Hierarchical A*: Searching Abstraction Hierarchies Efficiently

Robert C. Holte — Computer Science Dept., University of Ottawa, Ottawa, Canada K1N 6N5, holte@csi.uottawa.ca

M.B. Perez — BNR Ltd., Ottawa, Canada

R.M. Zimmer — Computer Science Dept., Brunel University, Uxbridge, England

A.J. MacDonald — Electrical Engineering Dept., Brunel University, Uxbridge, England

Abstract

Abstraction, in search, problem solving, and planning, works by replacing one state space by another (the "abstract" space) that is easier to search. The results of the search in the abstract space are used to guide search in the original space. For instance, the length of the abstract solution can be used as a heuristic for A* in searching in the original space. However, there are two obstacles to making this work efficiently. The first is a theorem (Valtorta, 1984) stating that for a large class of abstractions, "embedding abstractions," every state expanded by blind search must also be expanded by A* when its heuristic is computed in this way. The second obstacle arises because in solving a problem A* needs repeatedly to do a full search of the abstract space while computing its heuristic. This paper introduces a new abstraction-induced search technique, "Hierarchical A*," that gets around both of these difficulties: first, by drawing from a different class of abstractions, "homomorphism abstractions," and, secondly, by using novel caching techniques to avoid repeatedly expanding the same states in successive searches in the abstract space. Hierarchical A* outperforms blind search on all the search spaces studied.

Introduction

Several researchers have investigated abstraction as a means of automatically creating admissible heuristics for A* search (Gaschnig, 1979; Pearl, 1984; Prieditis, 1993; Guida & Somalvico, 1979). The general idea is to equate h(S), the estimate of the distance in the state space SS from state S to the goal G, with the exact distance in some other state space, SS′ — the "abstraction" of SS — from the state in SS′ corresponding to S to the state corresponding to G. As long as the distance between every pair of states in SS is greater than or equal to the distance between the corresponding states in SS′ h(S), defined in this way, is an admissible heuristic. This observation has been formalized in (Prieditis,1993) in the definition of an "abstraction transformation", as follows.

A "state space" is a pair <States,d>, where States is a set of states, and $d(S_1,S_2)$ is the distance (length of the shortest path) between two states ($d(S_1,S_2)=\infty$ if there is no path from S_1 to S_2), and a "problem" is a pair <Start,Goal>, where Start is a state and Goal is a (set of) state(s)[1]. Given state spaces SS=<States,d> and SS′=<States′,d′>, a mapping ϕ: States $\rightarrow$ States′ is an abstraction transformation (of SS) if every distance in SS is less than or equal to the corresponding (according to ϕ) distance in SS′. Formally: $d'(\phi(S_1),\phi(S_2)) \leq d(S_1,S_2)$ for all S_1, $S_2 \in$ States. Given a problem <Start,Goal> for SS the corresponding problem for SS′ is <ϕ(Start),Goal′>, where $Goal' \supseteq \{\phi(g) \mid g \in Goal\}$. Often we will refer to ϕ(SS) without specifying SS′. In this case, SS′ is implicitly taken to be <$image_\phi$,d_ϕ>, where $image_\phi = \{\phi(s) \mid s \in States\}$ and $d_\phi(S'_1,S'_2) = \min\{d(s_1,s_2) \mid \phi(s_1)=S'_1 \text{ and } \phi(s_2)=S'_2\}$.

The earliest and most commonly studied type of abstraction transformation is the "embedding". Informally, ϕ is an embedding transformation if it "adds edges" to SS. For example, techniques that add macro-operators to, or drop preconditions from a state space definition are embeddings. Formally, ϕ is an embedding if SS′ is a supergraph of SS, i.e. if States′ $\supseteq$ States, and $\phi(S)=S$ for all $S \in$ States. Because all the states and edges (with the same costs) in SS are present in SS′, distances in SS′ cannot possibly be longer than the corresponding distances in SS. Thus, any embedding is an abstraction transformation. Embeddings are the type of transformation studied in (Gaschnig,1979; Pearl,1984; and Guida & Somalvico, 1979).

The other main type of abstraction transformation is the "homomorphism" (or "natural transformation" (Kibler,1982)). Informally, a homomorphism ϕ groups together several states in SS to create a single abstract state. For example, techniques that drop a predicate entirely from a state space description (Knoblock,1994) are homomorphisms. Formally, if ϕ is any many-to-one mapping on SS's states then ϕ(SS), as defined above, is a homomorphism of SS.

[1] in (Prieditis,1993) a problem is simply a Start state and the Goal is part of the state space. This difference does not materially affect the general definition of what an "abstraction transformation" is.

The aim of creating a heuristic is to speed up search. Without a heuristic, A* blindly searches in the original space. With a heuristic, A*'s search will be more focused, and the search effort in the original space will be reduced by some amount (the "saving"). The primary risk in using a heuristic created by abstraction is that the total cost of computing h(-) over the course of the search can exceed the saving. If this happens the use of the heuristic is harmful: the cost to compute the heuristic outweighs the benefit derived from using it.

A cost-benefit analysis of heuristics automatically generated by embedding transformations was presented in (Valtorta,1984). It was proven that if SS is embedded in SS′ and h(-) is computed by blindly searching in SS′, then A* using h(-) will expand every state that is expanded by blindly searching directly in SS. In other words, this method of defining a heuristic cannot possibly speed up search – the total number of nodes (states) expanded using the heuristic (including those expanded in the abstract space) must equal or exceed the number expanded when blindly searching in the original space. Based on this theorem, we define "Valtorta's Barrier" to be the number of nodes expanded when blindly searching in a space. Valtorta's theorem states that this barrier cannot be "broken" using any embedding transformation.

Valtorta's theorem does not rule out the possibility of breaking Valtorta's barrier by using other sorts of transformations. However, to date Valtorta's barrier has been broken only twice. Absolver II (Prieditis,1993) created cost-effective heuristics for the state spaces (and particular goals) associated with the Fool's Disk and Instant Insanity puzzles by using homomorphic transformations. On the other 11 problems to which Absolver II was applied abstraction by itself was unable to create a cost-effective heuristic (on 5 of these cost-effective heuristics were created when abstraction was used together with other sorts of state-space transformations).

In this paper we show that Valtorta's barrier can be broken on a wide variety of search spaces using a general-purpose homomorphic abstraction technique. First we present a generalized version of Valtorta's theorem that applies to all types of abstraction transformations. This theorem provides a lower bound on the number of nodes that must be expanded using any abstraction transformation. For homomorphic transformations this bound is lower than the number of nodes expanded by blindly searching in the original space. This suggests that it is possible, at least in principle, for homomorphic abstractions to break Valtorta's barrier. However our initial experimental results show that the mere fact that abstractions are homomorphic does not guarantee that cost-effective heuristics will be created. We found that search using the homomorphic abstraction hierarchy expands many more states than blind search – over 17 times as many in one of the testbed search spaces. The cause of these negative results is identified and two complementary approaches to alleviating the problem are presented. The first is an algorithmic approach. The A* algorithm is customized for searching in abstraction hierarchies. The customized version, called Hierarchical A*, expands roughly 6 times fewer states than the original A* and breaks Valtorta's barrier in about half the testbed search spaces. The second approach is to create abstractions that are less fine-grained. This also breaks Valtorta's barrier for some of the testbed search spaces. When the two approaches are combined, Valtorta's barrier is broken for all the testbed search spaces.

Valtorta's Theorem Generalized

The main theorem in (Valtorta,1984), which is specific to embeddings, is easily generalized to any abstraction transformation. The generalized theorem is (for a formal statement and proof see (Holte et al., 1995)):

> Let ϕ be any abstraction mapping from state space SS to SS′ and $h_\phi(S)$ be the length of the shortest path in SS′ from $\phi(S)$ to $\phi(Goal)$ (computed by searching blindly in SS′). If S is any state necessarily expanded when a given problem is solved by blind search directly in state space SS, then, when the problem is solved using A* search with $h_\phi(-)$ in SS, either S itself will be expanded or $\phi(S)$ will be expanded.

When ϕ is an embedding, $\phi(S)=S$ and we get Valtorta's theorem: every state necessarily expanded by blind search is also necessarily expanded by A* using h_ϕ. But if ϕ is a homomorphism speedup is possible because the expansion of one state in the abstract space is equivalent to the expansion of many states in the original space.

A simple example illustrates that heuristics based on homomorphic abstractions can indeed greatly reduce the number of nodes expanded. Suppose state space SS is an $n \times n$ grid and that the problem being solved is to get from the bottom left corner (1,1) to the bottom right corner $(n,1)$. To solve this problem with blind search, it is necessary to expand all states at distance $(n-1)$ or less; thus $O(n^2)$ states are expanded. Let ϕ be the mapping that ignores the second co–ordinate. $\phi(SS)$, then, is a linear space with states (1), (2),... (n). Computing $h_\phi(Start)$ requires finding a path in $\phi(SS)$ from (1) to (n). $O(n)$ states are expanded during this search, but notice that this search generates the value of $h_\phi(S)$ for all S, so no further search need be done in the abstract space. Furthermore, the $h_\phi(-)$ defined by this particular abstraction is a perfect heuristic, and therefore the search in SS will expand only those states on the solution path. Thus, the total number of nodes

expanded by A* using $h_\phi(-)$ is $O(n)$, far fewer than the $O(n^2)$ nodes expanded by blind search.

The theorems in (Valtorta,1984) and (Kibler,1982) concerning the monotonicity of heuristics based on abstraction transformations also generalize: for any abstraction transformation ϕ, of any type, $h_\phi(S)$ is a monotone (consistent) heuristic (Holte et al.,1995).

Hierarchical Search using A*

Abstractions are created in the current system using the "max-degree" STAR abstraction technique described in (Holte et al.,1996). This technique is very simple: the state with the largest degree is grouped together with its neighbours within a certain distance (the "abstraction radius") to form a single abstract state. This is repeated until all states have been assigned to some abstract state. Having thus created one level of abstraction the process is repeated recursively until a level is created containing just one state. This forms an abstraction hierarchy whose top-level is the trivial search space. The bottom, or "base", level of the hierarchy is the original search space. This method of creating abstractions was originally designed and has proven very successful for the search technique known as "refinement" (Holte et al.,1996; Holte et al.,1994). It was not entirely clear at the outset of the present work whether it would create abstraction hierarchies suitable for A* search. It certainly is not suitable for abstracting search spaces in which different operators have different costs, but that consideration does not arise in the experiments in this paper.

The implementation of A* is standard except that in estimating the distance to the goal from a non-goal state, S, it uses the minimum cost of the operators applicable to S in addition to whatever other heuristic estimates might be available. When multiple sources of heuristic information are available, they are combined by taking their maximum. This way of combining multiple heuristics is guaranteed to be admissible if the individual heuristics are admissible, but it may not be monotone even though the individual heuristics are (for an example to the contrary see the next section). However, it is easy to see that combining the minimum operator cost with any monotone heuristic produces a monotone heuristic if every operator has an inverse of the same cost, as is the case in the search spaces in our experiments.

The "cheapest operator" information is most useful when no other heuristic estimates are available (Blind Search: A* with h(S)=0 for all S). In this case its use considerably reduces the number of nodes expanded. When other heuristic estimates are available the "cheapest operator" information is only useful for states estimated by the other means to be very close to the goal.

Hierarchical search using A* is straightforward. As usual, at each step of the A* search a state is removed from the OPEN list and each of its successors is added to the OPEN list (if it has not previously been opened). To add a state S to the OPEN list, h(S) must be known. This is computed by searching at the next higher level of abstraction, using the abstract state corresponding to S, ϕ(S), as the abstract start state and ϕ(goal) as the abstract goal. When an abstract solution path is found, the exact abstract distance from ϕ(S) to ϕ(goal) is known. As described in the preceding paragraphs this is combined with other estimates (e.g. the cost of the cheapest operator applicable to S) to produce the final h(S) value.

When the abstract path from ϕ(S) to ϕ(goal) is found, exact abstract distance-to-goal information is known for *all* abstract states on this path. And each of these abstracts states corresponds, in general, to many states in the level "below" (the state space containing S). Thus a single abstract search produces heuristic estimates for many states. All this information is cached. If an h-value is needed for any of these states, it is simply looked up without any search being done at the abstract level. This idea was illustrated in the example above, where a single search at the abstract level produced h(-) information for all the base level states. Despite this caching technique, it is generally true that to solve a single base level problem, a hierarchical search technique will need to solve many problems at the first level of abstraction. And each one of these abstract problems will require solving many problems at the next level of abstraction, etc. The number of nodes expanded to solve a single base level problem is the total number of states expanded during all searches related to that base level problem at all levels of abstraction (including the base level itself).

The hierarchical search technique just described will be referred to as naive hierarchical A* to contrast it with the versions presented in the next section. The various hierarchical A* techniques were evaluated empirically on 8 state spaces (described in (Holte et al.,1995)). All have invertible operators and the cost of applying all operators is the same (1). Test problems for each state space were generated by choosing 100 pairs of states at random. Each pair of states, <S1,S2>, defined two problems to be solved: <start=S1,goal=S2> and <start=S2,goal=S1>. To permit detailed comparison the same 200 problems were used in every experiment. The "nodes expanded" results shown are averages over these 200 problems.

Naive hierarchical A* expands many more states than Blind Search in every testbed search space (see Table 1). The clue to understanding why this happens is the fact that naive hierarchical A* expands many more states than there are states in the entire abstraction hierarchy (Table 1, second column). This implies that some states are being expanded many times in the course of solving a single base

Table 1. Naive Hierarchical A*. (radius = 2)

Search Space	# States (all levels)	Nodes Expanded: Blind Search	Hierarchical A*: all levels	Hierarchical A*: base level
Blocks-5	1166	389	2766	118
5-puzzle	961	348	3119	224
Fool's Disk	4709	1635	12680	629
Hanoi-7	2894	1069	18829	701
KL2000	3107	1236	7059	641
MC 60-40-7	2023	934	2412	702
Permute-6	731	286	806	77
Words	5330	1923	19386	604

level problem. Since A* with a monotone heuristic never expands the same state twice in a single search, the duplication must arise from the same states being expanded on many different searches associated with the same base level search. The next section presents methods for greatly reducing this sort of duplication.

The number of nodes expanded in the base level by hierarchical A* is a fundamental limit on the amount of speedup that A* can achieve with the given abstraction hierarchy. This number indicates the quality of the heuristic, i.e. how much the heuristic reduces the search effort in the original space. For example, in the Permute-6 search space hierarchical A* expanded 77 states in the base level, a reduction of 75% compared to the number expanded by Blind Search. This is a modest savings. And yet this is the best of the heuristics in Table 1. At the other extreme is the heuristic created for the Missionaries and Cannibals search space (MC 60-40-7) which produced a saving of only 25%. The quality of the heuristics is a property of the abstraction technique and can only be improved by using a different abstraction technique. Except for varying the granularity of abstraction, this topic is beyond the scope of the present paper.

The fact that the heuristics produced by the current abstraction technique result in only a modest reduction in the number of nodes expanded at the base level makes breaking Valtorta's barrier particularly challenging. In order to break the barrier the total number of nodes expanded in all the abstract searches must be less than the savings the heuristics produce. For example, consider the 5-puzzle. The heuristic created by abstraction results in A* expanding 224 states at the base level, a saving of 124 states over Blind Search. In order to break Valtorta's barrier the cost (in terms of nodes expanded at all the abstract levels) of computing the heuristic for all these 224 states (and the additional states that were opened but never closed) must not exceed 124. This is not impossible: expanding a few states at the abstract level can provide heuristic values for many states at the base level, as the example above illustrates.

Table 2. Hierarchical A*. (radius = 2)

Search Space	Nodes Expanded: Blind Search	Hierarchical A*: Naive	V1	V2	V3
Blocks-5	389	2766	1235	478	402
5-puzzle	348	3119	1616	854	560
Fool's Disk	1635	12680	8612	3950	**1525**
Hanoi-7	1069	18829	10667	5357	3174
KL2000	1236	7059	3490	1596	**1028**
MC 60-40-7	934	2412	1531	1154	**863**
Permute-6	286	806	482	**279**	**242**
Words	1923	19386	7591	2849	**1410**

Customizing A* for Hierarchical Search

In hierarchical search, a single base level search can spawn a large number of searches at the abstract levels. As the preceding results show, these searches will often expand many of the same states. The key to reducing this duplication is the observation that **all searches related to the same base level search have the same goal.** Naive hierarchical A*'s simple caching strategy exploits this observation to a small extent, because its cache at a particular level of abstraction is not cleared until the goal at that level changes. As the results in Table 1 show, additional ways of exploiting this observation are needed if hierarchical A* is to be effective.

When a search at an abstract level terminates the exact distance to the goal, h*(S), is known for every abstract state S on the solution path. This information can be cached and used in lieu of h(S) in subsequent searches with the same goal. This improves the quality of the heuristic at this level of abstraction and therefore may reduce the number of nodes expanded by subsequent searches. This technique is called h*-caching. As can be seen in Table 2 (column V1) this roughly halves the number of nodes expanded.

The heuristic produced by h*-caching is not monotone. For example, suppose all operators have cost 1 and that h(S)=1 for all non-goal states. After solving one problem and caching the h*(-) values, states on the solution path may have quite large h(-) values, while their neighbours off the solution path will still have h(-)=1. Thus the difference between neighbouring h(-) values will be greater than 1, the "distance" between the two states. Therefore, the new h(-) function will not satisfy the definition of monotone.

Because the h(-) function is not monotone it is possible for a state to be closed prematurely, i.e. closed before the shortest path to it from the start state has been found. With non-monotone heuristics it is, in general, necessary to re-open such states in order to be guaranteed of finding the shortest path. However, with the type of non-monotone heuristic produced by h*-caching this is not necessary and

in fact our A* implementation has no provision for re-opening closed states. To see this let S be the start state and P a prematurely closed state. For P to be prematurely closed it must be that h*(P) is not known and that each shortest path from S to P passes through some state, X, for which h*(-) is known. If no such X is on any shortest path from S to Goal then neither is P, so P's being prematurely closed is irrelevant. On the other hand, suppose the shortest path from S to Goal passes through such an X. The fact that h*(X) is known means that in a previous search X was on the solution path and therefore a shortest path from X to Goal was previously discovered. For every state on this path h*(-) is known. Therefore none of these states has yet been expanded in the current search and so this segment of the shortest path from S to Goal will be found without having to re-open any closed states.

As just noted, for every state X for which h*(X) is known a shortest path from X to Goal is also known. If this path is cached in addition to h*(X) there is no need to expand X. This is because knowing a path of length g(X) to X is equivalent to knowing a path of length g(X)+h*(X) to Goal. Thus, instead of adding X to the OPEN list, one can add Goal instead and terminate search, as usual, when Goal is atop the OPEN list. This technique is called optimal-path caching. Column V2 in Table 2 shows the number of nodes expanded when optimal-path caching is added to naive hierarchical A*. Optimal-path caching produces about double the savings of h*-caching, and breaks Valtorta's barrier in the Permute-6 search space.

The final technique pertains to states that were opened (or closed) during search but are not on the solution path. For each such state, S, a distance from the start, g(S), is known. If the solution path is length P, then by its optimality it must be that $P \leq g(S) + h^*(S)$. Re-arranging this as $P-g(S) \leq h^*(S)$ we see that P–g(S) is an admissible heuristic, which we call the P-g heuristic. When all operators have inverses it can be shown (Holte et al.,1995) that combining the P-g heuristic with any monotone heuristic produces a monotone heuristic. P-g caching subsumes h*-caching because $P-g(S) = h^*(S)$ when S is on the solution path. Finally, it is not necessary to compute P–g(S) if S is open when the search terminates because in this case $P \leq g(S)+h(S)$ and therefore $P-g(S) \leq h(S)$.

The system V3 includes optimal-path caching and P-g caching. It breaks Valtorta's barrier in 5 of the 8 search spaces used for testing (the bold figures in Table 2). This shows that with homomorphic abstractions it is possible to achieve in practice what is theoretically impossible to achieve with embeddings.

Varying the Granularity of Abstraction

Our abstraction technique allows us to control the granularity of the abstractions created by setting the radius of abstraction. The preceding results were obtained with an abstraction radius of 2, which means a state is grouped with its immediate neighbours. The net effect on "nodes expanded" of increasing the abstraction radius is not clear because two antagonistic effects interact. On one hand, a larger radius means that the abstract spaces contain fewer states and also that a single abstract search produces heuristic values for more states. These factors will reduce the number of abstract nodes expanded by hierarchical A*. On the other hand, a larger radius means the heuristic is less discriminating: this tends to increase the number of nodes expanded (if the heuristic is completely undiscriminating, A* degenerates to Blind Search). (Prieditis & Davis, 1995) presents an initial quantitative analysis of the relation between "abstractness" (i.e. granularity) and the accuracy of the resulting heuristics.

Table 3. Hierarchical A*. (best radius)

Search Space [radius]	Nodes Expanded: Blind Search	Nodes Expanded: Hierarchical A* Naive	Nodes Expanded: Hierarchical A* V3	CPU seconds: Blind Search	CPU seconds: V3
Blocks-5 [5]	389	611	**309**	69	86
5-puzzle [12]	348	354	**340**	36	40
Fool's Disk [4]	1635	**1318**	**1172**	872	902
Hanoi-7 [20]	1069	1097	**1055**	102	108
KL2000 [5]	1236	1306	**1072**	398	384
MC 60-40-7 [4]	934	**822**	**803**	266	253
Permute-6 [5]	286	**201**	**194**	82	67
Words [3]	1923	9184	**1356**	1169	1273

Experiments were run with the abstraction radius set at values 2 through 5, and at larger values for some search spaces. The number of nodes expanded by hierarchical A* decreased as the radius increased until a minimum was reached; increasing the radius beyond this point caused the number of nodes expanded to increase. Table 3 shows the results of the best radius found for V3 for each search space (the best radius for naive hierarchical A* was sometimes larger). V3 now breaks Valtorta's barrier on over half the problems in every search space (and on over 95% of the problems in 2 of the spaces).

Although the best radii are fairly small, in every case they represent a significant fraction of the search space diameter. The abstraction hierarchies thus created had only one non-trivial abstract level, and it contained a small number of states. It is surprising that so coarse a heuristic is able to reduce the number of nodes expanded.

A consequence of there being just a single small non-trivial abstract level is that the algorithmic enhancements introduced above have a much reduced effect: in several of the spaces naive hierarchical A* expands only slightly more states than V3. However, the algorithmic enhancements are important because with most techniques for creating abstractions there is no easy way to control the

granularity of the abstractions produced. Unlike naive hierarchical A*, V3 is robust: it performs well on abstractions of any granularity and is therefore the search algorithm of choice when granularity cannot be controlled.

"Nodes expanded" is a convenient theoretical measure, but it does not completely capture all the "work" that is done by a search system, especially by a hierarchical system that must repeatedly initialize the searches at the abstract levels and pass information from one level to the next. The actual speed of a search system depends on all these operations. Table 3 reports the CPU time taken by Blind Search and V3 to solve all 200 problems. Of course, one must be very cautious in interpreting CPU time results, as they can be heavily affected by low-level implementation details. In the present implementation, which is rather clumsy in many of its low-level details, it appears that the reduction in nodes expanded produced by V3 almost perfectly balances the additional overheads involved in hierarchical search.

With the aims of minimizing the overheads associated with hierarchical search and of evaluating Hierarchical A* on larger state spaces and different methods of homomorphic abstraction we have recently reimplemented the hierarchical search and caching techniques. Abstractions in this system are user-supplied, not created automatically, but to date we have not attempted to optimize the abstractions as our goal has been to evaluate Hierarchical A* on "typical" abstractions. Initial results with the new implementation confirm the general pattern of results for nodes expanded seen in Table 3: in some spaces there is a significant reduction in number of nodes expanded, while in others (e.g. Towers of Hanoi) V3 expands only slightly fewer nodes. However, the CPU time results are very much improved. For the 8-puzzle, for example, V3 reduces CPU time by a factor of 10 (it reduces the number of nodes expanded by a factor of 6.5).

Conclusions

In this paper we have shown that A* search using heuristics created automatically by homomorphic abstractions can "break Valtorta's barrier", that is, it can outperform blind search in terms of number of nodes expanded. This was accomplished in all the state spaces used in the experiments. To achieve this it was necessary to add two novel caching techniques to A*, optimal-path caching and P-g caching, and also, in some cases, to choose abstractions of suitable granularity. The reduction in number of nodes expanded was not large, but this was due to limitations of the abstraction technique and not to excessive algorithmic overheads. The development of an abstraction technique well-suited to hierarchical A* search is an important topic for future research.

Acknowledgements

This research was supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada. The software was written in part by Denys Duchier and Chris Drummond. Thanks to Marco Valtorta and Lise Getoor for their encouragement and helpful comments. The $n \times n$ grid example is derived from an example by Marco Valtorta.

References

Gaschnig, J. (1979), "A Problem Similarity Approach to Devising Heuristics: First Results", *Proc. IJCAI'79*, pp. 301-307.

Guida, G. and M. Somalvico (1979), "A Method for Computing Heuristics in Problem Solving", *Information Sciences*, vol. 19, pp. 251-259.

Holte, R.C., T. Mkadmi, R.M. Zimmer, and A.J. MacDonald (1996), "Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach". to appear in *Artificial Intelligence*.

R.C. Holte, M.B. Perez, R.M. Zimmer, A.J. MacDonald (1995), "Hierarchical A*: Searching Abstraction Hierarchies Efficiently", technical report TR-95-18, Computer Science Dept., University of Ottawa.

Holte, R.C., C. Drummond, M.B. Perez, R.M. Zimmer, and A.J. MacDonald (1994), "Searching with Abstractions: A Unifying Framework and New High-Performance Algorithm", *Proc. 10th Canadian Conf. on Artificial Intelligence (AI'94)*, Morgan Kaufman, pp. 263-270.

Kibler, D. (1982), "Natural Generation of Admissible Heuristics", technical report TR-188, Information and Computer Science Dept., University of California at Irvine.

Knoblock, C.A. (1994), "Automatically Generating Abstractions for Planning", *Artificial Intelligence*, vol. 68(2), pp. 243-302.

Pearl, J. (1984), *Heuristics*, Addison-Wesley.

Prieditis, A. and R. Davis (1995), "Quantitatively relating abstractness to the accuracy of admissible heuristics", *Artificial Intelligence*, vol. 74, pp. 165-175.

Prieditis, A. (1993), "Machine Discovery of Admissible Heuristics", *Machine Learning*, vol.12, pp. 117-142.

Valtorta, M. (1984), "A result on the computational complexity of heuristic estimates for the A* algorithm", *Information Sciences*, vol. 34, pp. 48-59.