Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning

Patrik Haslum and **Adi Botea**
NICTA & Australian National University
firstname.lastname@nicta.com.au

Malte Helmert
Albert-Ludwigs-Universität Freiburg
helmert@informatik.uni-freiburg.de

Blai Bonet
Universidad Simón Bolívar
bonet@ldc.usb.ve

Sven Koenig
USC
skoenig@usc.edu

Abstract

Heuristic search is a leading approach to domain-independent planning. For cost-optimal planning, however, existing admissible heuristics are generally too weak to effectively guide the search. Pattern database heuristics (PDBs), which are based on abstractions of the search space, are currently one of the most promising approaches to developing better admissible heuristics. The informedness of PDB heuristics depends crucially on the selection of appropriate abstractions (patterns). Although PDBs have been applied to many search problems, including planning, there are not many insights into how to select good patterns, even manually. What constitutes a good pattern depends on the problem domain, making the task even more difficult for domain-independent planning, where the process needs to be completely automatic and general. We present a novel way of constructing good patterns automatically from the specification of planning problem instances. We demonstrate that this allows a domain-independent planner to solve planning problems optimally in some very challenging domains, including a STRIPS formulation of the Sokoban puzzle.

Introduction

Pattern databases (PDBs; Culberson & Schaeffer 1998) are one of the most successful approaches to creating admissible heuristics for single-agent search. PDB heuristics have been used to solve a number of challenging search problems, including both benchmark problems such as the $(n^2 - 1)$-puzzle, and application problems, such as multiple sequence alignment (Felner, Korf, & Hanan 2004). PDBs are abstraction heuristics, obtained by abstracting away all but a part of the problem (the *pattern*) small enough to be solved optimally for every state by blind exhaustive search. The results are stored in a table in memory (the *pattern database*) and define an admissible and consistent heuristic function by mapping states into corresponding abstract states and reading their associated values from the table. Heuristic estimates from several abstractions can be combined by taking their maximum or, under certain conditions, their sum.

The main problem with PDB heuristics is that the quality of the heuristic depends crucially on the selection of a collection of abstractions (patterns) that are appropriate to the problem domain, or even the problem instance. What makes this problem hard is that the time and memory required to compute and store the pattern database limits the size of the patterns that can feasibly be used. There is some knowledge about what constitutes good quality patterns for some specific domains where PDB heuristics have been successfully applied (*e.g.* Holte *et al.* 2006; Korf & Felner 2007), but this has been obtained by experimentation and insight into the structure of the problem. This is not satisfactory for domain-independent planning, where the pattern selection process needs to be completely automatic and general. That is the problem we attack.

Pattern databases have been applied to domain-independent planning before: Edelkamp's (2001) was the first application of PDB heuristics in planning, and introduced abstractions based on multi-valued state variables. Haslum, Bonet & Geffner (2005) refined the abstraction, by the inclusion of global (mutex) constraints, obtaining more accurate heuristic estimates for patterns of a fixed size. They also introduced the idea of an "incremental" construction of the collection of patterns. However, their methods apply only to regression planning, and do not easily generalize to progression (forward search). This is a weakness, since progression is in some domains inherently more efficient and also easier to extend beyond STRIPS. Edelkamp (2006) cast pattern selection as an optimization problem, and used a genetic algorithm, with the mean heuristic value as the fitness function, to find good solutions.

Building on these works, our main contribution is a principled approach to pattern selection. The core of the approach is the measure of heuristic quality by which candidate collections of patterns are evaluted, based on estimating what the size of the search tree would be for a given PDB heuristic using a result due to Korf, Reid and Edelkamp (2001). We show this measure to be more informed than the mean heuristic value, proposed by Edelkamp. While the result has previously been used to explain experimentally observed differences in search performance arising from different PDB heuristics (Holte *et al.* 2006), we develop it into a practically usable method for constructing a good PDB.

Background

A propositional STRIPS planning problem P with additive costs consists of a finite set of atoms, a finite set of actions,

a completely specified initial state, s_0, and a goal condition, G. Each action a has a set of precondition atoms (*pre*(a)), which must hold in a state for it to be applicable, and sets of atoms made true (*add*(a)) and false (*del*(a)), respectively, when it is applied. A plan is an applicable sequence of actions that leads to a state satisfying the goal condition. Every action has a cost, *cost*(a), and the cost of a plan is the sum of the costs of the actions in the plan.

Parallel to the propositional representation of planning problems, we consider a representation by multi-valued state variables (*i.e.* variables taking values from a finite set other than the set $\{\text{true}, \text{false}\}$), which are implicit in many planning problems. These implicit variables correspond to a particular type of invariants, namely sets of atoms such that exactly one atom in the set is true in any reachable state. Methods for automatically extracting such "exactly-one" invariants from STRIPS encodings have been proposed (*e.g.* Helmert 2006). The rationale for the use of a multi-valued state variable encoding as the basis for abstractions is that it allows PDBs to be represented much more compactly: the PDB for a single variable with n values has n entries, whereas the corresponding PDB in a propositional encoding would have 2^n entries ($2^n - n$ of which are irrelevant, since they correspond to states that can never be reached).

Abstractions and Pattern Databases An abstraction of the problem is defined by a subset, A, of the state variables, by simply removing from the preconditions and effects of all actions, and from the initial state and goal, all variables (atoms) not in A. We call the set A the *pattern*, and denote by $h^A(s)$ the minimum cost of reaching a goal state from the abstract state corresponding to state s, in the state space of the abstract problem: this is a lower bound on the corresponding cost in the state space of the original problem P, and thus h^A is an admissible heuristic. It is also consistent. The size of the abstract state space is bounded by $\prod_{v \in A} |D_v|$, where D_v is the domain of variable v, *i.e.*, the set of values it can take. When this is sufficiently small, the optimal cost function $h^A(s)$ can be computed for all s by a breadth-first search in backward direction from all goal states and stored in a table, which constitutes the pattern database for pattern A.

Constrained Abstraction Because the abstraction ignores all conflicts involving variables not in the pattern A, states in the abstract space may become reachable even though they do not correspond to any reachable state in the real state space. This often causes the heuristic h^A to underestimate the cost of reaching the goal from a state s more than necessary and also weakens the ability of the heuristic to detect deadlock states (states from which no goal state is reachable). Constrained abstraction (Haslum, Bonet, & Geffner 2005) improves the accuracy of h^A by enforcing in the abstraction invariant conditions that hold in the original problem. Of particular interest are binary "at-most-one" invariants, *i.e.* static mutexes. In solving the abstract problem, application of an action to an abstract state is disallowed if the preconditions of the action and the abstract state together violate a mutex. The resulting heuristic function remains admissible and consistent.

Collections of Patterns

Before we go into the question of how to select a collection of patterns, we establish some results about the heuristic functions that can be constructed from such a collection.

The Canonical Heuristic Function An admissible heuristic function h is said to *dominate* another admissible heuristic h' iff $h(s) \geqslant h'(s)$, for all s. Clearly, if pattern A is a subset of pattern B, h^B dominates h^A. Note that this also means that h^A yields lower bounds (*i.e.*, is an admissible heuristic) for optimal cost in the abstract state space defined by pattern B.

Given two PDB heuristics h^A and h^B, the heuristic function $h(s) = \max(h^A(s), h^B(s))$ is also admissible and clearly dominates both h^A and h^B alone. If the set of actions that affect some variable in A is disjoint from the set of actions that affect any variable in B, the heuristic $h(s) = h^A(s) + h^B(s)$ is also admissible. In this case, we say the patterns are *additive* (this additivity condition is only sufficient, not necessary, but has the advantage that it is easy to check). A set of patterns is additive, according to this condition, iff all patterns in the set are pairwise additive. Clearly, $h^A(s)+h^B(s)$ dominates $\max(h^A(s), h^B(s))$. Note, however, that $h^{A \cup B}$, *i.e.* the PDB for a pattern containing all variables in both A and B, dominates both the sum and maximum of h^A and h^B.

In general, given a collection of patterns, $C = \{P_1, \ldots, P_k\}$, where the additivity condition stated above holds between some of the patterns but not all, there is a unique way of combining the corresponding PDB heuristics into an admissible (and consistent) heuristic function that dominates all others. We will call this the *canonical heuristic function* of the pattern collection and denote it by h^C.

Theorem 1 *Let $C = \{P_1, \ldots, P_k\}$ be a collection of patterns, and let A be the collection of all maximal (w.r.t. set inclusion) additive subsets of C: the canonical heuristic function of C is*

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

The canonical heuristic can be simplified by removing from it any sum over an additive set S for which there is another additive set S' 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'. Nevertheless, in the worst case, the number of additive non-dominated subsets of a collection may be exponential.

Finding the maximal additive subsets of a collection of patterns is equivalent to the problem of finding all maximal (*w.r.t.* set inclusion) cliques in a graph, which requires exponential time in the worst case because there can be exponentially many such cliques. The algorithm by Tomita, Tanaka & Takahashi (2004) runs in time polynomial in the number of maximal cliques, which we found to be efficient enough.

Dominance in a Collection As noted above, if all variables in pattern A are also included in pattern B, h^B dominates h^A. However, this does not always mean that pattern

A is of no use, since it may be additive with some patterns with which B is not. In a collection C, a pattern P is redundant if $h^C = h^{C-\{P\}}$, which is the case if 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 condition is always redundant.

Pattern Construction

The main question we address is the following: Given a planning problem, with (multi-valued) state variables $\{V_1, \ldots, V_n\}$, and a limit on the amount of memory that may be allocated to PDBs, how do we construct a collection of patterns, $C = \{P_1, \ldots, P_k\}$, that respects the memory limit and whose canonical heuristic function gives the best search performance?

Estimating Search Effort

Korf, Reid and Edelkamp (2001) develop a formula for the number of nodes expanded by a tree search (IDA*) with given cost bound c, using an admissible and consistent heuristic h for pruning. Their formula (slightly rewritten) is

$$\sum_{k=0,\ldots,c} N_{c-k} P(k) \tag{1}$$

where N_i is the number of nodes whose accumulated cost (g value) equals i and P is the so called *equilibrium distribution* of the heuristic function h: $P(k)$ is the probability that $h(n) \leqslant k$, where n is a node drawn uniformly at random from the search tree up to the cost limit c. Technically, the formula holds only in the limit of large c. The same formula holds also for a graph search (A*), but in that case the equilibrium distribution is defined by random drawing of nodes uniformly over the state space up to cost c, instead of the search tree. In the presence of transpositions (multiple paths to the same state) these are not the same.

¿From our perspective, c and N_i are fixed. However, because we can influence the heuristic by our selection of patterns, we can influence P. Thus, ideally, we want to select a collection of patterns whose associated PDB heuristic results in an equilibrium distribution that minimizes (1). ¿From N_i, P and c we can obtain an absolute measure of the quality of each candidate heuristic. These parameters are not known to us, but can be estimated. However, we do not need such an estimate of absolute quality. We only need to determine which of two given heuristic functions h and h' is better. Formula (1) suggests that the number of states whose heuristic value is increased is more important than the magnitude of the increase, in particular increasing the estimated cost of states whose current estimate is small.

In fact, we will be considering an even more restricted question: given a base heuristic h_0, and two alternative improvements h and h', which improves more over h_0?

Pattern Construction as Search

The problem of selecting the best collection of patterns, subject to the "at least one goal variable" restriction and the given memory limit, is a discrete optimization problem, which we approach by defining a search space whose states are pattern collections and in which the neighbourhood of a state is defined by a set of modifications to the collection. In this space, we then search for a good solution.

The starting point of the search is the collection consisting of one pattern for each goal variable, each containing only that variable. (We assume that the memory limit is large enough to fit this collection. This is a reasonable assumption due to the way variables are created from the planning problem.) ¿From a given collection of patterns, $C = \{P_1, \ldots, P_k\}$, a new collection C' can be constructed by selecting a pattern $P_i \in C$, a variable $V \notin P_i$, and adding the new pattern $P_{k+1} = P_i \cup \{V\}$ to the collection, provided the total PDB size of the new collection does not exceed the memory limit. This defines the search neighbourhood of C.

As each neighbour of a pattern collection C contains C, heuristic quality is non-decreasing. However, due to the memory limit finding the true optimum requires, in general, an exhaustive search through the space of pattern collections. As this is not feasible, we settle for finding a local optimum, using a local search method. In our experiments we have used simple hill climbing. The search starts with the initial collection described above, repeatedly evaluates the neighbourhood of expanded pattern collections and selects the best neighbour to be the current collection in the next iteration, ending when no extension, permitted by the size limit, of the current collection results in a significant improvement. PDBs corresponding to patterns in the collection are computed along the way.

Evaluating the Neighbourhood

Evaluating the neighbourhood is the critical step in our search for a good pattern collection. We need to rank the relative quality of the pattern collection in the neighbourhood of the current collection, and determine whether any of them offers a significant improvement.

Edelkamp (2006) suggests measuring the quality of a heuristic by its mean value, which is also related to the search effort but which is less discriminating than formula (1), since heuristics with very different equilibrium distributions can have the same mean. The mean heuristic value of a single PDB is easily obtained by a linear scan of the PDB, but the mean value of the canonical heuristic function is not so easily computed. Edelkamp solves this problem by forcing all PDBs to be additive, through cost relaxation, and averaging their mean values, but this weakens the heuristic and assumes that values of different PDBs are not correlated. Also, it is not clear how the mean is defined if the heuristic value of some states is infinite, *i.e.*, when deadlocks are recognised by the heuristic.

As an alternative, we derive a relative measure of the quality of a pattern collection based on formula (1). A collection C' in the neighbourhood of C differs from C only by the addition of one new pattern, P_{k+1}. Thus, $h^{C'}(s)$ can never be less than $h^C(s)$, for any state s, since C' subsumes C. Formula (1) predicts that the improvement, *i.e.*, the search effort saved, by using C' instead of C is

$$\sum_{k=0,\ldots,c} N_{c-k}(P(k) - P'(k)) \tag{2}$$

where P and P' are the equilibrium distributions of h^C and $h^{C'}$, respectively. Suppose we draw a sample $\overline{n}$ of m nodes uniformly at random from the search tree: the resulting estimate of (2) can be written as

$$\frac{1}{m}\sum_{n_i}\sum_{h^C(n_i)\leqslant k<h^{C'}(n_i)} N_{c-k} \tag{3}$$

Since N_k, the number of nodes in the search tree within cost bound k, tends to grow exponentially in general, we can reasonably assume that N_k dominates $\sum_{i<k} N_i$, and approximate the above expression with

$$\frac{1}{m}\sum_{\{n_i \mid h^C(n_i)<h^{C'}(n_i)\}} N_{c-h^C(n_i)} \tag{4}$$

This approximation greatly simplifies the evaluation, since for each node n_i in the sample we only need to determine if $h^{C'}(n_i) > h^C(n_i)$, rather than having to determine the exact value of $h^{C'}(n_i)$. A possible additional simplifying approximation is to ignore the weight that formula (4) attaches to each sample, *i.e.*, simply counting the number of nodes for which $h^{C'}$ is greater than h^C. (In the following, we refer to this as the *counting approximation*.) The impact of this approximation on heuristic quality is evaluated experimentally later.

Although the method outlined above is conceptually simple, there are a number of technicalities in it that are both involved and important for the performance of the search, which we discuss next.

Comparing $h^C(n)$ and $h^{C'}(n)$ For each node n in the sample, we need to determine if $h^{C'}(n) > h^C(n)$. Since we maintain PDBs for the current collection, $h^C(n)$ is available by simply evaluating the heuristic function. Because the collection C contains every pattern in C' except for the new pattern P_{k+1}, $h^{C'}(n) > h^C(n)$ holds iff $h^{P_{k+1}}(n) > h^C(n) - \sum_{P_i\in S-\{P_{k+1}\}} h^{P_i}(n)$, for some additive subset $S \subseteq C'$ that includes P_{k+1}.

The only unknown quantity in this condition is the value of $h^{P_{k+1}}(n)$. This could be obtained by computing the corresponding PDB, but, particularly for large patterns, this is costly, and also wasteful if the number of samples to be evaluated is small, relative to the size of the PDB. As an alternative, we use a search in the abstract state space defined by pattern $P_{k+1} = P_i \cup \{V\}$. Because we have already a PDB for P_i, and also for $\{V\}$ if V is a goal variable, we have good lower bounds for the abstract space, and the condition that we are testing provides a, typically rather tight, upper bound. This combines to make the search quite efficient.

Sampling the Search Space In order to perform the estimation to rank the neighbours of a pattern collection C, we need a random sample of nodes, $\overline{n}$, drawn uniformly from the search space up to cost bound c. If the space is a uniform and complete tree with branching factor b up to a depth of at least d, a uniformly sampled node within depth d can be found by a random walk of length l, where $l = i$ with probability $\frac{b^{i+1}-b^i}{b^{d+1}-1}$, that returns the node where the walk ends. However, the search space associated with a planning problem, is typically neither uniform nor a tree, and there are often dead-end states (*i.e.* states from which no action is applicable). Moreover, we do not know what is the relevant depth, since we do not know the solution.

Still, random walks is a simple method for efficiently sampling the reachable state space, so we use it as an approximation. The solution depth is estimated by the current heuristic, multiplied by a small constant (in our experiments 2) because the heuristic is underestimating. (For problems with non-uniform cost, the depth estimate is adjusted by the average cost/depth ratio, which is also estimated during the random walks.) To compensate for the imprecision of this estimate, random walk lengths are binomially distributed with the mean at the estimated depth. We constrain the random walk so as to avoid some known deadlock states (states for which the current heuristic value is infinite) that are efficiently detectable, and on encountering a dead-end state reset the walk to the initial state. Due to these compromises, our sample of states is not uniformly random over the search space, but it is still *relevant*, in the sense that it provides an estimate useful for ranking the neighbours of the current pattern collection.

Avoiding Redundant Evaluations Neighbourhood evaluation is the most time consuming part o the search. We use two techniques to avoid wasting time on evaluating unpromising neighbours.

The first is based on a static analysis: Adding a variable V to pattern P will only improve over the value of h^P if the variable directly influences any of those in the pattern. The causal graph is a common tool for analysing influences among variables in a planning problem: a variable V' can influence a variable V if some action that changes V has a precondition or effect on V'. Due to our use of constrained abstraction, V' also, potentially, influences the pattern P if some action that changes a variable in P has a precondition in P that is mutex with some value of V'. Variables that can not influence a pattern are not considered for extending it.

The second is statistical: After each $\frac{m}{t}$ samples, we calculate for each candidate collection a confidence interval for its "score" (*i.e.* how much it improves on the current pattern collection). If the upper end of the estimated interval for a collection C' is less than the lower end of the interval for a collection C'', it is unlikely that C' will turn out better than C'', so we do not evaluate C' any further. For the counting approximation we use Wilson's (1927) formula for the confidence interval, since the normal approximation is too imprecise when the estimated value is very small.

Ending the Search The pattern search comes to an end when no extension allowed by the size limit improves over the current pattern collection, but in many cases the point at which the effort of the search exceeds the value of the gain in heuristic accuracy occurs much earlier. Thus it is useful to consider more aggressive cut-off criteria, such as limiting the number of search iterations or setting a threshold and disregarding any extension with a smaller improvement. In the counting approximation of formula (4), the score of an extension is simply the fraction of samples for which the

Problem no.	Optimal length	Nodes exp.	Total time	Constr. time
7	26	9614	297.2	295.9
35	77	1122	177.1	177.0
54	82	89380	199.5	189.2
65	138	105236	164.7	154.9
78	135	3092044	768.2	336.6
83	164	971728	392.8	288.9
87	149	66719	124.0	117.3
95	25	5232	321.6	320.6
97	164	33559	286.9	283.5
102	149	114078	149.0	136.4
106	205	658844	823.9	752.7
107	38	89985	1272.0	1252.8
115	110	238867	394.6	366.8
118	172	392622	209.3	163.9
121	125	143407	547.3	530.6
125	125	187526	134.5	116.7
126	87	2088379	1014.2	695.8
129	99	36342	242.9	239.0
130	102	142576	261.6	245.2
131	76	170039	065.0	048.4
134	244	218202	695.4	670.1
137	177	3745013	1208.2	692.3
140	290	1097563	1116.8	963.6
141	134	103788	183.0	170.4
143	212	1680744	3607.2	3348.1
148	197	53370	721.9	716.2
150	135	2970347	1059.8	650.8
151	125	80472	168.3	160.2

Table 1: Summary of solved Microban problems. Unsolved problems in the test set are numbers 93, 99, 105, 108, 112 – 114, 117, 123, 133, 138, and 145.

heuristic value of the extension is greater than that of the current collection, thus normalised to $[0, 1]$. This makes it possible to chose a general threshold value.

Experiments

We evaluate the quality of the pattern collections found by the method outlined above by using the resulting heuristic in a cost-optimal forward search planner. The search for a plan is done using A*. We compare the result of using our method of evaluating pattern collections with the result of using the mean heuristic value, as suggested by Edelkamp (2006). Additionally, we test the impact of the counting approximation.

To evaluate the planner we use STRIPS encodings of two hard search problems, Sokoban and the 15-Puzzle, as well as the Logistics domain. Except where otherwise noted, results are obtained using the counting approximation and size limits of 20 million entries for the PDB collection and 2 million entries for any single PDB. Total memory use is limited to 1 GB; all problems reported unsolved are due to the A* search exhausting memory. Since the method is randomised, values reported are the median of 3 runs. Variation in the aggregated statistics between runs is relatively small (at most $\pm 10\%$ from the median), but a small number of instances exhibit much larger variance.

Sokoban The Sokoban puzzle[1] is a very difficult single-agent search problem, even for non-optimal domain-specific solvers (Junghanns & Schaeffer 2001). The problem instances we use are a collection of "introductory" problems, meant to be relatively easy for a human to solve, but hard enough to challenge any domain-independent planner.[2] Removing problems that are too easy (solved by blind search) and too hard (not solved by any planner we tried, including several state-of-the-art suboptimal planners), leaves 40 problems, with sizes ranging from 4 to 12 stones and (known) optimal solution lengths from 25 to 290 moves. The planner is able to solve 28 of these. The average (across solved problems) number of expanded nodes per instance is 679,650, and the average runtime is 600.4 seconds, of which 508.2 (84.6%) is spent constructing the PDB heuristic. Table 1 presents the results in detail.

Sokoban is an example of a domain where forward search is inherently more effective than regression: regression search using a variety of admissible heuristics (including those proposed by Haslum, Bonet, & Geffner 2005) does not solve any of our 40 test problems, or even all problems solved by blind forward search.

The 15-Puzzle The $(n^2 - 1)$-Puzzle is another extensively studied benchmark in the literature on search, and while domain-specific solvers are able to solve 24- and even some 35-Puzzle problems (Felner, Korf, & Hanan 2004), the 15-Puzzle is still a challenge for a domain-independent planner. We use Korf's set of 100 problems. The planner is able to solve 93 problems, using an average of 331,220 node expansions and 1,439.7 seconds, 1,381.9 (96%) spent constructing the PDB heuristic, per problem.

The 15-Puzzle domain is well suited to regression: Haslum, Bonet & Geffner (2005) demonstrate that their PDB heuristics yield good results, over a subset of 24 of Korf's problems. However, our pattern selection generates better heuristics for the same PDB size limit: over the same set of problems their total number of nodes expanded in search is 2,559,508 whereas we obtain a total number of 549,147 – less than one fourth.

Neighbourhood Evaluation by Mean Value We compare the results of our pattern selection method against using mean heuristic value, as suggested by Edelkamp (2006), to rank pattern collections in the search neighbourhood. The mean value of the canonical heuristic function is estimated by sampling. Due to the problems with defining a sensible mean value when some heuristic values are infinite, we restrict the comparison to the 15-Puzzle and Logistics domains, which have no (reachable) deadlock states. In these experiments we use a fixed limit on the number of iterations in the pattern search (set to roughly equal that resulting when using our threshold cut-off). Note that our method

[1] `http://en.wikipedia.org/wiki/Sokoban`.

[2] The collection is called "microban"; see `http://users.bentonrea.com/~sasquatch/sokoban/`. The STRIPS encodings of the problems are available from us.

here uses the same iteration limit, and hence results differ slightly from those reported above.

PDB heuristics constructed using the mean value perform generally worse. In the 15-Puzzle, the planner using this heuristic solves 66 problems, using an average of 657,380 node expansions, compared to 80 problems solved and an average of 418,730 node expansions (over problems solved by both versions) with the heuristic resulting from our method. The heuristic based on mean value ranking is not always worse, but if we count the number of "wins" for the two heuristics (a "win" meaning a problem solved with fewer expanded nodes), the ratio is $1 : 7$. The probability of this outcome under the hypothesis of an equal winning chance for both methods is less than 10^{-11}. In the Logistics domain (12 problems from the IPC-2 set, size 7 to 12), both planner versions solve the same set of instances, but, using the heuristic created by mean value ranking consistently expands more nodes, 176,850 compared to 23,992, on average. The variance across repeated runs is also significantly greater when using mean value ranking.

Impact of the Counting Approximation We compare the results of using an estimate of the parameters N_i in formula (4) against the results of using the counting approximation. As an estimate we used $N_i = \hat{b}^i$, where $\hat{b}$ is an estimate of the branching factor, obtained in the course of the random walk.

In the Sokoban domain, the heuristic obtained using estimated parameters is better, expanding an average of 551,290 nodes compared to 679,650 for the heuristic that results when using the counting approximation, across solved problems (both versions solve the same number of problems, but slightly different sets). In the 15-Puzzle and Logistics domains, however, the counting approximation generally results in a better heuristic: the average numbers of expanded nodes are 655,530 and 483,490 for the 15-Puzzle, and 24,153 and 23,557, for Logistics. Again, variance across repeated runs is greater when using the estimated parameters than when using the counting approximation.

Conclusions

Pattern database heuristics are currently one of the most promising approaches for cost-optimal planning to improve in a manner comparable to what the development of good non-admissible heuristics have done for non-optimal planning. But for this promise to be realised, methods of automatically selecting patterns that yield good PDB heuristics are needed. This is true not only of planning: any application of pattern databases in search can benefit from automatic construction techniques, especially when obtaining the in-depth knowledge of a domain required for manually constructing good PDB heuristics is infeasible.

Adopting the principle of pattern selection as a combinatorial optimisation problem, we proposed one concrete method based on local search in the space of pattern collections. The core of the method is a measure on the improvement of the heuristic quality which is derived from a formula that predicts the search tree size. We showed empirically that our measure results in better PDB heuristics than when the improvement of the heuristic is measured using mean values.

Numerous possibilities for improving the method remain to explore. For example, the search neighbourhood contains only extensions formed by adding one variable to one pattern in the current collection, but there are cases where it is necessary to add more than one variable simultaneously to get any improvement. Since there is an element of randomness in the search, general techniques for improving local search such as restarts can also be expected to be useful.

Acknowledgements

NICTA is funded through the Australian government's *backing Australia's ability* initiative, in part through the Australian research council.

References

Culberson, J., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. 6th European Conference on Planning (ECP'01)*, 13–24.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *Proc. 4th Workshop on Model Checking and Artificial Intelligence (MoChArt'06)*.

Felner, A.; Korf, R.; and Hanan, S. 2004. Additive pattern database heuristics. *Journal of AI Research* 22:279–318.

Haslum, P.; Bonet, B.; and Geffner, H. 2005. New admissible heuristics for domain-independent planning. In *Proc. 20th National Conference on AI (AAAI'05)*, 1163–1168.

Helmert, M. 2006. *Solving Planning Tasks in Theory and Practice*. Ph.D. Dissertation, Albert-Ludwigs-Universität Freiburg.

Holte, R.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170:1123–1136.

Junghanns, A., and Schaeffer, J. 2001. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence* 129(1-2):219–251.

Korf, R., and Felner, A. 2007. Recent progress in heuristic search: A case study of the four-peg Towers of Hanoi problem. In *Proc. 20th International Joint Conference on AI (IJCAI'07)*, 2324–2329.

Korf, R.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129:199–218.

Tomita, E.; Tanaka, A.; and Takahashi, H. 2004. The worst-case time complexity for generating all maximal cliques. In *Computing and Combinatorics (COCOON'04)*, volume 3106 of *LNCS*, 161–170. Springer Verlag.

Wilson, E. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association* 22:209–212.