Multimapping Abstractions and Hierarchical Heuristic Search

Bo Pang
Computing Science Department
University of Alberta
Edmonton, AB Canada T6G 2E8
(bpang@ualberta.ca)

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

Abstract

In this paper we introduce a broadly applicable method, called multimapping abstraction, that allows multiple heuristic values for a state to be extracted from one abstract state space. The key idea is to define an abstraction to be a multimapping, i.e., a function that maps a state in the original state space to a set of states in the abstract space. We performed a large-scale experiment on several benchmark state spaces to compare the memory requirements and runtime of Hierarchical IDA* (HIDA*) using multimapping domain abstractions to HIDA* with individual domain abstractions and to HIDA* with multiple, independent domain abstractions. Our results show that multimapping domain abstractions are superior to both alternatives in terms of both memory usage and runtime.

Introduction

An abstraction ϕ of a state space S is a mapping of the states of S to the states of another state space T (the abstract space) such that the distance between any pair of states in S is greater than or equal to the distance between the corresponding states in T, i.e., $d(s_1, s_2) \geq d(\phi(s_1), \phi(s_2))$, where $d(x, y)$ is the cost of a least-cost path from state x to state y. $h(s) = d(\phi(s), \phi(goal))$ is therefore an admissible heuristic for searching in S.[1]

Techniques have been developed that allow several heuristic lookups to be done for a state using a single abstraction ϕ. These techniques use special properties of the state space to define "symmetry" mappings sym_i: $S \rightarrow S$ such that $d(sym_i(s), goal)$, the cost to reach the goal state from $sym_i(s)$, is guaranteed to be the same as $d(s, goal)$. With this guarantee, $h(sym_i(s))$ never overestimates $d(s, goal)$ and the heuristic $h_{max}(s) = \max(h(s), \max_i\{h(sym_i(s))\})$ is admissible. Commonly used sym functions are the geometric symmetries in puzzles such as the 15-puzzle, Rubik's Cube, and TopSpin, and the "dual state" function in permutation state spaces (Zahavi et al. 2008). General symmetry mappings, based on automorphisms (Domshlak, Katz, and Shleyfman 2012), could also be used for this purpose. Substantial speedup can result from taking the maximum over multiple heuristic lookups in a single abstract space (Zahavi et al. 2008).

The main drawback of these techniques is that they apply only to state spaces that contain symmetries. The main contribution of the present paper is a general technique, called "multimapping abstraction", allowing multiple heuristic lookups in the same abstract space to be defined for a broad range of state spaces including ones that do not contain symmetries. The key idea is to define an abstraction to be a multimapping, i.e., a function that maps a state in the original space to a set of states in the abstract space. This technique was briefly introduced by us last year (Pang and Holte 2011) but here we give a thorough discussion and evaluation.

The use of multiple abstractions is well-known and the heuristic $h(s) = \max_i\{d(\phi_i(s), \phi_i(goal))\}$ has been shown to be highly beneficial (Holte et al. 2006) when the abstract distances in each abstract space are pre-computed to create a pattern database (PDB) (Culberson and Schaeffer 1996). In hierarchical heuristic search, where abstract distances are computed during search on an as-needed basis, the cost of computing multiple abstract distances can outweigh the benefits of having an improved heuristic value (Holte, Grajkowski, and Tanner 2005). What differentiates multimapping abstractions from the normal use of multiple abstractions is that in the latter each abstract space is entirely distinct ($\phi_i : S \rightarrow T_i$ with $T_i \neq T_j$ if $i \neq j$) whereas in our method the abstractions of a state are all in the same abstract space ($\phi(s) \subseteq T$).

The difference between a multimapping and the use of multiple, independent abstractions is illustated with the 5-pancake puzzle in Figure 1. We wish to estimate the distance from the state $s = \langle 3, 4, 2, 5, 1\rangle$ (at the bottom of the figure) to the goal state $g = \langle 1, 2, 3, 4, 5\rangle$ (at the top of the figure). The two abstractions, ϕ_1 and ϕ_2, both map to the abstract space in which states contain three 1s and two 2s; all such states are reachable from one another using the pancake puzzle operators. If treated as multiple independent abstractions, $h(s)$ would be computed using each abstraction separately (the dotted lines in the figure) and the maximum taken (4 in this example). If treated as a multimapping, we would have $\phi(s) = \{\phi_1(s), \phi_2(s)\}$ and the heuristic func-

[1]The paper is written assuming there is just one goal state, but multimapping abstractions apply equally well when there is a goal condition, as long as the condition can be abstracted properly.

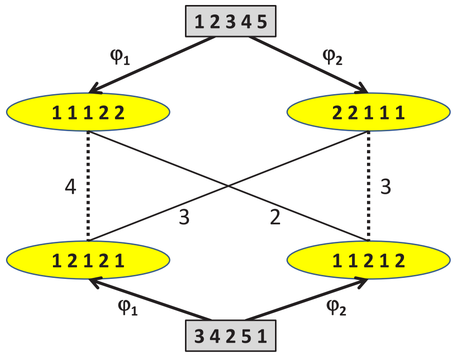


Figure 1: Pancake puzzle with two domain abstractions. ϕ_1 maps constants 1, 2, and 3 to 1, and maps 4 and 5 to 2. ϕ_2 maps 3, 4, and 5 to 1, and maps 1 and 2 to 2. Rectangles are states in the original space, ovals are abstract states. Dotted lines are distances between states based on the same abstraction, thin solid lines are distances between states based on different abstractions.

tion would be defined as

$$h(s) = \max_{s' \in \phi(s)} \min_{g' \in \phi(g)} d(s', g')$$

Hence, all four distances between the abstract states in the figure would be taken into account (the solid diagonal lines as well as the dotted lines), producing the value 3, the maximum of $\min(4,3)$ and $\min(2,3)$.

Given the same set of abstractions, multimapping cannot produce a heuristic value larger than would be produced by using them independently. The potential advantage of multimapping is computational. If the abstract states generated by different abstractions are reachable from each other, the memory and time needed for computing the heuristic could be substantially smaller for multimapping than for multiple abstractions. For example, if the heuristic is stored as a PDB, multimapping would build just one PDB whereas the multiple abstraction approach would build and store multiple copies of the PDB. Similar memory and time savings are expected in the hierarchical heuristic search setting (Holte, Grajkowski, and Tanner 2005), because all the abstractions would share the same cache, as opposed to each having its own cache.

Compared to using a single abstraction mapping, multimapping has the advantage that in computing $h(s)$ it takes the maximum abstract distance to goal over all the states in $\phi(s)$. This advantage might be offset by having a set of abstract goal states over which distance is minimized. In this paper we present three ways of reducing the negative effect due to this minimization. An additional disadvantage of multimapping compared to a single abstraction mapping is that computing the heuristic is slower with multimapping because it computes more than one abstraction of a state and abstract distance to goal.

We chose to evaluate multimapping abstraction in the hierarchical heuristic search setting. Our experiments show that Hierarchical IDA* with multimapping abstractions solves problems using less time and less memory than it does with a single abstraction or with multiple, independent abstractions.

Multimapping Abstractions

A multimapping from S to T is a function, ϕ, that maps each $s \in S$ to a subset of T. If S and T are state spaces, multimapping ϕ is an *abstraction* if, for every $s \in S$, $\phi(s)$ is not the empty set and the following holds for all pairs of states $s, g \in S$:

$$\forall s' \in \phi(s) : \min_{g' \in \phi(g)} d(s', g') \leq d(s, g).$$

This definition of abstraction guarantees the admissibility of $h(s)$ for multimappings, as defined in the Introduction. Note that we do not require every abstract goal state to be reachable from every abstract state: admissibility is still guaranteed even if $d(s', g') = \infty$ for some $s' \in \phi(s)$ and $g' \in \phi(g)$. If exactly one $g' \in \phi(g)$ is reachable from each $s' \in \phi(s)$, multimapping abstraction is equivalent to using multiple, independent abstractions. When several, or all, of the g' are reachable from each s' the two methods are different.

Lemma 1 *If ϕ is a multimapping abstraction of state space S then the heuristic h based on ϕ is consistent.*

Proof. Let s and t be any two states in S and g the goal state in S. We need to prove that $h(s) \leq d(s,t) + h(t)$. The key ideas behind the proof are illustrated in Figure 2. Let $s' \in \phi(s)$ and $g'_s \in \phi(g)$ be such that $h(s) = d(s', g'_s)$ (i.e., these are the abstract states that define the max and the min, respectively, in the calculation of $h(s)$). Let $t' \in \phi(t)$ be such that $d(s', t') \leq d(s, t)$. The existence of t' is guaranteed by how we defined multimapping abstractions. Finally let $g'_t \in argmin_{g' \in \phi(g)} d(t', g')$. From the definition of g'_s we have:

$$h(s) = d(s', g'_s) \leq d(s', g'_t)$$

By the triangle inequality we get:

$$\leq d(s', t') + d(t', g'_t)$$

From the definition of t' we get:

$$\leq d(s, t) + d(t', g'_t)$$

Finally, from the definition of $h(t)$ we get:

$$\leq d(s, t) + h(t)$$ □

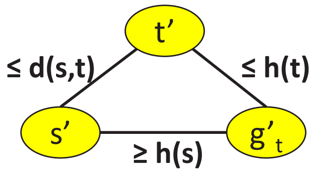


Figure 2: Key ideas in the proof that $h(s)$ is consistent. t' and g'_t are chosen so that the inequalities shown hold. $h(s) \leq d(s,t) + h(t)$ follows immediately from these inequalities and the triangle inequality.

The definition of $h(s)$ for multimappings is inherently asymmetric, even if distances in S are symmetric ($d(s,g) = d(g,s)$ for all $s, g \in S$). For example, suppose ϕ maps s to two abstract states, s'_1 and s'_2, and maps g to just one abstract state, g'. If g is the goal, then the heuristic estimate of the distance between s and g will be $\max(d(s'_1, g'), d(s'_2, g'))$, but if s is the goal then the heuristic estimate of the same distance will be $\min(d(s'_1, g'), d(s'_2, g'))$. A similar phenomenon occurs with dual lookups: if s^d is the dual of state s and g is the goal, we are guaranteed $d(s^d, g) = d(s, g)$ and yet it frequently happens that $h(s) \neq h(s^d)$ (Zahavi et al. 2008). This can be exploited by computing both heuristic values and taking the maximum but we did not do this in our experiments.

Ways to Define Multimapping Abstractions

One way to define a multimapping abstraction ϕ is to use an ordinary abstraction ψ together with a set of symmetry functions (as defined in the Introduction), $sym_1, ..., sym_{n-1}$, and define $\phi(s) = \{\psi(s), \psi(sym_1(s)), ..., \psi(sym_{n-1}(s))\}$. n is called ϕ's *mapping factor*. This is what has been described in the past as multiple lookups in a single PDB.

A second way to create a multimapping is to use the state-set abstraction method we presented last year (Pang and Holte 2011). This method applies to state spaces in which there are multiple occurrences of the same constant in the representation of a state. The idea is to replace some of the occurrences of each constant by a special kind of symbol (see last year's paper for details). Because there are different ways of choosing which occurrences are replaced, this naturally produces a multimapping. For example, if a state contains three 1s and two 2s (examples of such states can be seen in Figure 1) there are six ways to replace two of the 1s and one of the 2s by special symbols x, y, and z: state $\langle 1, 2, 1, 2, 1\rangle$, for example, would be mapped to $\langle x, y, z, 2, 1\rangle$, $\langle x, 2, 1, y, z\rangle$, $\langle 1, x, y, 2, z\rangle$ etc.

A third way to define a multimapping abstraction ϕ is to use a set of ordinary abstractions, $\phi_1, ..., \phi_n$, that all map to the same abstract space and define $\phi(s) = \{\phi_1(s), ..., \phi_n(s)\}$. Again, n is called ϕ's mapping factor. This is the method illustrated in Figure 1 and which we will use to define multimappings in the rest of the paper.

Minimizing the Effect of Minimizing

The biggest weakness of multimapping compared to either using a single abstraction or using multiple, independent abstractions, is the fact that in computing $h(s)$ it minimizes over the set of states in $\phi(g)$. This is necessary to guarantee admissibility but can weaken the heuristic so much that it is inferior to a heuristic based on a single abstraction mapping. In this section we will look at three ways to reduce the harmful effect of this minimization: (1) choosing an appropriate mapping factor n, (2) Goal Aggregation (minimizing the distance between the states in $\phi(g)$), and (3) Remapping (to prevent goal "creep" in hierarchical search).

Choosing a Mapping Factor

The mapping factor n affects the quality of the heuristic produced by a multimapping in two ways. In computing $h(s)$ a maximum is taken over all the states in $\phi(s)$ so increasing the number of such states, i.e., the mapping factor, will increase the maximum, all other things being equal. At the same time however, a minimum is being taken over all the states in $\phi(g)$ so increasing the mapping factor will make the minimum smaller. It is hard to predict which of these effects will dominate in general, but very large mapping factors will certainly produce poor heuristics. This situation is analogous to the effect of increasing the number of PDBs while keeping the total memory usage constant when maximizing over multiple PDBs (Holte et al. 2006). Increasing the number of PDBs will increase the maximum, all other things being equal, but in order to keep memory use constant, increasing the number of PDBs means each PDB must be smaller, which tends to decrease the values over which the maximum is taken.

To illustrate the effect on the heuristic when the mapping factor is increased, we ran a small experiment on the 8-puzzle. Before describing the experiment, let us review the notions of domain abstraction and granularity. We represent an 8-puzzle state using 9 constants, one for each of the 8 tiles and one for the blank. A "domain abstraction" is a mapping of these constants to a smaller set of constants. For example, ϕ^3 in Table 1 (bottom row) maps the blank to 0 and maps all the tiles to 1. Applied to an 8-puzzle state, this produces an abstract state in which one position in the puzzle is 0 and the rest are 1s. The "granularity" of this domain abstraction is $\langle 8, 1\rangle$ because 8 constants are mapped to one abstract constant and 1 to the other. In general a granularity is a vector of non-increasing values $\langle k_1, k_2, ...\rangle$ with k_i indicating how many constants in the original domain are mapped to the i^{th} abstract constant. When the total number of constants in the original state space is clear from context, the 1s in the granularity vector can be omitted. For example, for the 8-puzzle, granularity $\langle 6, 1, 1, 1\rangle$ would be written as $\langle 6\rangle$.

In this experiment we generated all 280 of the $\langle 3, 3, 2, 1\rangle$ domain abstractions of the 8-puzzle that map the blank to a different constant than any tile. For each mapping factor $n > 1$ we generated a multimapping by choosing n of these domain abstractions uniformly at random. We evaluated the resulting heuristic by counting the average number of 8-puzzle nodes expanded in solving 500 test problems (randomly generated solvable start states). We repeated this 280 times for each value of $n > 1$. For $n = 1$ we evaluated each of the 280 domain abstractions individually. The

original	blank	1	2	3	4	5	6	7	8
ϕ^0	0	1	2	3	3	3	4	4	4
ϕ^1	0	1	2	3	3	3	3	3	3
ϕ^2	0	1	2	2	2	2	2	2	2
ϕ^3	0	1	1	1	1	1	1	1	1

Table 1: Domain abstractions of the 8-puzzle.

n	1	2	3	4	5	24
avg.	3700	1824	1545	1610	1699	4101
std.	495	513	273	276	307	609

Table 2: Nodes Expanded for various mapping factors.

original	1	2	3	4	5	6	7	8	9
ϕ_1	1	1	1	1	2	3	4	5	6
ϕ_2	2	1	1	1	1	3	4	5	6
ϕ_3	3	2	1	1	1	1	4	5	6

Table 3: Abstractions based on Goal Aggregation.

average results and standard deviations, truncated to integer values, are shown in Table 2.

The entry for $n = 1$ shows the average performance if no multimapping is done, i.e., if we just use one domain abstraction to define the heuristic. Using a multimapping based on $n = 2$ domain abstractions reduces the number of nodes expanded by 50%. The number of nodes expanded decreases again when one more domain abstraction is added ($n = 3$) but begins to increase after that. With $n = 24$, the number of nodes expanded is worse than using a single abstraction.

Although this experiment involves just one state space and tiny abstract spaces, we believe the general lesson to be drawn is that multimapping will be most effective when the mapping factor is quite small. In our experiments, we use $n = 3$.

Goal Aggregation

The second technique to reduce the harmful effect of minimizing over all the states in $\phi(g)$ is to choose ϕ so that the maximum distance, Δ, between states in $\phi(g)$ is as small as possible. We call this Goal Aggregation. It is expected to be effective because Δ is an upper bound on the "harm" that can be done by taking the minimum: for any abstract state s',

$$\max_{g' \in \phi(g)} d(s', g') \leq \Delta + \min_{g' \in \phi(g)} d(s', g').$$

For example, $\phi_1(g)$ and $\phi_2(g)$ in Figure 1 are just 1 move apart (reversing one of them produces the other) and therefore the minimum distance to these goal states must be within 1 of the distance to either one of them—it was not a coincidence that $h(s)$ using multimapping in this example was only 1 smaller than $h(s)$ using multiple independent abstractions.

In hierarchical heuristic search, mapping the goal to abstract states that are near one another is expected to have a second benefit: after very few searches at the abstract level the entire space around all the goal states will be fully explored and stored in cache, thereby speeding up subsequent searches and possibly reducing the number of cache entries as well.

Example 1 *As an example of how to construct a set of n domain abstractions based on Goal Aggregation consider the 9-Pancake puzzle with a goal state of $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9\rangle$ and a mapping factor of $n = 3$. The first abstraction, ϕ_1, is picked arbitrarily and applied to the goal state to produce state s'_1. Using ϕ_1 from Table 3, s'_1 would be $\langle 1, 1, 1, 1, 2, 3, 4, 5, 6\rangle$. Next, pick any two operators and apply them separately to s'_1 to produce two adjacent abstract states, e.g., $s'_2 = \langle 2, 1, 1, 1, 1, 3, 4, 5, 6\rangle$ and $s'_3 = \langle 3, 2, 1, 1, 1, 1, 4, 5, 6\rangle$. Each of these abstract states uniquely defines an abstraction, ϕ_2 and ϕ_3 in Table 3, that will map the goal to them (e.g., $\phi_2(goal) = s'_2$).*

For each state space there is a limit on how small Δ can be. We have seen that $\Delta = 1$ is possible for the pancake puzzle. This is not possible for the 8-puzzle for any abstraction that keeps enough information to allow the exact location of the blank to be determined because distinct states that have the blank in the same location are at least four moves from one another, so Δ must be 4 or more. $\Delta = 4$ still makes the states "close" to one another, relatively speaking, since the average distance to the goal state in a $\langle 3, 3\rangle$ abstraction of the 8-puzzle is 16. In the Blocks World our goal state has all the blocks in one stack. Our Blocks World has a hand to pick up and put down blocks, so 8 moves are required to reach a different state with a single stack (the top two blocks have to swap positions) so Δ must be 8 or more. If the mapping factor is 3, Δ must be at least 12 because the top three blocks need to be re-arranged to generate a third state with all the blocks in one stack. With 8 blocks and a $\langle 3, 3\rangle$ abstraction, the average distance to goal is 24.7, so $\Delta = 12$ is too large for Goal Aggregation to be useful.

Remapping

In hierarchical heuristic search there is a sequence of increasingly abstract spaces, $T_0, T_1, ..., T_L$ starting with the original state space $T_0 = S$ (the "base level"), with each space connected to the next by some sort of abstraction mapping. ϕ^i denotes the abstraction that maps T_i to T_{i+1}. As usual, exact distances in space T_{i+1} are used as a heuristic to guide search in T_i.

Assume that ϕ^0 is a multimapping with a mapping factor of n. If ϕ^1 is also a multimapping with a mapping factor of n there could be as many as n^2 goal states in T_2, with the consequence that the heuristic for guiding search in T_1 is very weak. To avoid this proliferation of abstract goal states, we use normal abstraction mappings, not multimappings, for all ϕ^i except ϕ^0. That way there will be at most n goal states at each level.

This does not entirely solve the problem because, as discussed above, a critical factor is the distance between the different goal states relative to the average distance between states. Since the latter is decreasing as we move up the abstraction hierarchy, it is important for the distance between goal states to decrease at least as quickly. In early experiments we did nothing to control this and found that the heuristics became very weak very quickly, especially in the larger state spaces.

Our solution, called "Remapping", is to choose ϕ^1 so that there is only one goal state in T_2. In other words, we choose ϕ^1 so that $\phi^1(g') = \phi^1(g'')$ for all $g', g'' \in \phi^0(g)$. For example, suppose in the pancake puzzle that ϕ^0 has a mapping factor of $n = 2$ and that the two domain abstractions that define ϕ^0, ϕ_1^0 and ϕ_2^0, agree on how to map all the pancakes except pancakes 1, 2, and 3. ϕ_1^0 maps pancakes 1 and 2 to $1'$ and maps pancake 3 to $2'$, whereas ϕ_2^0 maps pancake 1 to $1'$ and maps pancakes 2 and 3 to $2'$. ϕ^0 maps the one goal state in T_0 to two goal states in T_1. To ensure that there is just one goal state in T_2, it suffices to choose any ϕ^1 that maps constants $1'$ and $2'$ to the same constant.

Goal Aggregation and Remapping interact with one another, because the former dictates what the abstract goal states shall be and the latter dictates how those will be mapped to the next higher level. Used together, the granularity of ϕ^2 is entirely determined by Goal Aggregation's choices. This can be problematic if a particular granularity for T_2 is required, as in our experiments.

Example 2 *If Remapping is applied to the three abstract goal states that Goal Aggregation produces in Example 1, it must map constants* $1, 2,$ *and* 3 *to the same constant, which produces a granularity of at least* $\langle 6 \rangle$. *If we required that level to have a granularity of* $\langle 5 \rangle$ *we would need to abandon one of the two techniques or choose a different* ϕ_1 *for the Goal Aggregation process.*

Experiment Design

The aim of our experiments is to compare multimapping abstractions with single abstraction mappings and with multiple, independent abstractions in a hierarchical search setting. We use domain abstractions and the Hierarchical IDA* (HIDA*) search algorithm (Holte, Grajkowski, and Tanner 2005). We use MM-HIDA*, DA-HIDA*, and MA-HIDA* to refer to HIDA* when it is used with multimapping abstractions, single domain abstractions, and multiple, independent abstractions, respectively. The abstraction hierarchies used for each version of HIDA* are illustrated in Figure 3. For DA-HIDA*, every abstract level consists of only one abstract space and each state, at every level, maps to just one state at the next higher level. For MA-HIDA*, each abstract level consists of three separate abstract spaces; each state in the original space is mapped to one abstract state in each space at the first level of abstraction, and each abstract state, at every level, is mapped to just one abstract state at the next higher level. For MM-HIDA*, there is just one abstract space at each level; each state in the original space is mapped to $n = 3$ abstract states in the first level of abstraction, and each abstract state, at every level, is mapped to just one abstract state at the next higher level.

Our experiments were run on a computer with two AMD Opteron 250 (2.4GHz) CPUs and 8GB of memory. We used four problem domains: the sliding-tile puzzle, the Pancake puzzle, Topspin, and the Blocks World with a hand and distinct table positions, a variant of the Blocks World in which there are a fixed number of positions on the table (3 in our experiments) that are distinguishable from each other, so that having all the blocks in a stack in position 1 is not the same state as having them all in a stack in position 2.

We use two sizes of each problem domain. The smaller size allows a large number of abstraction hierarchies of each type to be compared, giving us a very accurate picture of the range of possible behaviours of the different abstraction

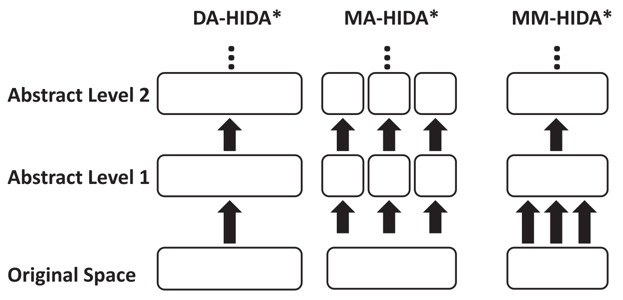


Figure 3: Abstraction hierarchies for the HIDA* variants.

techniques. The larger size allows us to test, in a limited manner, whether the observations made in the small versions continue to hold as the state spaces scale up in size.

Experiments with Small State Spaces

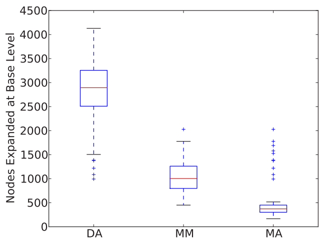
The smaller state spaces used in this experiment were the 3×3 sliding-tile puzzle (8-puzzle), the 9-Pancake puzzle, (10,4)-Topspin (10 tokens and a turnstile of width 4), and the (8,3)-Blocks World (8 blocks, 3 distinct table positions). The abstraction hierarchies always had four levels and the first-level (ϕ^0) abstractions all had a granularity of $\langle 3, 3 \rangle$.[2] A typical $\langle 3, 3 \rangle$ domain abstraction of the 8-puzzle is ϕ^0 in Table 1.

The mapping to the second abstract level (ϕ^1) was created by mapping the two abstract constants that had 3 things mapped to them by ϕ^0 to the same abstract constant and leaving the other abstract constants unique, for a granularity of $\langle 6 \rangle$; see ϕ^1 in Table 1. The mapping to the third abstract level (ϕ^2) adds one additional constant to the group of six that are indistinguishable at the second level (granularity $\langle 7 \rangle$), and the mapping to the fourth and final level (ϕ^3) adds one more constant to this group (granularity $\langle 8 \rangle$).

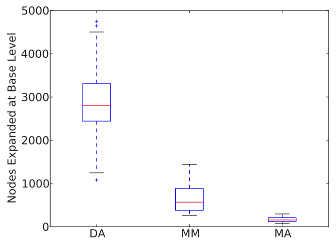
For the 8-puzzle we used all 280 of the $\langle 3, 3 \rangle$ domain abstractions that do not map the blank and a tile together. For the other domains we used the analogous abstractions.[3] For DA-HIDA*, each abstraction was used on its own. For MM-HIDA* on the 8-puzzle, Pancake puzzle, and Topspin, we used each of these abstractions together with two abstractions of the same granularity determined by Goal Aggregation. If the combination of Goal Aggregation and Remapping produced a second level of abstraction that did not have granularity $\langle 6 \rangle$ (the granularity of DA-HIDA*'s second level) we did not include it in our experiments. Thus there are fewer abstraction hierarchies in our experiments for MM-HIDA* than for the other methods (100 for the 8-puzzle, 68 for the 9-Pancake puzzle, and just 13 for (10,4)-Topspin). For MM-HIDA* on the Blocks World, we created 280 sets of domain abstractions by repeating the following process 280 times: (1) choose 6 blocks at random; (2) create three domain abstractions by separating the 6 blocks into

[2] In the standard encoding of TopSpin token 0 is regarded as being fixed in the leftmost position so there are only 9 tokens of consequence in (10,4)-Topspin.

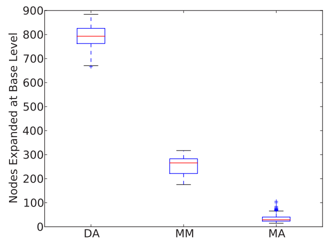
[3] There are 840 such abstractions for the 9-Pancake puzzle, 280 for the other domains.



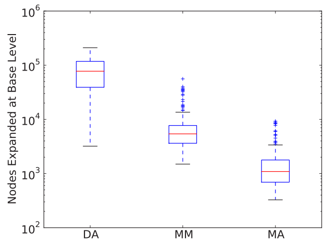
(a) 8-Puzzle



(b) 9-Pancake



(c) (10,4)-Topspin



(d) (8,3)-Blocks World

Figure 4: Average number of nodes expanded at the base level. The y-axis in (d) is on a log scale.

two groups of three at random in three different ways. This process guarantees that the first-level abstractions have granularity $\langle 3, 3\rangle$ and that the second-level abstraction has granularity $\langle 6\rangle$ when Remapping is applied. For MA-HIDA* on all the state spaces, we used the same first-level abstractions as MM-HIDA*. These performed uniformly better, in terms of CPU time, than choosing sets of three domain abstractions at random.

For each state space, we used 500 random, solvable start states as test cases. Each version of HIDA* was run with each abstraction hierarchy. We measured the average memory used by HIDA* (number of cache entries at all levels of the hierarchy), and the average CPU time (in seconds) needed to solve the test cases. We also measured the average number of nodes expanded at the base level as an indication of the effectiveness of the heuristic defined by the first level of abstraction. The results are shown in Figures 4 to 6 using box plots. For each state space there are three box plots, one for each type of abstraction (DA on the left, MM in the middle, MA on the right). The box for a given abstraction method contains exactly half the abstraction hierarchies used for that method; the horizontal line inside the box shows the median performance and the bottom and top of the box represent the 75^{th} and 25^{th} percentiles respectively (lower is better in all these figures). The vertical line below the box extends to the best performance or to 1.5 times the interquartile range ($1.5 \times IQR$), whichever is larger. If there are results beyond $1.5 \times IQR$, they are plotted as individual points. The vertical line and points above the box are analogous, but for the performances in the bottom quartile.

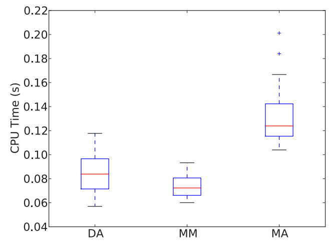
Results in all four state spaces show exactly the same trends. Figure 4 shows that the heuristics created by multimapping abstractions are far superior to the heuristics created by a single domain abstraction and inferior to the heuristics created by multiple, independent abstractions. If pattern databases were being used, these results indicate that search would be faster with heuristics defined by multiple abstractions than with heuristics defined by multimapping since the number of nodes expanded at the base level is strongly correlated with CPU time when PDBs are used. The advantage of multimapping over multiple PDBs in this setting is that it requires n times less memory and preprocessing time if the same n abstractions are used.

However, in the hierarchical search setting the number of nodes expanded at the base level does not entirely dictate CPU time because the cost to search in the abstract levels can be substantial. In Figure 5 we see that MM-HIDA* is 2–3.5 times faster than MA-HIDA* and somewhat faster than DA-HIDA*. Memory usage follows the same pattern (Figure 6): MM-HIDA* uses considerably less memory than MA-HIDA* and slightly less than DA-HIDA*. In terms of both CPU time and memory, MM-HIDA* is the best of the three methods.

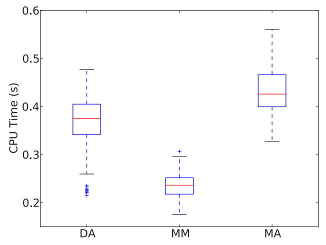
Experiments With Large State Spaces

Our final experiments are with large versions of the problem domains: the normal 4×4 sliding-tile puzzle (15-puzzle) and the "glued tile" variant in which tile 9 (second column, second row from the bottom) cannot be moved from its goal location, the 14-Pancake Puzzle, (15,4)-Topspin, and the (12,3)-Blocks World. We used 100 random, solvable start states as test cases; for the 15-puzzle these were the standard test cases (Korf 1985). For each state space, we hand-generated 5 "reasonable" abstraction hierarchies for each abstraction method (DA, MM, and MA). The granularity of the abstraction hierarchies for DA-HIDA* and MM-HIDA* were identical at all levels. The granularity of the first-level abstraction was $\langle 6, 2\rangle$ for the 15-Puzzle, 14-Pancake puzzle, and (15,4)-Topspin, and $\langle 5\rangle$ for the (12,3)-Blocks world. Each successive level reduced the number of abstract constants by one. The uppermost abstract level had just one abstract constant except for the 15-puzzle where it had two, "blank" and "not blank". For MM-HIDA* we always used Remapping and we also used Goal Aggregation except on the Blocks World. MA-HIDA* used the same abstractions as MM-HIDA* except on the 15-puzzle where it was necessary to use a coarser-grained first-level abstraction, $\langle 8, 3\rangle$, to fit in memory. Since the number of abstractions is small and non-random, the purpose of this experiment is to test if the conclusions drawn on the small state spaces apply equally well to reasonable, hand-crafted abstractions of larger state spaces.

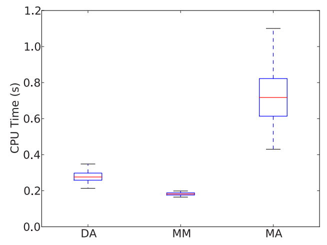
Tables 4 to 8 present the average number of nodes expanded at the base level, the average CPU time (in seconds) and average memory needed (number of cache entries at all levels) for each abstraction hierarchy tested. The first column indicates with a check mark the type of abstraction used in each row. The rows are sorted by average CPU time. The conclusions drawn from the experiments on the smaller



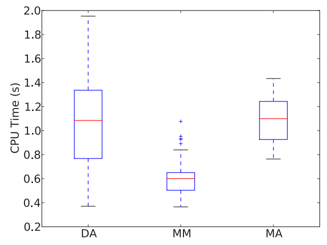
(a) 8-Puzzle



(b) 9-Pancake

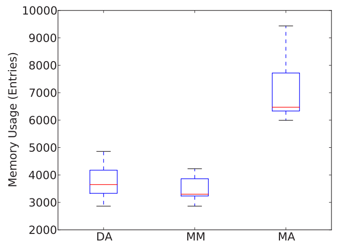


(c) (10,4)-Topspin

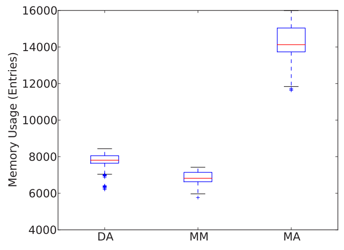


(d) (8,3)-Blocks World

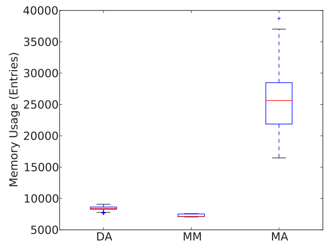
Figure 5: Average CPU time (seconds).



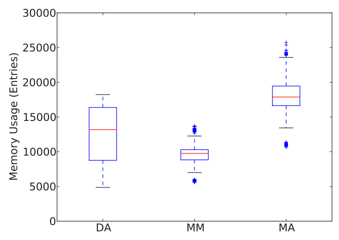
(a) 8-Puzzle



(b) 9-Pancake



(c) (10,4)-Topspin



(d) (8,3)-Blocks World

Figure 6: Average memory used (number of cache entries).

DA	MM	MA	Nodes	CPU(s)	Mem ($\times 10^7$)
	✓		3,669,519	768.2	2.784
	✓		1,667,888	782.0	2.758
	✓		7,945,182	798.1	2.802
✓			20,571,539	806.4	2.758
✓			5,545,817	914.6	2.687
✓			24,547,168	1,029.4	2.753
	✓		2,910,399	1,040.6	4.430
		✓	552,289	1,224.7	3.673
✓			54,322,104	1,253.1	4.453
		✓	906,147	1,344.0	4.297
		✓	789,905	1,360.4	4.295
		✓	1,007,468	1,557.2	5.092
		✓	926,771	1,616.2	4.435
	✓		1,178,024	1,681.8	5.978
✓			16,793,560	1,909.7	5.834

Table 4: Results for the 15-Puzzle.

DA	MM	MA	Nodes	CPU(s)	Mem ($\times 10^6$)
	✓		78,519	54.2	2.176
✓			1,172,133	58.4	2.143
	✓		3,104,027	60.3	1.529
	✓		3,777,306	65.4	2.163
✓			4,247,013	73.8	2.287
	✓		345,574	73.9	2.474
✓			9,108,588	74.2	2.087
✓			23,194,543	98.5	2.144
✓			20,474,809	103.5	1.469
	✓		12,363,685	104.2	2.189
		✓	29,862	112.8	4.196
		✓	1,405,176	114.3	3.269
		✓	1,502,349	115.2	4.307
		✓	4,333,375	131.3	4.430
		✓	108,947	134.9	4.687

Table 5: Results for the Glued 15-puzzle.

state spaces are confirmed here. MM-HIDA* is the fastest method in all domains except the Blocks World, where it is slower than the two fastest DA-HIDA*s but still occupies 3 of the top 6 positions. MA-HIDA* is much slower than the fastest method even though it expands far fewer nodes at the base level than the other methods. Also as before, memory usage is highly correlated with CPU time.

Conclusions

We have introduced multimapping abstractions and proven that they produce admissible, consistent heuristics. We have described three different ways of defining multimapping abstractions in practice and examined one in depth—multiple domain abstractions that all map to the same abstract space. We proposed three methods for overcoming the main potential weakness of multimapping abstractions: (1) using a small mapping factor, (2) Goal Aggregation, and (3) Remapping. Our experiments showed that HIDA* with multimapping abstractions solved problems using less time and less memory than HIDA* with one domain abstraction or with multiple, independent domain abstractions.

DA	MM	MA	Nodes	CPU(s)	Mem ($\times 10^6$)
	✓		587,931	284.6	7.646
	✓		454,853	293.3	7.976
	✓		480,962	304.8	7.934
	✓		217,028	312.2	7.887
	✓		206,675	322.1	7.758
✓			1,369,956	397.9	8.892
✓			1,176,908	401.2	8.778
✓			2,861,843	507.0	11.745
		✓	118,043	525.1	13.870
✓			1,818,312	531.6	11.780
		✓	114,719	535.1	14.197
		✓	79,196	540.1	13.837
		✓	46,751	565.3	13.849
		✓	102,077	593.6	14.350
✓			1,253,427	635.1	11.963

Table 6: Results for the 14-Pancake Puzzle.

DA	MM	MA	Nodes	CPU(s)	Mem ($\times 10^6$)
	✓		11,870	143.5	5.390
	✓		12,312	147.6	5.433
	✓		12,471	150.7	5.387
	✓		12,849	153.8	5.328
	✓		13,498	160.8	5.456
✓			47,907	183.3	6.649
✓			52,970	186.8	6.676
✓			47,905	197.2	6.604
✓			43,576	199.5	6.720
✓			47,959	217.5	6.651
		✓	3,506	279.7	9.838
		✓	3,541	286.0	9.783
		✓	3,338	300.5	9.874
		✓	3,458	301.5	9.783
		✓	3,759	309.8	10.018

Table 7: Results for (15,4)-TopSpin.

DA	MM	MA	Nodes	CPU(s)	Mem ($\times 10^6$)
✓			43,896	77.4	2.062
✓			589,399	106.4	2.464
	✓		6,202,221	123.1	2.117
		✓	2,109	126.3	3.506
	✓		1,065,182	131.8	3.514
	✓		3,138,661	138.1	3.033
✓			6,310,237	143.7	2.930
✓			6,271,341	147.6	3.531
	✓		7,647,763	166.4	2.538
		✓	22,149	187.4	4.441
✓			18,038,164	194.9	3.510
		✓	5,214	204.1	5.295
	✓		13,099,419	218.9	3.648
		✓	46,776	222.8	6.567
		✓	48,895	229.3	6.690

Table 8: Results for the (12,3)-Blocks World.

Acknowledgements

We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council.

References

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Proceedings of the Canadian Conference on Artificial Intelligence*, volume 1081 of *LNAI*, 402–416. Springer.

Domshlak, C.; Katz, M.; and Shleyfman, A. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

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 *Proc. 6th Intl. Symposium on Abstraction, Reformulation and Approximation (SARA 2005)*, volume 3607 of *LNAI*, 121–133.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

Pang, B., and Holte, R. C. 2011. State-set search. In *Symposium on Combinatorial Search (SoCS)*.

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.