New Strategies in Learning Real Time Heuristic Search

Stefan Edelkamp *, Jürgen Eckerle *

Abstract

In contrast to off-line search algorithms such as A^* and IDA^*, in real-time heuristic search we have to commit a move within a limited search horizon or time. One well known algorithm in this class is RTA^*.

An algorithm is said to learn if it improves its performance over successive problem trials. In RTA^* the heuristic estimation is in general not admissible. Thus RTA^* has to be modified to a variant $LRTA^*$ that is capable of learning. The aim of the strategies proposed in this paper is to improve the estimations found in $LRTA^*$. First, we examine two new schemas *forward updating* and *backward updating* for $LRTA^*$. Then we propose $CRTA^*$ which works similar to RTA^* but terminates with admissible heuristic values. It is shown that the strategy used in $CRTA^*$ can be made efficiently. Combined with lazy evaluation updating $CRTA^*$ leads to an improved real time learning algorithm called $SLRTA^*$. Experimentally we show that $CRTA^*$ expands significantly less nodes than $LRTA^*$ and thus converges faster to the optimal values.

Introduction and Background

State space search problems are defined by a set of states Q, a transition function $\delta : Q \rightarrow Q$, an initial state s and a set of goal states $F \subset Q$. A path from s to $g \in F$ is called a solution path. Usually a weight w for each state transition is given and the task is to find the solution path with minimal weight. The state space can be represented by a weighted graph $G = (V, E, w)$, in which the shortest path has to be found. We may assume that G is undirected, i.e., $(v, u) \in E$ and $w(u, v) = w(v, u)$ hold for each $(u, v) \in E$.

Heuristic search takes advantage of a estimation function h for the shortest path h^* to a goal state. The heuristic is admissible or optimistic if the estimate is a lower bound that is $h(u) \leq h^*(u)$ for all $u \in V$. The heuristic estimation should be inexpensive to compute and is used to focus the search into the direction of F.

* Institut für Informatik, Albert-Ludwigs-Universität, Am Flughafen 17, D-79110 Freiburg eMail: {edelkamp,eckerle}@informatik.uni-freiburg.de

There are two different groups of heuristic search algorithms. Off-line algorithms such as A^* (Hart et al. (1968)) and IDA^* (Korf (1985)) examine various paths to the goal nodes before they follow the best one whereas on-line or real-time algorithm have to commit one move even if no goal node has been found. This move can never be went back on. Thus real-time search may decide to move to a node that is not on the optimal solution path. Real time heuristic search can be used for navigating a robot in a set of rooms. The robot itself has no memory but he can read and write signs in each room. His task is to find one of the goal rooms. The signs may inform the robot about the length of the solution path providing data like lower and upper bounds.

The efficiency of the search in successive trials may be improved since the real-time algorithm can use the new heuristic information at the nodes in the search tree. In some learning algorithms such as $LRTA^*$ (Korf (1990)) these values converge eventually to the optimal ones. The quality of the learning strategy reflects how fast the optimal values can be achieved and how many nodes have to be expanded in each trial.

The paper is organized as follows. First we review backward search, which is a fast off-line algorithm to find the optimal distances from each node to the set of goals. Then we describe the algorithm RTA^* and present results found by Korf (1990). After that the learning variant $LRTA^*$ of RTA^* is described by a slight modification to obtain an admissible heuristic at each node and the new strategies *forward updating* and *backward updating* are introduced. Then we consider two new algorithms $CRTA^*$and $SLRTA^*$, which are based on signs at each node in each direction. The signs can be compared to road signs for the robot, providing the minimal distance he has at least to cover. The crucial fact is that the minimum of all sign information leads to an admissible heuristic. The idea of $CRTA^*$ is to obtain these values in a backward traversal of the solution path correcting wrong estimates. We

show that if the problem graph is a tree then this strategy can be done efficiently. Moreover, if the search tree is balanced we achieve an updating time of $O(\log n)$ with n being the number of generated nodes. In the variant *SLRTA** of *CRTA** we postpone the calculation of the minimum until the time when the node is revisited. Last but not least we present experimental results, draw conclusions and outline some questions for further research.

Backward Search

A state space can be searched from the initial state to the set of goal states, applying operators forward. Alternatively, the state space search can be directed from the goal states to the initial state applying the operators in reverse. Note that in bidirectional search, backward and forward search are carried out concurrently.

There are simple algorithms to determine the shortest path from each node to the set of goal nodes in the explicit representation of the problem graph G. Let n be the number of nodes in the search space. First of all we can solve the all pair shortest path problem by reinvoking Dijkstra's shortest path algorithm (1957) for every non-goal node. Using Fibonacci heaps we achieve an amortized time of $O(n(e + n \log n))$, with e being the number of edges in G (Cormen, Leiserson and Rivest (1993)). Note, that a heuristic simply reweights the problem graph and that a consistent heuristic leads to a positive weighting. When we assume a small number g of goal nodes, we can do even better. We simply use backward search, invoke *Dijkstra* from the set of goal nodes and get a time bound of $O(g(e + n \log n))$. The distance of one node to the set of goal nodes F is the minimum of the shortest path values for all $g \in F$.

The most efficient approach (with amortized time of $O(e + n \log n)$) to find all shortest path values to the set F is to put all goal nodes in the priority queue at once and to calculate the distances f to the set F dynamically. If u is the expanded node and $\Gamma(u)$ the set of successors in the backward search we apply the optimality equation $f(v) = \min\{f(u) + w(u,v), f(v)\}$ to each $v \in \Gamma(u)$. After having executed all iterations, $f(v)$ represents the shortest path from v to F. We conclude that learning shortest path values in state space search is simple when no real-time constraints are given.

Real Time A^*

Real time A^* or *RTA** has been developed by Korf (1990). When expanding u to generate the set of successors $\Gamma(u)$, the following two steps are carried out successively until a goal node has been found:

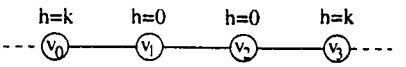

Figure 1: An example for the ping-pong effect.

1. The heuristic value $h(u)$ is set to the second best value of $h(v) + w(u,v)$ of all $v \in \Gamma(u)$. If there is none, h is set to infinity.
2. The algorithm commits the move to v^* that has the best value $h(v) + w(u,v)$ of all $v \in \Gamma(u)$.

Korf (1990) has proven that *RTA** acts locally optimal on trees. In addition, he showed that *RTA** is complete, i.e. it finds a solution path if one exists. The bad news on *RTA** are detected when a goal is found such that the algorithm terminates. Some expanded nodes u may overestimate the distance to a goal, $h^*(u)$ for short, since their h-value are still set to the second best value of $h(v) + w(u,v)$ of all $v \in \Gamma(u)$. But a reinvocation of the algorithm is only possible if the improved h-values are admissible such that *RTA** has to be modified.

Learning Real Time A^* with Updating

If we invoke an algorithm several times, thereby improving the quality of the solution over time, then we say the algorithm learns. In our case we want to learn the heuristic function. One iteration in learning real time A^*– *LRTA** – differs only marginally from the algorithm *RTA**:

1. The heuristic value $h(u)$ is set to $min_{v \in \Gamma(u)}\{h(v) + w(u,v)\}$.
2. The algorithm commits the move to v^* that has the best value $h(v) + w(u,v)$ of all $v \in \Gamma(u)$.

Fig.1 illustrates that *LRTA** can get arbitrarily bad compared to *RTA**. We consider the uniformly weighted problem graph $G = (V,E)$ with $h(v_0) = k$, $h(v_1) = h(v_2) = 0$ and $h(v_3) = k$ and with $v_i \notin F$, $0 \leq i \leq 3$. Starting at v_1 or v_2 we need $O(k)$ expansions in *LRTA** until we are able to explore the rest of the graph outside the given nodes. We might call this a ping–pong effect. The algorithm *RTA** however finds the way in $O(1)$ because the second best value of $h(v) + w(u,v)$, $v \in \{v_1, v_2\}$ is $k + 1$.

On the other hand *LRTA** is able to learn since it keeps the heuristic estimates at each node admissible. Korf (1990) shows that the heuristic values in *LRTA** are converging eventually to the shortest path ones. But this convergence is not fast since the heuristic estimates are often bad. So *LRTA** has been extended in several ways. Barto et al. (1995) investigate a real time

dynamic programming approach. Their main focus are uncertainties and adaptive control strategies. Ishida and Shimbo (1996) introduce the ϵ and δ search algorithms which improve *LRTA** using cutoff strategies such as weighting and upper bounds. The following two strategies called *forward updating* and *backward updating* allow to improve the heuristic by updating all values involved in an expansion. The invariant kept at the nodes is that h is admissible.

Lemma 1 *(Forward Updating) Let* $u \in Q - F$, $h(u) \leq h^*(u)$ *and for all* $v \in \Gamma(u)$ *let* $h(v) \leq h^*(v)$. *After setting* $h(v)$ *to* $\max\{h(v), h(u) - w(u,v)\}$ *for all* $v \in \Gamma(u)$, $h(v) \leq h^*(v)$ *still holds.*

Proof: If $h(v) \geq h(u,v) - w(u,v)$ we have nothing to prove. Otherwise $h(v)$ is set to $h(u) - w(u,v)$. Suppose that $h(v) > h^*(v)$ and let p be the shortest path corresponding to the optimal cost $h^*(v)$ and extend p to p' by adding the edge $\{u,v\}$. Then we have $w(p') = h^*(v) + w(u,v) > h(v) + w(u,v) = h(u)$. But this contradicts $h(u) \leq h^*(u)$. □

Lemma 2 *(Backward Updating) Let* $u \in Q - F$, $h(u) \leq h^*(u)$ *and for all* $v \in \Gamma(u)$ *let* $h(v) \leq h^*(v)$. *Setting* $h(u)$ *to* $\max\{h(u), min_{v\in\Gamma(u)}\{h(v) + w(u,v)\}\}$ *implies* $h(u) \leq h^*(u)$.

Proof: If $h(u) \geq \min_{v\in\Gamma(u)}\{h(v) + w(u,v)\}$ there is nothing to show. Every path p from u to F has to pass one of u's successor nodes $v \in \Gamma(u)$ such that

$$w(p) = w(p')+w(u,v) \geq h^*(v)+w(u,v) \geq h(v)+w(u,v).$$

This is especially true if p is the shortest path with length $h^*(u)$. Thus we have $h^*(u) \geq \min_{v\in\Gamma(u)}\{h(v) + w(u,v)\}$ to which $h(u)$ is set. □

Note, that the proof of Lemma 2 establishes the correctness of *LRTA** and that the new idea is to take $h(u)$ itself into account. We have established the following theorem.

Theorem 1 *In* LRTA* *with forward and backward updating the heuristic evaluation function h keeps admissible.*

Note that h can be interpreted as a lower bound of the shortest path from a given node to the set of goal nodes. Defining an upper bound function h' for the shortest path to a goal node the dual forward updating

$$h'(v) \leftarrow \min\{h'(v), h'(u) - w(u,v)\}$$

and/or backward updating

$$h'(u) \leftarrow \min\{h'(u), max_{v\in\Gamma(u)}\{h'(v) + w(u,v)\}\}$$

assignments keep h' as an upper bound. One way to establish better bounds is to look several steps ahead and propagate the results of the horizon back to the root. This lead to the so-called Minimin scheme introduced by Korf (1990). Another one is described in the following section.

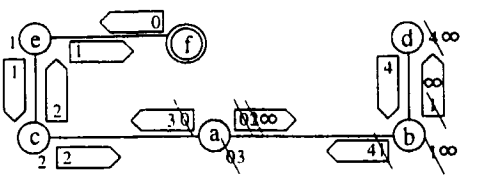

Figure 2: Moving from a to b and back to a in *SRTA**.

The Algorithms *CRTA** and *SLRTA**

Our idea for improvement is based on *signs* that are assigned to each edge in the state space. Each sign is seen as a lower bound for the shortest path passing this edge. We interpret the signs as a mapping f from the set of edges in the problem graph to $\mathbb{R}^+$. Since the shortest path has no cycles the lower bound values need to hold for all cycle free paths only.

We define the *expanding path* p as the committed path from the start to the detected goal node. Every node on p has been expanded at least once. If we assume that for all states u and for all $v \in \Gamma(u)$ the value $f(u,v)$ is set to the initial heuristic estimate $h(u)$ the following algorithm *SRTA** (real-time A^* using signs) is similar to the *RTA** procedure

1. If $u \in F$ then terminate and return u.
2. For all $v \in \Gamma(u) - F$ set $h(v)$ to the second best value $f(v,w)$ of all $(v,w) \in E$. If there is none, $h(v)$ is set to infinity. Further on, set $f(u,v)$ to $h(v) + w(u,v)$.
3. Compute v^* with $f(u,v^*) = \min_{v\in\Gamma(u)}\{h(v) + w(u,v)\}$.
4. return *SRTA**(v^*)

The initial values $f(u,v)$ do not have to be calculated in advance. They can be determined when v is generated for the first time. In Fig.2 a simple example of the *SRTA** procedure is provided. The algorithm is invoked with the nodes a,b and a and the changes of the h and f values are shown. The similarity to *RTA** is easily obtained since the calculated h-values are set with a lookahead of one. Moreover, if one does not change the h–values of a newly generated node then the h values in *RTA** and *SRTA** are the same. The algorithm uses the h values only in one call and does not need them further on. The following result proves the correctness of the f-values in the algorithm *SRTA**.

Theorem 2 *The calculated f values are correct, i.e.. they underestimate the length of the cycle free path to the goal.*

Proof: The proof is done uses induction, i.e. we use the stated result as an invariant (I).

It is obvious that the result is true after initializing since h is admissible. After the first invocation of *SRTA** with node u, the invariant (I) is also fulfilled since $h(v)$ equals the initial heuristic value and the edge (u, v) has to be traversed on a path from u to a goal node passing v. Suppose (I) is valid before invoking *SRTA**(u). We distinguish the following two cases.

The successor v of u is not expanded up to now. Then $h(v)$ is equal to the initial admissible estimate and the $f(u, v)$ values are correct.

If v has already been expanded then $h(v)$ denotes the best alternative to the path on which v has been left the last time. Since $f(u, v)$ is defined as a lower bound for all cycle free paths from u to a goal node using the edge (u, v), the values $f(u, v) = h(v) + w(u, v)$ are correct since v has already been left at the edge with the best f value. □

As an immediate consequence for the definition of signs we have

Theorem 3 *Define*

$$h'(u) \leftarrow \min_{v \in \Gamma(u)} \{f(u, v)\}.$$

Then $h'(u)$ is admissible.

See Fig.2 to verify that $h'(b) = 4$ is optimistic and $h(b) = \infty$ is not. This theorem leads to two new learning algorithms: *CRTA** (*RTA** with cleaning up strategy) and *SLRTA** (*LRTA** using signs).

We describe *CRTA** first. The algorithm traverses the expanding path $p = (s = v_0, \ldots, v_k)$ to a goal node $v_k \in F$ in reverse order to clean up over–estimations of h, i.e. values for which $h(v_i) > \min_{v \in \Gamma(v_i)}\{f(v_i, v)\}$. We simply set $h(v_i)$ to $\min_{v \in \Gamma(v_i)}\{f(v_i, v)\}$ for $i \in \{k-1, \ldots, 0\}$ and use Theorem 3.

Observe that the heuristic values of *CRTA** can be slightly improved by updating the f values in the backward traversal. We set $f(v_i, v_{i+1})$ to the new value $h(v_{i+1})$ plus the weight $w(v_i, v_{i+1})$ and determine $h(v_i)$ as the minimum of the new f-values. We gain that $h(v_i)$ is admissible using a simple inductive argument. In this approach the f-values are not needed to be stored explicitly. We get the similar heuristic estimates in applying *RTA** with the the dynamic backward updates of $h(v_i) = \min\{h(v_i), h(v_{i+1}) + w(v_i, v_{i+1})\}$.

The algorithm *CRTA** may be criticized since its clean up phase traverses the whole of the expanding path. Consider the robot navigation situation. First, we see that the backward traversal will be faster than expanding the path since the robot knows where to go. Further on, it can be modeled that the robot has to inform the person who had started it. In the following section we will examine how to handle cycles in the backward traversal of the expanding path.

Another aproach is to postpone the updating request until the node is revisited. The algorithm *SLRTA** uses iteration numbers at each node to distinguish the newly generated nodes in the actual iteration from already expanded nodes in former iterations. The idea is simple. Instead of recalculating the heuristic values on the expanding path after the current search iteration is terminated we perform the calculation at the time when the node is generated again. Using the iteration number we may encounter a node that has been expanded in some former iteration. The request at node u to reset the h values can now be satisfied setting $h(u)$ to $\min_{v \in \Gamma(u)}\{f(u, v)\}$. After that we can initialize the f values for node u in the current iteration by setting all values $f(u, v)$ to $h(u)$. The algorithm *SLRTA** is correct since the outgoing f-values at each node are modified only if u is expanded. There are two main drawbacks of *SLRTA**. First, the iteration number cannot be bounded by a constant and second, *SLRTA** cannot be improved by changing the f values of the expanding path as in *CRTA**.

In both algorithms the admissible estimate h' does converge to the shortest path values using a similar argument as in Korf (1990) for proving the convergence of *LRTA**. The algorithms can be modified to learn upper values for the shortest path.

Balancing the Search Tree with *BCRTA**

The algorithm *CRTA** allows us to use improved heuristic values compared to *LRTA**. If graph traversing implies robot motion as in the model above the proposed backward traversal is expensive. Let us see now how this updating costs can be reduced to a minimum size preserving the possibility for locally optimal decisions in next iteration.

For the sake of simplicity, let us assume that the problem graph is acyclic. Let T be the set of generated nodes. For each $x \in T$ the set T can be structured in a tree-like way rooted at x denoted by T_x. The tree structure is defined recursively by the predecessor function *pred*. The value *pred*(x) is empty, since x has no predecessor, and *pred*(u) is the last node before u on the unique path from x to u. Let $T_{x,u}$ be the subtree of T_x rooted at u.

Let us start *RTA** at root node x. If $h(u)$ is set to the second best value as done in *RTA**$h(u)$ is always the minimum cost value over all frontier nodes below u in the tree $T_{v,u}$ where v is the current position (see

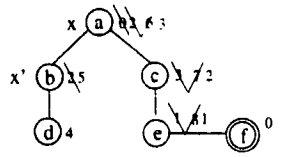

Figure 3: The exploration and update of T_a.

Korf (1990)). But what can we conclude for the tree T_x rooted at $x \neq v$?

Lemma 3 *Let* RTA* *be applied to an acyclic problem graph G. Let x be the root of the current search tree and let v be the current position. Then for each u which is not ancestor of v in T_x we have*

$$h(u) = min\{w(u,u') + h(u') | u' \in \Gamma(u) - \{pred(u)\}\}.$$

As a result: For each u which is not predecessor of the current state v the value $h(u)$ is the minimum of all values $g_u(y) + h(y)$ where y is a leaf of the subtree $T_{x,u} \subset T_x$ and $g_u(y)$ is the weight of the path from u to y.

Proof: Korf (1990) has shown that $h(u)$ is always the minimum over all frontier nodes below u in the tree rooted at v if v is the current position. Let u be a node which is not a predecessor of the root x. Then the proof follows immediately from the fact that $T_{x,u}$ is equal to $T_{v,u}$. □

This fact allows to reduce the effort of the last update step. It suffices to traverse the states from the goal node g to root node x to clean up the heuristic estimates related to x. By traversing back from g to x for each u on the solution path we set $h(u)$ by $h(u) = min\{w(u,u') + h(u') | u' \in \Gamma(u) - pred(u)\}\}$, where $pred(u)$ denotes the predecessor of u in tree T_x. Then, after the last update step and with termination of the search Lemma 3 is true for all nodes in tree T_x.

Fig.3 illustrates the exploration and updating of the search tree T_a in the example of Fig.2. If the problem graph is not acyclic then the update process is more complicated and can not be done as efficiently as in the acyclic case. We omit the details.

In the next iteration move decisions can be made again locally optimal. Let us assume that the next iteration starts with expansion of x. The first move decision is locally optimal, since each neighbor of x in the graph is an immediate successor of x in tree T_x and the transition is done in direction to the best one. By going downwards in the tree the heuristic values of the ancestors of the current node are updated as before. If a predecessor value is better than the values of the successors the move decision is made in favor of the predecessor. Otherwise the next node to visit is one of the immediate successors. This shows that the update can be reduced to $\log(|T_x|)$ in the average case.

How can locally optimal decisions be preserved if the next iteration will start at a node $x' \neq x$? First, let us assume that x' is one of the already generated nodes, i.e. $x' \in T_x$ for one root node x. Then the update has to be extended by an additional traversal from x to x'. This step fulfills the precondition of Lemma 3 with respect to the new tree $T_{x'}$. If x' has not been generated before, i.e. $x' \notin T_x$ for each root x, then the described step cannot be carried out at the beginning. First the tree $T_{x'}$ is build up until a node $z \in T_x \cap T_{x'}$ is found. Then the additional step is performed by traversing back from x to z before the search continues.

The algorithms *RTA** and *CRTA** do not work that well if the heuristic function is pathologic (Korf (1990)). The reason is that a lot of transitions have to be done leading from one side to the other side in the search tree before the frontier node with minimal cost can be expanded next. For that reason it makes sense to take care of the distance between the current position and the frontier nodes to expand next. It seems useful to make it harder to traverse a long path for the next expansion of a frontier node. The idea is to minimize the number of transitions by the use of another criterion favoring an equally growing search tree. We assume that a second function value $n(u)$ is stored for each u counting the number of states behind u with respect to the current position v, i.e., $n(u) = |T_{v,u}|$. Then *BCRTA** (balanced *CRTA**) uses the following steps:

1. If u is a goal node then terminate.
2. If u is not expanded before then expand it.
3. Let u^* be the best and u^+ be the second best successor in $\Gamma(u)$ (ties are broken in favor of lower n-value). If $n(u^*) \leq \alpha \sum_{u' \in \Gamma(u)} n(u')$ then go to u^*; otherwise go to the second best successor u^+.
4. Update $h(u)$ as given above w.r.t the choice in 3. Update $n(u)$ by $n(u) = 1 + \sum_{u' \in \Gamma(u) - \{u^*\}} n(u')$ if the transition to u^* is done; otherwise set $n(u) = 1 + \sum_{u' \in \Gamma(u) - \{u^+\}} n(u')$.
5. Invoke *BCRTA**(u^*) resp. *BCRTA**(u^+)

The value α has to be well chosen via experimental results. Ping-pong effects as in Fig. 1 are excluded: If the condition in 3. is not fulfilled the first time it is not fulfilled until the next frontier node has been reached and expanded.

Learning real time search can also be based on *BCRTA**. Then the n-values have to be updated with a step similar to *CRTA**. This idea of dynamic balancing is used in a similar way in Eckerle (1996).

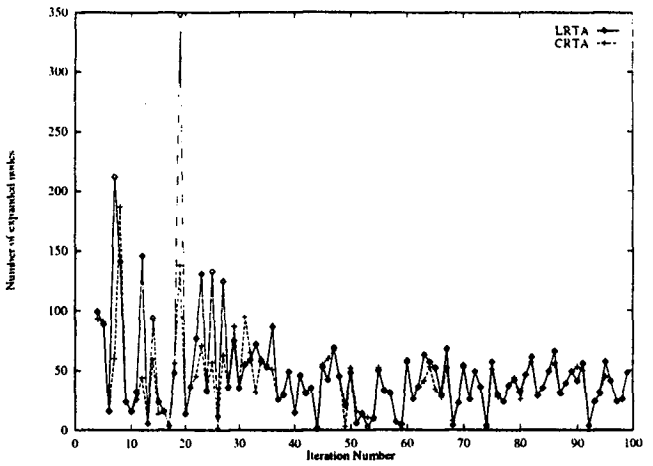

Figure 4: Comparing *LRTA** with *CRTA**.

Experimental Results

The problem space is a fixed *Maze* of size (30×30). A wall at square (i, j), $1 \leq i, j \leq 30$, is set with a probability of 35 percent. We examine the algorithm *CRTA** in the improved extension to the *RTA**, i.e., the $h(v_i)$ values in the backward traversal of the expanded path $p = (v_0, \ldots, v_k)$ are updated with $h(v_i) = \min\{h(v_i), h(v_{i+1}) + w(v_i, v_{i+1})\}$.

When the heuristic estimates exceed a given threshold of 100 we conclude that no goal can be found and no solution is returned.

Fig. 4 illustrates that the *CRTA** performs better than *LRTA** regarding the number of node expansions as we might have expected. The first values that have been omitted for the sake of a neat presentation are (214/166/2094) for *CRTA** and (364/260/4848) for *LRTA**. The basic fact to keep in mind is that the accumulated total number of expanded nodes at the beginning of *LRTA** is high (6801 for the first twenty iterations) compared to *CRTA** (3339 for the first twenty iterations). After a while the learning qualities in both algorithms settle to a fairly good approximation and do not differ much. The number of iterations is, of course, not sufficient to evaluate the learning quality. It is be more realistic to count the total number of expanded nodes so far.

There is a direct correspondence between the quality of the heuristic estimate and the number of expanded nodes. If the heuristic estimates are good the expanded path will be short and vice versa.

Conclusion and Outlook

A lot of new aspects for improving the efficiency of learning the heuristic functions have been studied in this paper. All provided algorithms act locally within a depth bound of one. The idea of signing the edges at each node has been proven to be useful in both providing more information and getting better insight into the problem structure.

Usually, the problem spaces of state space problems are assumed to be huge. This implies that not every node of the whole search tree can be stored in the hash table. Moreover, memory restrictions are seen to be more crucial to search algorithm than time restrictions. This has led to the design of improved heuristics and many memory bounded search algorithms.

In the algorithm *LRTA** and in its presented alternatives however, the hash table is extended in every iteration and thus will exceed the given space within a short time. This problem has not been tackled yet. It would also be interesting to compare off-line and on-line learning schemes. For example the robot may stroll and explore the environment without the request to find the goal fast and thus he has the chance to change the h values even in areas that are far from the set of goals. Thus we might ask for a good trade off between short and long time performance.

Acknowledgements S. Edelkamp is supported by a grant from DFG within the Freiburg University Graduate Program on *Human and Artificial Intelligence*, while J. Eckerle's contribution is supported by a grant from DFG within the project Ot 64/11-1.

References

Barto, A. G.; Bradtke, S. J.; and Singh. S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Eckerle, J. 1996. BDBIDA*: A new approach for space-limited bidirectional heuristic graph search. In *Proc. of ECAI-96*, 370–374.

Hart, P. E.; Nilsson, N. J.; and Raphael. B. 1968. A formal basis for heuristic determination of minimum path cost. *IEEE Trans. on SSC* 4:100.

Ishida, T., and Shimbo, M. 1996. Improving the learning efficiencies of realtime search. *Proc. of AAAI-96* 13(1):305–310.

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

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.