation* (SPDBG). SPDBG builds upon the mutex constrained iPDB-method, but the iterative variable selection is replaced by a combinatorial approach based on user-specified information for a problem or a set of problems. The user is presented with a set of different variable types identified in the planning task or in the set of planning tasks under consideration. Variables of the same type such as the balls or the grippers in the Gripper tasks often occur in planning tasks and can automatically and efficiently be identified (Domshlak *et al.* , 2012). Based on the categorization, the user chooses several combinations of variable types which each gives rise to a set of patterns, where each pattern is one of the possible combination of variables with the chosen types.

Example 6.1 (structural pattern database generation) *Consider a set of* Gripper *tasks with a varying number of balls. The following variable types are easily identified: balls, grippers, robot(s). The user might specify the following three combinations: 1: three balls, 2: two balls, (one) robot, 3: three balls, two grippers, (one) robot. For a task with n balls, combination 1 would give rise to a oattern for each combination of three balls, combination 2 would give rise to a pattern for each combination of two balls and the robot, etc.*

We have also experimented using different learning strategies on sets of related problems to select which types of variables to consider. All in all, the preliminary results exhibit some ambiguity. First of, the heuristic now becomes dependant on the learning strategy or the information specified by the user. In the latter case this implies that the user needs to analyze or experiment with the problem set to make informed choices. For some related planning tasks, we get a dramatic improvement in CPU time whereas other tasks show a degradation in both informedness and CPU time when comparing SPDBG to the general pattern database heuristics. The results indicate that there is potential, but further work and research is required to make the approach viable.

CHAPTER 7

Conclusion

This project set out with the goal of improving general heuristics for automated planning. A discussion of the inherent complexity of planning and heuristic search has shown that heuristic search has limited use in the completely general setting. Yet, for less general settings and in particular for tractable problems, heuristic search is the forefront of automated planning.

The admissible PDB heuristic has been successfully applied to find optimal solutions to big combinatorial problems but is also on par with state-of-the-art for general heuristic via the iPDB-method.

We have combined well known mutex constraining techniques with novel approaches and considered the effects on the iPDB-method. Our results are theoretical improvements and refinements of the iPDB-method. Moreover our redundant variable pruning technique generalizes to all heuristics, using constrained abstraction. We have also considered how to make the results viable to practical applications and implemented the new heuristics in the FastDownward planning system.

Our main result is *variable pruned mutex constrained extended pattern database generation* of which experimental evaluation shows that the improvements carry over to practice and sets the pattern database heuristic in front of currently leading heuristics for optimal planning on the benchmark set considered.

APPENDIX A

Source

Documentation for the FastDownward planning system is available at:

www.fast-downward.org

To obtain and build the planner the following tools are needed: Mercurial, GNU C++ compiler, GNU make, gawk, flex, bison and Python 2.7+. The original source code can be obtained by the command:

```
hg clone http://hg.fast-downward.org
```

Our fork of the project, where the heuristics discussed in this thesis are implemented can be obtained by the command:

```
hg clone https://bitbucket.org/mettienne/fastdownward
```

Change into the "src" directory and run the script "build_all" to build the planner. We refer to the FastDownward documentation instructions on how to run the planner. The configuration space for our heuristic is given below:

```
impdb(pdb_max_size=2000000, collection_max_size=20000000,
      num_samples=1000, min_improvement=10,
      cost_type=NORMAL, mutex=FALSE, extend=FALSE)
```

- pdb_max_size (int): max number of states per PDB.
- collection_max_size (int): max number of states for collection.
- num_samples (int): number of samples.
- min_improvement (int): minimum improvement while hill climbing.
- cost_type (NORMAL, ONE, PLUSONE): Action cost adjustment type. No matter what this setting is, axioms will always be considered as actions of cost 0 by the heuristics that treat axioms as actions.
  - NORMAL: all actions are accounted for with their real cost.
  - ONE: all actions are accounted for as unit cost.
  - PLUSONE: all actions are accounted for as their real cost +1 (except if all actions have original cost 1, in which case cost 1 is used). This is the behaviour known from the LAMA planner.
- merge (bool): use constrained abstractions (mc-iPBD/mc-EPDBG).
- mutex (bool): use extended variable selection (EPDBG) instead of iPDB.
- prune (bool): use mutex pruning.

Bibliography

Bäckström, Christer. 1992. Equivalence and Tractability Results for SAS+ Planning. *Pages 126–137 of:* Nebel, Bernhard, Rich, Charles, & Swartout, William R. (eds), *KR*. Morgan Kaufmann.

Bäckström, Christer. 1995. *Five Years of Tractable Planning.* LiTH-IDA-R. Univ., Inst. för Datavetenskap.

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

Bonet, Blai, & Geffner, Hector. 1999. Planning as Heuristic Search: New Results. *Pages 360–372 of:* Biundo, Susanne, & Fox, Maria (eds), *ECP*. Lecture Notes in Computer Science, vol. 1809. Springer.

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

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

Culberson, Joseph C., & Schaeffer, Jonathan. 1998. Pattern Databases. *Computational Intelligence*, **14**(3), 318–334.

Dechter, Rina, & Pearl, Judea. 1985. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*, **32**(3), 505–536.

Domshlak, Carmel, Katz, Michael, & Shleyfman, Alexander. 2012. Enhanced Symmetry Breaking in Cost-Optimal Planning as Forward Search. *In:* McCluskey, Lee, Williams, Brian, Silva, José Reinaldo, & Bonet, Blai (eds), *ICAPS*. AAAI.

Edelkamp, Stefan. 2001. Planning with Pattern Databases. *Pages 13–24 of: PROCEEDINGS OF THE 6TH EUROPEAN CONFERENCE ON PLANNING (ECP-01).*

Edelkamp, Stefan. 2006. Automated Creation of Pattern Database Search Heuristics. *Pages 35–50 of:* Edelkamp, Stefan, & Lomuscio, Alessio (eds), *MoChArt*. Lecture Notes in Computer Science, vol. 4428. Springer.

Edelkamp, Stefan, & Hoffmann, Jorg. 2004. PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition. *Technical Report.*

Edelkamp, Stefan, Englert, Roman, Hoffmann, Jörg, dos S. Liporace, Frederico, Thiébaux, Sylvie, & Trüg, Sebastian. 2011. Engineering Benchmarks for Planning: the Domains Used in the Deterministic Part of IPC-4. *CoRR*, **abs/1110.1016**.

Felner, Ariel, Korf, Richard E., & Hanan, Sarit. 2004. Additive Pattern Database Heuristics. *J. Artif. Intell. Res. (JAIR)*, **22**, 279–318.

Ghallab, Malik, Nau, Dana S., & Traverso, Paolo. 2004. *Automated planning - theory and practice.* Elsevier.

Haslum, Patrik, Bonet, Blai, & Geffner, Hector. 2005. New Admissible Heuristics for Domain-Independent Planning. *Pages 1163–1168 of:* Veloso, Manuela M., & Kambhampati, Subbarao (eds), *AAAI*. AAAI Press / The MIT Press.

Haslum, Patrik, Botea, Adi, Helmert, Malte, Bonet, Blai, & Koenig, Sven. 2007. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. *Pages 1007–1012 of: AAAI*. AAAI Press.

Helmert, Malte. 2003. Complexity results for standard benchmark domains in planning. *Artif. Intell.*, **143**(2), 219–262.

Helmert, Malte. 2004. A Planning Heuristic Based on Causal Graph Analysis. *Pages 161–170 of:* Zilberstein, Shlomo, Koehler, Jana, & Koenig, Sven (eds), *ICAPS*. AAAI.

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

Helmert, Malte. 2008. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition.* Lecture Notes in Computer Science, vol. 4929. Springer.

Helmert, Malte, & Röger, Gabriele. 2008. How Good is Almost Perfect? *Pages 944–949 of:* Fox, Dieter, & Gomes, Carla P. (eds), *AAAI*. AAAI Press.

Helmert, Malte, Mattmüller, Robert, & Röger, Gabriele. 2006. Aproximation Properties of Planning Benchmarks. *Pages 585–589 of:* Brewka, Gerhard, Coradeschi, Silvia, Perini, Anna, & Traverso, Paolo (eds), *ECAI*. Frontiers in Artificial Intelligence and Applications, vol. 141. IOS Press.

Helmert, Malte, Haslum, Patrik, & Hoffmann, Jörg. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. *Pages 176–183 of:* Boddy, Mark S., Fox, Maria, & Thiébaux, Sylvie (eds), *ICAPS*. AAAI.

Kautz, Henry A., & Selman, Bart. 1996. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. *Pages 1194–1201 of:* Clancey, William J., & Weld, Daniel S. (eds), *AAAI/IAAI, Vol. 2*. AAAI Press / The MIT Press.

Korf, Richard E., & Felner, Ariel. 2007. Recent Progress in Heuristic Search: A Case Study of the Four-Peg Towers of Hanoi Problem. *Pages 2324–2329 of:* Veloso, Manuela M. (ed), *IJCAI*.

Korf, Richard E., Reid, Michael, & Edelkamp, Stefan. 2001. Time complexity of iterative-deepening-A^*. *Artif. Intell.*, **129**(1-2), 199–218.

McDermott, Drew V. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, **21**(2), 35–55.

Nissim, Raz, Hoffmann, Jörg, & Helmert, Malte. 2011. Computing Perfect Heuristics in Polynomial Time: On Bisimulation and Merge-and-Shrink Abstraction in Optimal Planning. *Pages 1983–1990 of:* Walsh, Toby (ed), *IJCAI*. IJCAI/AAAI.

Russell, Stuart J., & Norvig, Peter. 2010. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Savitch, Walter J. 1970. Relationships Between Nondeterministic and Deterministic Tape Complexities. *J. Comput. Syst. Sci.*, **4**(2), 177–192.

Sievers, Silvan, Ortlieb, Manuela, & Helmert, Malte. 2012. Efficient Implementation of Pattern Database Heuristics for Classical Planning. *In:* Borrajo, Daniel, Felner, Ariel, Korf, Richard E., Likhachev, Maxim, López, Carlos Linares, Ruml, Wheeler, & Sturtevant, Nathan R. (eds), *SOCS*. AAAI Press.

Tomita, Etsuji, Tanaka, Akira, & Takahashi, Haruhisa. 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, **363**(1), 28–42.