Korf's Conjecture and the Future of Abstraction-based Heuristics

Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, AB, Canada T6G 2E8
(holte@cs.ualberta.ca)

Abstract

In his 1997 paper on solving Rubik's Cube optimally using IDA* and pattern database heuristics (PDBs), Rich Korf conjectured that there was an inverse relationship between the size of a PDB and the amount of time required for IDA* to solve a problem instance on average. In the current paper, I examine the implications of this relationship, in particular how it limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. I discuss methods that might allow abstraction-based heuristics to scale better than Korf's Conjecture suggests and identify important auxiliary roles for abstraction-based heuristics in heuristic planning and search systems of the future that do not depend on their ability to scale well. Finally, I examine some key assumptions in the analysis underlying Korf's Conjecture, and identify two complications that arise in trying to apply it in practice.

Introduction

In 1997, Rich Korf published a paper (Korf 1997) in which random instances of Rubik's Cube were solved optimally for the first time using a general-purpose search algorithm (IDA*). This outstanding achievement was made possible by abstraction-based heuristics called pattern databases (PDBs), which had only recently been invented (Culberson and Schaeffer 1996). Korf's paper launched the golden age of abstraction-based methods for heuristic search, and a few years later Edelkamp (2001) introduced PDBs to the fledgling world of heuristic-search planning.

Korf's Rubik's Cube paper contained a second important but largely overlooked contribution that casts serious doubt on the long-term future of abstraction-based heuristics. Korf conjectured that there was an inverse relationship between m, the size of a PDB, and t, the number of node expansions that IDA*, using a PDB of size m, would perform to solve a problem instance on average, i.e. $m \cdot t = n$, where n is the size of the brute-force search tree for the original state space. Since node expansion is the primary operation in IDA*, t is indicative of IDA*'s execution time. Korf later gave a rigorous analysis, refining the conjecture to be $M \cdot t = n$, where $M = \frac{m}{1+log(m)}$ (Korf 2007).

In this paper I examine the implications of $M \cdot t = n$, in particular how it limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. Much of the discussion surrounds methods that might allow abstraction-based heuristics to scale better than Korf's Conjecture suggests. There are several promising possibilities, which I believe should be the focus of research in the next few years. I also identify important auxiliary roles for abstraction-based heuristics in heuristic planning and search systems of the future that do not depend on their ability to scale well. Finally, I examine some key assumptions in the analysis underlying Korf's Conjecture, and identify two complications that arise in trying to apply it in practice. This paper supercedes the version published last year (Holte 2013).

Background

Planning and heuristic search study the problem of finding a least-cost path (sequence of actions) from a given start state to a given goal state ($goal$).[1] The distance from state s to state t, denoted $d(s,t)$, is the cost of a least-cost path from s to t. An abstraction of a state space S is a mapping ϕ to another state space, S', such that $d(\phi(s), \phi(t)) \leq d(s,t)$ for all pairs of states $s, t \in S$. For any choice of S' and ϕ, $h(s) = d(\phi(s), \phi(goal))$ is an admissible, consistent heuristic for S. For example, consider the standard 3x3x3 Rubik's Cube, which consists of 8 corner cubies and 12 edge cubies. One way it can be abstracted is to consider all the corner cubies indistinguishable, e.g. to paint all the faces of every corner cubie black. A solution to this abstract problem is a sequence of actions that puts all the edge cubies in their goal positions without regard for what happens to the corner cubies. Distances in this space cannot be greater than the corresponding distances in the original Rubik's Cube space because the same operators exist in both spaces and solutions in the original space also must put the edge cubies in their goal positions. There are a variety of different families of abstraction functions that can be implemented as simple operations on standard state space representations, the most common of which are projection (Edelkamp 2001), and domain abstraction (Holte and Hernádvölgyi 1999).

I will use the term "pattern database" (PDB) to refer to any data structure that is indexed by individual abstract states and stores $d(s', \phi(goal))$ for each $s' \in S'$ from which the abstract goal can be reached. I will assume for the

[1] Korf's analysis assumes there is just one goal state.

present that the PDB is uncompressed, i.e. that the amount of memory needed for the PDB is linear in m, the number of abstract states in S' from which the abstract goal can be reached. Sometimes "PDB" is used in a narrower sense than this, but the reasoning underlying Korf's Conjecture applies to any data structure of this kind. A PDB is computed during a pre-processing phase by enumerating abstract states backwards from the abstract goal and recording the distance to each abstract state reached. During search, $h(s)$, the heuristic value for state $s \in S$, is computed by looking up the distance for $\phi(s)$ in the PDB.

Overview of Korf's Analysis

At the core of the analysis underlying Korf's Conjecture is the distribution of distances in the abstract space, which Korf calls the heuristic distribution since these distances are being used as heuristic values. Table 1 shows the distribution of distances in the abstraction of the 3x3x3 Rubik's Cube that ignores the corner cubies, as described above. In this space all moves cost 1, so the distance to the goal is the number of moves from the goal ("depth").

Given a heuristic distribution, Korf's Conjecture is derived in two steps. In the first step, the heuristic distribution is related to the size of the PDB. For this purpose, Korf assumes the number of abstract states at a given distance from the abstract goal grows exponentially as the distance increases, i.e. that there will be b^d abstract states at distance d for some branching factor b.[2] Korf recognizes that abstract spaces are usually graphs, not trees, so his analysis may underestimate the amount of pruning the heuristic will cause. The key to the analysis being a good approximation is that the exponential growth assumption be true of the majority of heuristic values, as it clearly is in Table 1 (see the "ratio" column).

The second step of the derivation relates the heuristic distribution to the number of nodes expanded by IDA*. In the formal analysis (Korf 2007) this step is done using the KRE

[2]To simplify the analysis, Korf assumes that this b is the same as the branching factor in the search tree for the original space.

depth	#states	ratio
0	1	-
1	18	18.0
2	243	13.5
3	3,240	13.3
4	42,807	13.2
5	555,866	13.0
6	7,070,103	12.7
7	87,801,812	12.4
8	1,050,559,626	12.0
9	11,588,911,021	11.0
10	110,409,721,989	9.5
11	552,734,197,682	5.0
12	304,786,076,626	-
13	330,335,518	-
13	248	-

Table 1: Distance-to-goal ("depth") distribution for the 3x3x3 Rubik's Cube abstraction that ignores the corner cubies (Table 2 in (Korf 2008)). "ratio" is #states at depth d divided by #states at depth $d-1$ (not shown if less than 1).

formula (Korf, Reid, and Edelkamp 2001). The relationship $M \cdot t = n$ follows directly from some straightforward manipulation of the KRE formula with an exponential heuristic distribution.

Because this derivation is entirely formal, Korf's Conjecture is no longer a conjecture, except in the sense that the assumptions on which the analysis is based are being conjectured to hold in problems of interest. That is a topic I will return to later in the paper, but for the next few sections I will assume Korf's Conjecture has been proven and explore its implications.

The Problem of Scaling

The fact that the amount of memory needed to store a PDB is linear in m, the number of abstract states for which the PDB stores information, limits how useful abstraction-based heuristics can be in solving combinatorial problems. n grows exponentially as the size of a combinatorial problem increases (e.g. add 1 more block to the blocks world, 1 more disk to the Towers of Hanoi, 1 more row or column full of tiles to the sliding-tile puzzle). If Korf's Conjecture is true, then $M \cdot t$ must also grow exponentially. If we have a fixed amount of memory, t would have to grow exponentially, and, if we have an upper bound on how much time we are willing to wait for a solution, then m must grow exponentially. This represents a fundamental limitation on the ability of abstraction-based heuristics to guide search effectively.

If we allow m and t to both increase as n increases, it is somewhat encouraging to see that they can increase with the square root of n ($M{=}\sqrt{n} \Rightarrow t{=}\sqrt{n}$). If the time to construct the PDB is linear in M, then the total time to solve a single problem instance will also grow with the square root of n, instead of being linear in n as brute-force search would be. Nevertheless, if n grows exponentially as we increase the size of our combinatorial problems, t and m will both also grow exponentially.

Possible Solutions

In this section I review existing technologies that might provide a solution to the scaling problem.

Disk-based and Distributed PDBs. Storing PDBs on disk instead of in RAM (Sturtevant and Rutherford 2013; Zhou and Hansen 2005) or distributing them across a cluster of workstations that each have their own RAM (Edelkamp, Jabbar, and Kissmann 2008) allows m to be two or three orders of magnitude larger than it could be if the PDB had to fit in the RAM of one workstation. This is extremely useful, but, in the end, is just a big constant factor, not a solution to the scaling problem.

PDB Compression. Lossless compression of PDBs, such as symbolic PDBs (Edelkamp 2001), routinely reduces the memory needed to store a PDB by one to two orders of magnitude (Ball and Holte 2008). In certain special cases symbolic PDBs have been proven to be logarithmic in the uncompressed size of the PDB (Edelkamp and Kissmann 2011). The scaling problem is solved by symbolic PDBs in these spaces, but not in general. Large memory reductions have also been obtained with lossy PDB compression methods (Felner et al. 2007; Samadi et al. 2008), which offer a little more hope for addressing the scaling issue because it seems that by allowing some loss of heuristic information, these methods sometimes produce a more favourable tradeoff between time and memory than is predicted by Korf's Conjecture. For example, Felner et al. (2007)'s Table 9 reports that a 9 times reduction in memory results in only 2.25 greater search time, and an even more favourable tradeoff is reported in their Table 4 for the Towers of Hanoi.

Hierarchical Heuristic Search. Instead of precomputing, and storing, the entire PDB, hierarchical heuristic search (Holte, Grajkowski, and Tanner 2005; Leighton, Ruml, and Holte 2011) computes, on demand, precisely those abstract distances that are required as heuristic values in solving a given problem instance. Experimentally, only about 1% of the PDB entries actually need to be computed to solve a problem instance (see Table 3 in (Holte, Grajkowski, and Tanner 2005)), which means about one order of magnitude less memory is required than for the full PDB since the data structure for caching distance-to-goal information in hierarchical heuristic search is not as compact as a good PDB data structure. What is not known is how the memory requirements of hierarchical heuristic search scale as the state space size increases. It is possible that hierarchical heuristic search scales better than PDBs and therefore provides at least a partial solution to the scaling problem.

Multiple PDBs. One direction that offers clear hope for mitigating the scaling problem is the use of multiple PDBs. Korf's Conjecture, as I have described it here, is about how search time is related to the size of a PDB when just one PDB is used to guide IDA*. But it is known that, for a fixed amount of memory, search using one PDB that uses all the memory is much slower than search that takes the maximum of two PDBs that each require half the memory (Holte et al. 2004; 2006). Could it be that as a search space scales up, the total size (m) of a set of PDBs does not have to grow exponentially in order to keep t constant?

In my opinion, the answer is almost certainly "no" when the maximum is taken over the set of PDBs. Korf (2007) analyzes this case when the PDBs are "independent". According to the formula he derives with this assumption, the optimal number of PDBs is two, which is not consistent with experimental data. Korf offers two reasons for this discrepancy. One is that the heuristic distribution is not necessarily exponential. I return to this point below. The other is that the derived formula overestimates the pruning power of a set of PDBs if they are not truly independent. I suspect that non-additive PDBs will rarely be independent, or even approximately independent, and so their total size will have to scale almost as quickly as a single PDB.

On the other hand, a set of PDBs based on additive abstractions (Yang et al. 2008) might well give us the scaling behaviour we want. Breyer and Korf (2010) proved that the speedup produced using a set of additive heuristics is the product of their individual speedups over brute-force search if the heuristics are independent in the same sense as above. This seems to me a much more plausible assumption for additive abstractions than non-additive ones. Breyer and Korf's analysis suggests that as a state space is scaled up, t could be kept constant by increasing the number of PDBs, which is only a linear increase in memory. A more powerful version of this idea, which I call "factored heuristics", is discussed below.

Multiple PDB Lookups. Another source of hope related to the use of multiple PDBs is the use of multiple lookups in a single PDB: to compute the heuristic value of state s one not only does the normal PDB lookup, but has a mechanism for extracting one or more additional values from the PDB that are also lower bounds on s's distance to goal. This obtains the benefits of having multiple PDBs while using the memory needed by only one PDB. Most studies that make multiple lookups in a PDB use symmetries in the search space to define the additional lookups (Zahavi et al. 2008).

A particularly powerful form of symmetry is seen in the Towers of Hanoi. The standard method of abstracting this space is to choose k disks and build a PDB containing the number of moves needed to get those k disks to their goal positions, entirely ignoring all the other disks (Felner et al. 2007). If the Towers of Hanoi problem one is trying to solve has $2k$ disks, two lookups can be made in this PDB and their values added: one lookup is based on any set of k disks, the other lookup is based on the other k disks. If the problem is made bigger by increasing the number of disks, the PDB remains the same, but more lookups are done and added together to compute a state's heuristic value. I call this a "factored heuristic" because the heuristic calculation is done by decomposing the state into parts (in this example, partitioning the set of disks into groups that each contain k or fewer disks), making separate lookups for each part in the same PDB, and then adding the results.

The same idea has been used for the Pancake puzzle (Torralba Arias de Reyna and Linares López 2011) and produced extremely good (sub-exponential) scaling behaviour. Doubling the number of pancakes from 20 to 40 increased the number of nodes generated by a factor of 120, a miniscule fraction of the increase in the size of the state space (from 20! to 40!).

Reliance on symmetries in the state space has limited the use of multiple PDB lookups. However, Pang and Holte (2012) report a technique that allows multiple PDB lookups to be based on multiple abstractions that all map to the same abstract space, thereby allowing multiple lookups to be done for state spaces that do not have symmetries.

Alternative Ways of Representing Abstraction-based Heuristics. It may be possible to avoid the scaling problem by representing abstraction-based heuristics in a form that is entirely different than a lookup table. Manhattan Distance, for example, could be implemented as an additive PDB, but is more commonly implemented as a procedure that is executed for a given state. In fact, this procedure strongly resembles hierarchical heuristic search, but with individual abstract spaces that are so small there is no need for a hierarchy of abstractions or for search results to be cached.

An alternative representation of abstraction-based heuristics that is especially useful if admissibility is not required is to use machine learning to create an extremely compact approximation of a PDB (e.g. a small neural network), as was done by Samadi et al. (2008). If admissibility is required, a lookup table can be used to store the PDB entries that the neural network overestimates. In the experiments by Samadi et al., only about 2% of the PDB entries needed to be explicitly stored.

A different way of compactly approximating an abstraction-based heuristic is to map the abstract states into a low-dimensional Euclidean space in such a way that Euclidean distances between states are admissible and consistent. The basic technology for this exists (Rayner, Bowling, and Sturtevant 2011), but at present it requires the same amount of memory as a PDB.

Alternative Roles for Abstraction-based Heuristics

In this section I consider what role there might be for abstraction-based heuristics if, indeed, they are unable to scale to provide effective guidance for solving much larger problems than we currently study. Assuming they do not scale, we need to consider roles in which weak heuristics can make important contributions to solving search problems.

Node Ranking. Many search algorithms, including depth-first branch and bound, beam search, greedy best-first search (Doran and Michie 1966), and limited discrepancy search (Harvey and Ginsberg 1995), sort the nodes they generate according to what might be called a ranking function. Although there is no need for the ranking function to be an estimate of the cost to reach the goal (e.g. (Xu, Fern, and Yoon 2009)), a heuristic function is an obvious candidate to use for ranking. Even a relatively weak heuristic can be effective for ranking. For example, BULB (Furcy and Koenig 2005), which combines beam search and limited discrepancy backtracking, solves instances of the 9×9 sliding-tile puzzle in 120 seconds using Manhattan Distance for ranking, which is not an especially effective heuristic for that size of puzzle.[3]

Type Systems. Levi Lelis and his colleagues have developed methods for a variety of search-related tasks that require the nodes in a search tree to be partitioned into a set of "types" (Lelis et al. 2012; Lelis, Zilles, and Holte 2013a; 2013b; Lelis, Otten, and Dechter 2013; Xie et al. 2014). In the ideal partitioning, the search subtrees below nodes of the same type are identical in certain key regards, such as the size of the subtree, the cost of the cheapest solution in the subtree, etc. Type systems based on heuristic functions have proven very effective in all these studies, and, as with node ranking, there is no need for the heuristic to be especially accurate (many of Lelis's heuristics are small PDBs).

[3]In 2001 it was estimated that IDA* would need 50,000 years of CPU time to solve one instance of the 5×5 sliding-tile puzzle using Manhattan Distance (Korf, Reid, and Edelkamp 2001). Korf has re-calculated this based on the speed of today's computers and estimates that only 7,600 years would be required now (personal communication).

Features for Learning. There has been recent interest in using machine learning to create heuristic functions (Samadi, Felner, and Schaeffer 2008; Jabbari Arfaee, Zilles, and Holte 2011). In these studies small PDBs have proven to be excellent features for machine learning. Again, there is no obvious need for heuristics used for this purpose to be especially accurate.

Heuristic Search as a Component Technology. One approach to solving a very large problem is to decompose it into a sequence of subproblems that are then solved one by one, in order, to produce a solution to the original problem. Korf developed this technique for solving Rubik's Cube, with the subproblems being solved by brute-force search (Korf 1985). Later, Hernádvölgyi (2001) realized that abstraction-based heuristics could be used to guide the solving of the subproblems. This speeded up search so much that it allowed several subproblems to be merged to form a single subproblem that was still feasible to solve, resulting in much shorter solutions (50.3 moves, on average, compared to 82.7). This is just one example of how abstraction-based heuristic search can be used as a component in a different type of search system.

The Assumptions Underlying Korf's Analysis

The previous sections have outlined directions for the future of abstraction-based heuristics assuming that Korf's Conjecture holds. In this section, I examine the assumptions underlying Korf's analysis, and describe two complications that arise in trying to apply it in practice. The two points I will make in this section will both be illustrated using the 5-disk, 3-peg Towers of Hanoi puzzle, with states being represented the same way that Zilles and Holte (2010) represented states in the Blocks World with distinct table positions. In this representation there are 6 state variables for each peg. The first variable is an integer $(0 \dots 5)$ saying how many disks are on the peg. The other 5 variables give the names of the disks $(d_1 \dots d_5$ or $nodisk)$ in the order they occur on the peg from bottom to top: the first of these variables names the disk at the bottom ($nodisk$ if the peg has no disks on it), the second names the disk second from the bottom ($nodisk$ if the peg has 0 or 1 disks on it), and so on. The abstraction of this space that I will discuss is shown in Figure 1. This abstraction keeps the variables that say how many disks are on each peg and deletes all the other variables, so that the identities of the disks on each peg are entirely lost. For example, the Towers of Hanoi goal state, which has all the disks on peg_1, would be mapped to abstract state **500** (at the top of the figure), and any Towers of Hanoi state in which there are 3 disks on peg_1, 2 disks on peg_2, and no disks on peg_3 would be mapped to abstract state **320**.

The first thing one notices about this space is that the distance distribution is not exponential, it is linear; there are exactly $d + 1$ abstract states at distance d from the abstract goal. It would seem therefore that Korf's Conjecture does not apply in this case. However, Korf's analysis could be

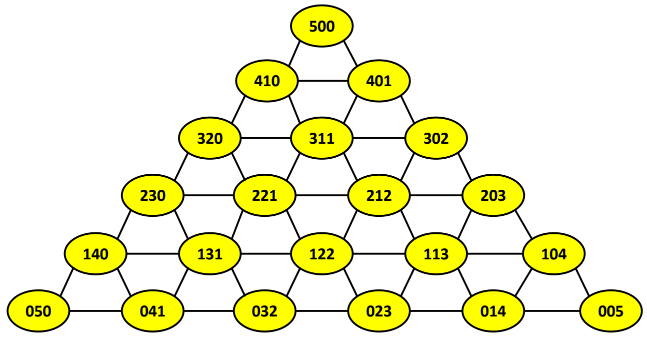

Figure 1: Abstraction of the 5-disk, 3-peg Towers of Hanoi state space that keeps track of how many disks are on each peg but not their identities. **320**, for example, is the abstract state in which there are 3 disks on peg_1, 2 disks on peg_2, and no disks on peg_3.

repeated with a linear model plugged into the KRE formula to derive the relation between t and m for this situation. I have not actually done that, but using the simpler, "back of the envelope" method of Korf's original, intuitive analysis, I estimate that, with a heuristic defined by an abstract space with a linear distance distribution, if n is doubled, m does not need to be doubled to keep t constant, it only needs to be increased additively by approximately $2\sqrt{m}$.[4]

In the preceding paragraph I said "it would seem" Korf's Conjecture does not apply because it is not correct, in general, to equate the distance distribution in the abstract space with the heuristic distribution. In the KRE formula the heuristic distribution is the fraction of states in the original state space that have a particular heuristic value,[5] not the fraction in the abstract space. The distance distribution in the abstract space and the heuristic distribution are only the same if the same number of states in the original space map to each abstract state. That is certainly not the case for the Towers of Hanoi abstraction in Figure 1. Only one Towers of Hanoi state maps to abstract state **500**, but 10 map to abstract state **320**, and 30 map to abstract state **221**. The number of Towers of Hanoi states that map to each abstract distance are 1 (distance 0), 10 (distance 1), 40, 80, 80, and 32 (distance 5), which is probably close enough to being an exponential distribution that Korf's Conjecture would hold in this case. This is an important consideration because it oftens happens naturally that different numbers of states that map to different abstract states, as in this example, and, in addition, there are several abstraction methods that are very likely to produce abstractions in which this occurs (e.g. CEGAR (Seipp and Helmert 2013), CFDP (Raphael 2001), and merge-and-shrink (Nissim, Hoffmann, and Helmert 2011)).

An important, but easily overlooked, feature of the numbers (1, 10, 40, etc.) just given is that they refer to the number of **reachable** states that map to each abstract state. If we consider unreachable states as well, then the same number of states map to each abstract state and we are back to a linear heuristic distribution. So, Korf's Conjecture rests on the heuristic distribution of reachable states being distributed exponentially. That is the first point about the conjecture I wish to make. This does not make the analysis any less valid, it just means it is more difficult to apply than doing a simple inspection of the abstract space, and that similar problems might have very different behaviours: removing the Towers of Hanoi restriction that a larger disk cannot be put onto a smaller one changes the heuristic distribution from being (approximately) exponential to being linear.

The other point I wish to make is that although the parameter m in the analysis seems perfectly well defined, it is, in fact, anything but. The most obvious source of doubt about how exactly m should be defined is spurious abstract states, which are abstract states that can be reached by backwards search from the abstract goal but whose pre-image[6] cannot be reached by forward search from the start state in the original space.[7] Because they are reached by backwards search from the abstract goal, there is an entry in the PDB for each spurious state, but because their pre-images cannot be reached during forward search from the start state in the original state space, they contribute nothing to the pruning power of the heuristic and therefore should not be included in the heuristic distribution or the value of m used in Korf's analysis (if m is increased by adding spurious states, t will obviously not change at all). For example, Zilles and Holte (2010) report an abstraction of the Blocks World in which there are $1,310,720$ abstract states from which the abstract goal can be reached, of which only $89,400$ are non-spurious. In Korf's analysis, $m = 89,400$ must be used, not $m = 1,310,720$. Unfortunately, there is no practical way, in general, of determining how many abstract states are spurious, or even deciding whether or not there are any spurious states (Zilles and Holte 2010).

The difficulties concerning m's definition raised by spurious states is a special case of a more general phenomenon where two abstract spaces of different sizes give rise to exactly the same heuristic values for all states in the original space. The space in Figure 1 has 21 states. Once **500** has been designated as the goal, the space can be compressed to just 6 states without losing any distance-to-goal information. This is done by mapping all the states on the same level in the figure to the leftmost state on the level. This is an ordinary abstraction of the space in Figure 1 defined by keeping the first state variable and deleting the other two. We now have a real dilemma in deciding what value of m to

[4]This calculation assumes that $t = 2^{D-\overline{h}}$, where D is the depth of the search in the original space, 2 is the branching factor in both the original space and the abstract space, and $\overline{h}$ is the average heuristic value, which, for a space with a linear distance distribution, is approximately $\sqrt{m}$.

[5]Even this is not a perfectly correct statement. The distribution described here is what Korf calls the overall distribution. The KRE analysis requires the "equilibrium" distribution, which in general is not the same as the overall distribution.

[6]If S is a state space and ϕ is an abstraction mapping S to S', the pre-image of $s' \in S'$ is $\{s \in S | \phi(s) = s'\}$.

[7]Spurious states can be caused either by the backwards (regression) search (Bonet and Geffner 2001) that computes the PDB or by the abstraction process itself (Zilles and Holte 2010).

use. There are no spurious states in Figure 1, so $m = 21$ is genuinely the size of the space, but the heuristic based on it is identical to a heuristic based on a linear space of size 6. If n is doubled and we want t to remain constant, doubling "m" means to make a space equivalent to a linear space of size 12, not a triangular space of size 42.[8] For the purpose of defining a heuristic, additional states at each level are as useless as spurious states: increasing m by adding more of them does not affect t at all.

Methods for creating symbolic PDBs (Edelkamp 2001) can be seen as doing exactly what I have just illustrated with the space in Figure 1: they take the distances-to-goal defined in one abstract space (the space in Figure 1, for example), and construct a smaller abstract space that returns exactly the same heuristic value for every state in the original space (the linear space with just 6 nodes, for example). This is also what merge-and-shrink aspires to do with its shrinking stategies that are h-preserving (Helmert, Haslum, and Hoffmann 2007) or based on bisimulation (Nissim, Hoffmann, and Helmert 2011). In fact, these methods can be seen as always producing a linear abstract space with one state at each distance, coupled with a memory-based indexing mechanism for mapping a given state to one of these abstract states.

The fact that abstract spaces of very different sizes can produce identical heuristics suggests that Korf's goal of relating the number of node expansions by IDA* to the number of states in an abstract space is simply impossible. And yet there is experimental data showing that a strong relation does indeed hold between PDB size and the number of node expansions, exactly as predicted by Korf's Conjecture. The data in Table 2 is based on the results reported in Korf (2007)'s Table 2, columns "Size" (m) and "IDA*" (t). One iteration of IDA* with a depth bound of 12 was run on the same 1000 Rubik's Cube instances using four PDBs of different sizes based on abstractions called "6 corners", "6 edges", "8 corners", and "7 edges". For a given PDB, t is the average number of nodes expanded by IDA* on these runs. The asymptotic branching factor of Rubik's Cube, 13.34847 (Korf 1997), was used as the base of the logarithm in computing M from m. If Korf's Conjecture was perfectly correct, the numbers in the $M \cdot t$ column would be identical. They are nearly so: the largest value is only about 13% larger than the smallest value. This shows that, in this particular experiment, Korf's Conjecture makes very accurate predictions about how the number of node expansions will change if the PDB size is changed. Additional support for Korf's Conjecture is given in an experiment that looked at a large number of PDBs of many different sizes for three state spaces (Holte and Hernádvölgyi 1999), although this study uses A*, not IDA*.

[8]The two are not the same; a triangular space would need 78 states to correspond to a linear space of length 12.

PDB	m	t	$M \cdot t$
6 corners	14,696,640	20,918,500	41,722 $\times 10^9$
6 edges	42,577,920	8,079,408	44,222 $\times 10^9$
8 corners	88,179,840	3,724,861	40,752 $\times 10^9$
7 edges	510,935,040	670,231	39,191 $\times 10^9$

Table 2: Evidence supporting Korf's Conjecture. $M \cdot t$ is almost constant for four PDBs of different sizes for Rubik's Cube. m and t are from Table 2 in (Korf 2007).

Conclusions

In this paper, I have examined how Korf's Conjecture ($M \cdot t = n$), if it is true, limits the ability of abstraction-based heuristic methods, such as PDBs, to scale to larger problems. It is certain that abstraction-based heuristics can and should play important auxiliary roles in the heuristic planning and search systems of the future, whether or not they scale well. If they do not scale well, there are several alternative types of heuristics in the planning literature—delete relaxations (McDermott 1996; Bonet, Loerincs, and Geffner 1997), heuristics based on linear programming (Bonet 2013), the causal graph heuristic (Helmert 2004), landmark-based heuristics (Bonet and Helmert 2010), and h^m (Haslum 2006)—whose scaling behaviour has only partly been studied (Helmert and Mattmüller 2008).

However, there are several reasons to expect that abstraction-based heuristics may continue to play a primary role in planning and search systems. In part, this hope lies in the use of merge-and-shrink, multiple additive abstractions, and multiple heuristic lookups in a single abstract space. These technologies exist, and might possibly greatly mitigate the scaling problem, but more research is needed on them. The second place where hope lies is in the scaling behaviour of compression methods and hierarchical heuristic search, which I assumed to be linear in m in this paper. If any of these were to scale logarithmically, say, instead of linearly, abstraction-based heuristic methods would continue to play a central role in solving large combinatorial problems.

This paper has also shown that it is not straightforward to apply Korf's Conjecture in practice. One reason is that the poor scaling behaviour rests on an assumption about the heuristic distribution of reachable states, not just on the distribution of distances in the abstract space. The other reason is that the number of entries in a PDB is definitely not the sole determining factor of the pruning power of the resulting heuristic: PDBs of very different sizes can give rise to exactly the same heuristic. More research is needed to better understand, and exploit, this phenomenon.

Acknowledgements

Thanks to Malte Helmert, Rich Korf, Carlos Linares López, Rong Zhou, Roni Stern, and Ariel Felner for their feedback, and to Canada's NSERC and Switzerland's SNF for their financial support.

References

Ball, M., and Holte, R. C. 2008. The compression power of symbolic pattern databases. In *ICAPS*, 2–11.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Bonet, B., and Helmert, M. 2010. Strengthening landmark heuristics via hitting sets. In *ECAI*, 329–334.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *AAAI*, 714–719.

Bonet, B. 2013. An admissible heuristic for SAS+ planning obtained from the state equation. In *IJCAI*.

Breyer, T. M., and Korf, R. E. 2010. Independent additive heuristics reduce search multiplicatively. In *AAAI*, 33–38.

Culberson, J., and Schaeffer, J. 1996. Searching with pattern databases. In *Proc. 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, volume 1081 of *Lecture Notes in Computer Science*, 402–416. Springer.

Doran, J. E., and Michie, D. 1966. Experiments with the Graph Traverser program. In *Proceedings of the Royal Society*, 235–259.

Edelkamp, S., and Kissmann, P. 2011. On the complexity of BDDs for state space search: A case study in Connect Four. In *AAAI*, 18–23.

Edelkamp, S.; Jabbar, S.; and Kissmann, P. 2008. Scaling search with pattern databases. In *MoChArt*, 49–64.

Edelkamp, S. 2001. Planning with pattern databases. In *Proc. European Conference on Planning*, 13–24.

Felner, A.; Korf, R. E.; Meshulam, R.; and Holte, R. C. 2007. Compressed pattern databases. *Journal of Artificial Intelligence Research* 30:213–247.

Furcy, D., and Koenig, S. 2005. Limited discrepancy beam search. In *IJCAI*, 125–131.

Harvey, W. D., and Ginsberg, M. L. 1995. Limited discrepancy search. In *IJCAI*, 607–615.

Haslum, P. 2006. Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *Journal of Artificial Intelligence Research (JAIR)* 25:233–267.

Helmert, M., and Mattmüller, R. 2008. Accuracy of admissible heuristic functions in selected planning domains. In *AAAI*, 938–943.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

Hernádvölgyi, I., and Holte, R. C. 2000. Experiments with automatically created memory-based heuristics. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, volume 1864 of *Lecture Notes in Artificial Intelligence*, 281–290. Springer.

Hernádvölgyi, I. T. 2001. Searching for macro operators with automatically generated heuristics. In *Canadian Conference on AI*, 194–203.

Holte, R. C., and Hernádvölgyi, I. T. 1999. A space-time tradeoff for memory-based heuristics. In *AAAI*, 704–709.

Holte, R. C.; Newton, J.; Felner, A.; Meshulam, R.; and Furcy, D. 2004. Multiple pattern databases. In *ICAPS*, 122–131.

Holte, R. C.; Felner, A.; Newton, J.; Meshulam, R.; and Furcy, D. 2006. Maximizing over multiple pattern databases speeds up heuristic search. *Artificial Intelligence* 170(16-17):1123–1136.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 121–133.

Holte, R. C. 2013. Korf's conjecture and the future of abstraction-based heuristics. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*.

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Korf, R. E.; Reid, M.; and Edelkamp, S. 2001. Time complexity of iterative-deepening-A*. *Artificial Intelligence* 129(1-2):199–218.

Korf, R. E. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Korf, R. 1997. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI*, 700–705.

Korf, R. E. 2007. Analyzing the performance of pattern database heuristics. In *AAAI*, 1164–1170.

Korf, R. E. 2008. Minimizing disk i/o in two-bit breadth-first search. In *AAAI*, 317–324.

Leighton, M. J.; Ruml, W.; and Holte, R. C. 2011. Faster optimal and suboptimal hierarchical search. In *Symposium on Combinatorial Search (SoCS)*.

Lelis, L.; Stern, R.; Felner, A.; Zilles, S.; and Holte, R. C. 2012. Predicting optimal solution cost with bidirectional stratified sampling. In *ICAPS*.

Lelis, L. H. S.; Otten, L.; and Dechter, R. 2013. Predicting the size of depth-first branch and bound search trees. In *IJCAI*.

Lelis, L.; Zilles, S.; and Holte, R. C. 2013a. Stratified tree search: A novel suboptimal heuristic search algorithm. In *Twelfth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 555–562.

Lelis, L. H. S.; Zilles, S.; and Holte, R. C. 2013b. Predicting the size of IDA*'s search tree. *Artificial Intelligence* 196:53–76.

McDermott, D. V. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS)*, 142–149.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *IJCAI*, 1983–1990.

Pang, B., and Holte, R. C. 2012. Multimapping abstractions and hierarchical heuristic search. In *Symposium on Combinatorial Search (SoCS)*.

Raphael, C. 2001. Coarse-to-fine dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23(12):1379–1390.

Rayner, D. C.; Bowling, M. H.; and Sturtevant, N. R. 2011. Euclidean heuristic optimization. In *AAAI*, 81–86.

Samadi, M.; Siabani, M.; Felner, A.; and Holte, R. 2008. Compressing pattern databases with learning. In *ECAI*, 495–499.

Samadi, M.; Felner, A.; and Schaeffer, J. 2008. Learning from multiple heuristics. In *AAAI*, 357–362.

Seipp, J., and Helmert, M. 2013. Counterexample-guided Cartesian abstraction refinement. In *ICAPS*.

Sturtevant, N. R., and Rutherford, M. J. 2013. Minimizing writes in parallel external memory search. In *IJCAI*.

Torralba Arias de Reyna, Á., and Linares López, C. 2011. Size-independent additive pattern databases for the pancake problem. In *Symposium on Combinatorial Search (SoCS)*, 164–171.

Xie, F.; Müller, M.; Holte, R.; and Imai, T. 2014. Type-based exploration with multiple search queues for satisficing planning. In *AAAI*.

Xu, Y.; Fern, A.; and Yoon, S. W. 2009. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research* 10:1571–1610.

Yang, F.; Culberson, J.; Holte, R.; Zahavi, U.; and Felner, A. 2008. A general theory of additive state space abstractions. *Journal of Artificial Intelligence Research* 32:631–662.

Zahavi, U.; Felner, A.; Holte, R. C.; and Schaeffer, J. 2008. Duality in permutation state spaces and the dual search algorithm. *Artificial Intelligence* 172(4-5):514–540.

Zhou, R., and Hansen, E. A. 2005. External-memory pattern databases using structured duplicate detection. In *AAAI*, 1398–1405.

Zilles, S., and Holte, R. C. 2010. The computational complexity of avoiding spurious states in state space abstraction. *Artificial Intelligence* 174:1072–1092.