Developing an Optimal Learning Search Method for Networks

M.R. Zamani[1] and L.Y. Shue[1]

Among real-time search algorithms for networks which utilise heuristic estimates is the recent algorithm called Learning Real-Time A*(LRTA*). This learning algorithm continuously improves the initial heuristic estimates as the search process continues. It builds and updates a table containing heuristic estimates of actual distances to the goal state in the problem. Initially, the entries in this table correspond to the initial heuristic evaluation which is assumed to be a lowerbound on actual distance. Through repeated exploration of state-space, however, these estimates will lead to more accurate heuristic values. In this approach optimal solution cannot be found in a single trial and, moreover, updated heuristic values cannot be used in the same problem. However, starting from various states to go to the goal state, these updated heuristic values can be used. By incorporating a backtracking process to LRTA* and activating this process whenever an updating in the heuristic estimate of a state occurs, an algorithm called Learning and Backtracking A*(LBA*) is introduced. Contrary to LRTA*, in LBA* updated heuristic values can be used in the same problem in which these values are learned. The main theme of this algorithm is to guarantee improvement of the heuristic estimate of each state from which backtracking is done. Backtracking along with updating heuristic values helps LBA* to find the optimal solution using the circulation of updated heuristic estimates through the states. It is proven that under any circumstances the application of LBA* will lead to the optimal solution for any state-space problem in which heuristic estimates of states do not overestimate their actual values. The contribution that LBA* makes to the general idea of backtracking is that: assuming that all numbers used in a problem are integer, the number of backtracks in any state is limited to the amount which heuristic estimate of that individual state underestimates its actual value. This is something that, as yet, no backtracking algorithm has guaranteed.

INTRODUCTION

In this paper, a learning search technique is presented which finds optimal solutions for graph-search problems. This technique, LBA* (Learning and Backtracking A*), has been developed based on the incorporation of a backtracking process to an algorithm called LRTA* (Learning Real Time A*) developed by Korf [1].

LRTA* is an efficient real-time algorithm that guarantees neither optimality nor near-optimality of the solutions found. The reason that this algorithm is called real-time is that existing state-space search methods are divided into two classes of off-line and real-time algorithms. Off-line algorithms such as A* [2-4] and IDA* [5] examine the entire solution-tree and among all examined routes select the best

1. Department of Business Systems, Wollongong University, Wollongong, NSW 2522, Australia.

Scientia Iranica, Vol. 2, No. 3, © Sharif University of Technology, October 1995.

one. In other words, they compute an entire solution before executing the first step in the final path [6,7]. Real-time algorithms [1,8,9], such as LRTA*, on the other hand, perform just sufficient computation to determine each step independently until the goal state is reached. Therefore, real-time algorithms cannot find and guarantee optimal solutions. The topics of graph-search techniques [10,11,12], in general, and real-time and off-line algorithms, in particular, are of common interest for both operations research and artificial intelligence communities [6,13,14].

LRTA* is a graph-search algorithm which utilises initial heuristic estimates and continuously improves them as the search process continues. This algorithm updates heuristic estimates of states by comparing them with those of their neighbours in the process of search. The initial heuristic value of every state is an evaluation of the distance of that state from the goal state and is assumed not to overestimate the actual distance. Through repeated exploration of the state-space, however, these estimates will lead to more accurate heuristic values. In this approach the optimal solution cannot be found in a single trial and, moreover, updated heuristic values cannot be used in the same problem. However, while starting from various states to go to the goal state, these updated heuristic values can be utilised.

By incorporating a backtracking process into LRTA* and activating this process when updating the heuristic estimate of a state, the algorithm LBA* (Learning and Backtracking A*) is produced. Contrary to LRTA*, in LBA*, updated heuristic values can be used in the same problem. The main theme of this algorithm is to guarantee the improvement of the heuristic estimate of each state from which backtracking is done. Backtracking along with updating heuristic values helps LBA* to find a solution using the circulation of updated heuristic estimates through the states. It will be proven that the application of LBA* leads to an optimal solution for any state-space problem in which heuristic estimates of states do not overestimate their actual values. The contribution that LBA* makes to the general idea of backtracking is that: assuming that all numbers used in a problem are integers, the number of backtracks in any state is limited to the amount by which heuristic estimate of that individual state underestimates its actual value. This is something that, as yet, no backtracking algorithm has guaranteed.

Since LRTA* [1] is the starting point from which this search technique is developed, its functioning is briefly described.

LEARNING REAL-TIME A* (LRTA*)

Among the search algorithms with finite state-space and heuristic estimates for every state to the goal state, LRTA* represents a major research direction which takes into consideration the effect of learning in the search process. LRTA* builds and updates a table containing heuristic estimates of the cost (distance) from each state in the problem to the goal state. Initially, the entries in the table correspond to the initial heuristic evaluations which are assumed to be lowerbounds on actual costs. LRTA* manages to improve these entries during the search process making them more accurate estimates.

From the initial state, the search process starts by comparing its heuristic estimate with the "compound values" of all neighbouring states, where each "compound value" includes the estimate of the distance from the respective neighbouring state to the goal state and the edge cost from the current state to each neighbouring state. The neighbouring state with the minimum compound value is chosen for the next stage of expansion and the heuristic estimate of the current state is replaced with this value to reflect a more accurate estimate. This later part represents the updating mechanism of the Korf's algorithm which can be considered as a learning process.

Assuming that x is the current state of a search process, the LRTA* algorithm repeats the following steps until the goal state becomes the current state.

1. Calculate the compound value of $f(x') = h(x') + k(x, x')$ for each neighbour x' of the current state x, where $h(x')$ is the current heuristic estimate of the distance from x' to the goal state and $k(x, x')$ is the cost of the edge from x to x'.
2. Move to a neighbour with the minimum compound value, $f(x')$, and consider it as the current state.
3. Update the value of $h(x)$ to the minimum compound value of its neighbours.

The reason for updating the value of $h(x)$ is that since the compound value of $f(x')$ represents a lowerbound on the actual distance to the goal through each of the neighbours, the actual distance from the given state must be, at least, as large as the smallest of these compound values. The valuable contribution of this method to search techniques is the improvement of the heuristic estimates of states during the process of problem solving. These improved heuristic estimates can be used in following trials, when starting from other initial states and going to the same goal state.

In a finite problem space with positive edge costs, in which there exists a path from every state to the goal, this algorithm will always be able to reach the goal. Although there is no guarantee of optimality or near-optimality of the solutions produced by Korf's algorithm in any single problem-solving trial, the repeated trials will eventually adjust each heuristic estimate on the final path to its actual value and, hence, this leads to the optimum solution.

LEARNING AND BACKTRACKING A*(LBA*)

The fundamental feature of this algorithm (LBA*) is the repetitive application of the updating cycle which normally consists of a trial through searching, evaluating after the trial and updating through feedback. The major difference between this algorithm and Korf's algorithm lies in the implementation of a backtracking process which occurs when updating the heuristic estimation of any state.

LBA* starts with the initial state as the current state and changes the current state until a path to the goal state is found. To search for the next state for expansion, it calculates the compound values of all neighbouring states by adding the heuristic estimate of each neighbouring state and the edge cost from the current state to that neighbouring state. The neighbouring state with the minimum compound value is selected as the current state and computing the new compound values as well as comparing them for selection of the next current state continues. This process of selecting the neighbouring state with the minimum compound value continues until the heuristic estimate of the current state is less than the minimum of the compound values of its neighbouring states. In this case, the heuristic estimate of the current state is updated to this minimum compound value and the decision about selecting this current state to be on the path is revised by removing it from the path and backtracking to the previous state currently on the path.

The fact that a state is blocked for further expansion (dead-end) may also serve as an indication for the algorithm to learn not to enter the same state again in the future. The backtracking routine is initiated following the updating of the heuristic estimate of a state. Through this backtracking process, the current state leaves the path and the previous state on the path will become the current state. In other words, whenever the heuristic estimate of the current state is updated to the minimum compound value of its neighbouring states, this state will leave the path and the previous state on the path will become the current state. Then, again, re-examination of neighbouring states starts and it is likely that this re-examination will lead either to a change of direction for expansion or to the adjustment of its heuristic value and, hence, one more stage of backtracking. Depending on the original estimate of the initial state, the backtracking process may retreat all the way

back to the initial state as often as is needed in order to update its estimate before the final path is found. The rationale for this is that the inclusion of a state in the final path is promising only when its neighbours indicate so.

This algorithm can be implemented in the following manner:

STEP 0. Apply a heuristic function to generate a non-overestimating initial heuristic estimate $h(x)$ for the distance of every state x to the goal state and calculate the sum of all edge costs as an upper-bound U for the cost of reaching the goal state from the initial state.

STEP 1. Put the initial state on the backtrack list called OPEN.

STEP 2. Call the top-most state on the OPEN list x. If x is the goal state, stop.

STEP 3. If x is a dead-end state, replace its $h(x)$ with a very large value, remove x from the OPEN list and go to Step 2.

STEP 4. For every neighbouring state y of x, evaluate the compound value of $k(x,y) + h(y)$ and find the state with the minimum value. Call this state x'. Break ties randomly. $k(x,y)$ represents the positive edge cost from state x to state y.

STEP 5. If $h(x) \geqslant k(x,x') + h(x)$, then add x' to the OPEN list as the top-most state and go to Step 2. Otherwise, continue.

STEP 6. Replace $h(x)$ with $k(x,x') + h(x')$ and if $h(x) > U$, then print "goal state is not accessible from initial state", and stop.

STEP 7. If x is not the root state, remove x from the OPEN list.

STEP 8. go to Step 2.

With this algorithm, the memory requirement includes the storage of the most up-to-date heuristic value of each state and the OPEN list which at the end of the algorithm shows a path from the initial state to the goal state. The list keeps only the states on a path from the initial to the current state, and the current state is always on the top of the list. When the heuristic estimate of a current state is updated, backtracking is carried out by removing the current state from the list and letting its previous state become the current state as indicated in Step 7. Step 7 is carried out after Step 6 which updates the heuristic estimate of the current state while detecting the non-accessibility of the goal state from the initial state. The adjustment made by Step 6 is the result of the comparison of estimates with neighbours as indicated in Step 4 and the false condition of Step 5. Should the condition of Step 5 be true, the algorithm will continue to expand the current state without the need for backtracking. Step 3 is arbitrary and has been used to assign a large value to the heuristic estimate of a state that has been considered dead-end, ensuring no future visit to such states. There are many ways to define a dead-end state and here the following simple definition is proposed: "x is dead-end if it has only one neighbour and this neighbour is currently on the OPEN list".

The way in which the heuristic estimate of a state is adjusted, as indicated in Step 6, does ensure that the newly adjusted value raises the lowerbound and will never be greater than its actual value. As the search process continues, the heuristic estimates of states on the final path will finally converge to their actual values through the guidance of the edge costs.

A proof can be presented that the application of LBA* will lead to finding the optimal solution of problems. This proof is presented after showing the application of LBA* to the following simple problem.

An Example

To show how LBA* works, we apply it to a grid problem represented in Figure 1. This grid problem can be considered as a state-space with sixteen states (cells) on which operators with the cost of one operate and transform them to one of their neighbouring states (cells), provided that no highlight line (barrier) exists

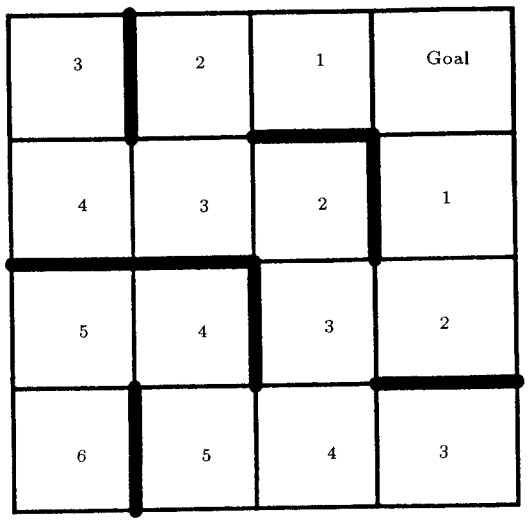

Figure 1. A grid problem (with its initial heuristic estimates).

between the state on which the operator acts and its neighbouring state. The left bottom cell is called state (1,1) which is the initial state and the top right cell is called state (4,4) which is the goal state. Obviously, the purpose of the application of LBA* is to find a series of operators with minimum cost to transform the initial state to the goal state. This can be stated by the simple fact that LBA* tries to find a path with minimum cost from the initial state to the goal state.

A heuristic estimate for every state is easily constructed based on the assumption of removing all vertical and horizontal barriers. Figure 1 shows these estimates as well as the barriers. Now all operations, in detail, are described.

At first, state (1,1), the left bottom cell, is put on the OPEN list as the initial state (Step1). Now the OPEN list has only one member which is the initial state. State (1,1) as the top-most state on the OPEN list is called x (Step2). All neighbouring states of x, in this case only state (2,1), are evaluated and among them state (2,1), itself, with the minimum compound value of $1+5$ is selected and is called x' (Step4). Notice that in the compound value of $1+5$ the value of 1, $k(x,y)$, is the edge cost from state (1,1) to state (2,1) and the value of 5, $h(y)$, is the current heuristic estimate of state (2,1). Since the heuristic estimate of x, 6, is not less than $1+5$, no updating occurs and x' is added to the top of the list. Similarly, state (2,2) is added. State (2,2) as the top-most state on the list is called x (Step2). All neighbouring states of x, (2,1) and (1,2), are evaluated and both result in the same compound value of $1+5$. The tie is broken randomly and state (2,1) with the compound value of $1+5$ is selected and is called x' (Step4). Since the heuristic estimate of x, which is now 4, is less than $1+5$, the old estimate is replaced with the compound value of $1+5$ and this state is removed from the list. The same process causes state (2,1), with the updated heuristic estimate of $1+6$, to be removed from the list. Now state (1,1) as the top-most state on the list is called x. All neighbouring states of x, in this case only state (2,1), are evaluated and among them state (2,1), itself, with the mininmum compound value of $1+7$, is selected and is called x' (Step4). Since heuristic estimate of x, 6, is less than the compound value of $1+7$, the old value is replaced with $1+7$. Notice that now x, state (1,1), is the initial state and based on Step7 when the initial state experiences learning, no backtracking happens. All neighbouring states of x are evaluated and among them state (2,1), with the mininum compound value of $1+7$, is selected and is called x'. Since the heuristic estimate of x, which is now 8, is not less than $1+7$, x' is added to the top of the list (Step5). Similarly, the states (2,2), (1,2), (1,3), (2,3) and (3,3) are added. Notice that after the selection of state (2,3), both states (3,3) and (2,4) had the same chance to be selected and that the tie was broken randomly.

Now state (3,3), as the top-most state on the list, experiences 2 units of learning and is removed from the list and state (2,3) is called x. All neighbouring states of x are evaluated and the compound values of $1+4$ and $1+2$ are obtained. The minimum which is associated with state (2,4) is selected and state (2,4) is added to the list. The same process causes states (3,4) and (4,4) to join the list; finally, since state (4,4) is the goal state, the algorithm stops (Step2).

After applying LBA* to this problem some heuristic estimates are updated and the optimal solution is found. Figure 2 shows updated

heuristic estimates and the optimal path found by LBA*.

As shown, the algorithm progressed from the initial state to the goal state by updating initial heuristic estimates during the process of search and activating a backtracking process whenever an updating occurred. Notice that even after finding the optimal path, still the heuristic estimates of some states are underestimated. For instance, the heuristic estimate of state (4,1), 3, which originally underestimated the actual distance by two units, still is the same, indicating that this state was not visited at all during the search. The same is true for state (1,4). However, the heuristic estimates of all the states on the optimal path have been improved to their actual distances. The point is that whenever a state is close to the optimal path, there is a better chance of its heuristic estimate being updated. This means that the learning process and the circulation of this "learned information" through activating a backtracking process has led to many states not to be visited even once and only the close neighbours to the final optimum path have been considered.

Now a proof is presented for the theorem that LBA* always finds the optimal solution of a problem.

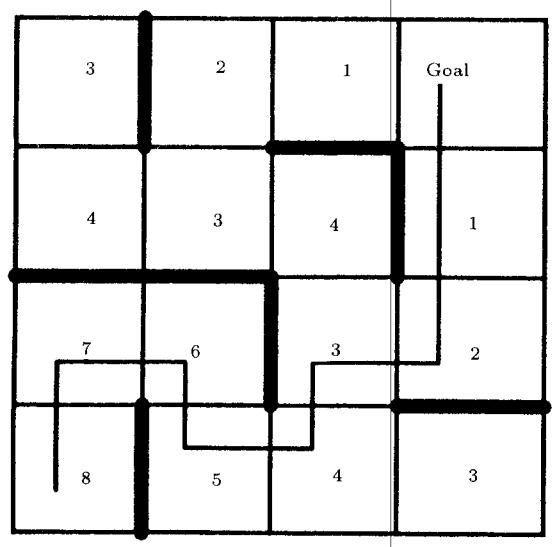

Figure 2. Updated heuristic estimates and the optimal path found by LBA*.

Theorem and Proof

Theorem 1.

For a finite problem space with positive edge cost and non-overestimating initial heuristic values, in which the goal state is reachable from the initial state, the application of LBA* will find the optimum path.

Proof:

Because of positive edge cost assumption, Step 5 prevents the creation of any loop, therefore, the algorithm either terminates or does an infinite number of backtracks. Considering the facts that with every backtrack the heuristic estimate of a state increases and that Step 6 keeps all heuristic estimates non-overestimated, it is concluded that the algorithm cannot do an infinite number of backtracks. Therefore, if the goal state is reachable from the initial state the algorithm terminates with finding a path, P, otherwise it stops in Step 6 when the heuristic estimate of a state exceeds the upper-bound U. It is assumed that the goal state is reachable from initial state, therefore, the algorithm terminates with finding a path P.

Now, suppose that there exists a path P' to the goal state with a smaller cost than that of the path P found by LBA*. Let the states on the path P be denoted as $x(1)$, $x(2), \ldots x(n)$ and the states on the path P' be denoted as $y(1)$, $y(2), \ldots y(q)$, where $x(1)$ and $y(1)$ are the same initial state, and $x(n)$ and $y(q)$ are the same goal state.

Let $y(m)$ be the first state of P' which is not on the path P. Thus, both $x(m)$ and $y(m)$ are the neighbours of their previous common state $x(m-1)$.

With k representing the edge cost from a state to a neighbouring one, the following relation must be satisfied by P' to produce a smaller cost than P.

$$k\{x(m-1), y(m)\} + k\{y(m), y(m+1)\} + \ldots + k\{y(q-1), y(q)\} < k\{x(m-1), x(m)\} + k\{x(m), x(m+1)\} + \cdots + k\{x(n-1), x(n)\} \; . \quad (1)$$

This relation can be further modified to State-

ment 2 because of the fact that the adjustment of a heuristic estimate in the algorithm will never lead to a value higher than its actual value.

$$k\{x(m-1), y(m)\} + h\{y(m)\} < k\{x(m-1), x(m)\} + k\{x(m), x(m+1)\} + \cdots + k\{x(n-1), x(n)\} \ . \quad (2)$$

As indicated by Steps 5 and 6 of the algorithm, the following relation is always true for a state $x(r)$ under LBA*.

$$h\{x(r)\} \geqslant k\{x(r), x(r+1)\} + h\{x(r+1)\} \ . \quad (3)$$

Statement 3 can be rearranged as:

$$h\{x(r)\} - h\{x(r+1)\} \geqslant k\{x(r), x(r+1)\} \ . \quad (4)$$

By expanding and summing the inequality in Statement 4 over the state space of path P from $x(m)$ to the goal state, the following relation is obtained:

$$h\{x(m)\} - h\{x(n)\} \geqslant k\{x(m), x(m+1)\} + k\{x(m+1), x(m+2)\} + \cdots + k\{x(n-1), x(n)\} \ . \quad (5)$$

With the estimate from the goal state to itself being zero, $h\{x(n)\} = 0$, Statement 5 becomes:

$$h\{x(m)\} \geqslant k\{x(m), x(m+1)\} + k\{x(m+1), x(m+2)\} + \cdots + k\{x(n-1), x(n)\} \ . \quad (6)$$

The fact that LBA*, at state $x(m-1)$, has preferred $x(m)$ to $y(m)$, as indicated by Step 4, also leads to the following relation,

$$k\{x(m-1), y(m)\} + h\{y(m)\} \geqslant k\{x(m-1), x(m)\} + h\{x(m)\} \ . \quad (7)$$

With the substitution of $h\{x(m)\}$ from Statement 6 into Statement 7, Statement 8 is obtained,

$$k\{x(m-1), y(m)\} + h\{y(m)\} \geqslant k\{x(m-1), x(m)\} + k\{x(m), x(m+1)\} + k\{x(m+1), x(m+2)\} + \cdots + k\{x(n-1), x(n)\} \ . \quad (8)$$

It is obvious that Statement 2, which is the result of the earlier assumption that P' is a better path than P, contradicts Statement 8, which is the result of the actual application of LBA*. Thus, P' does not exist and the application of LBA* will always lead to finding the minimum path.

Efficiency of LBA*

The fact that LBA* needs to store the heuristic estimate for every state of a problem has led to the upperbound for space complexity of LBA* being n, where n is the number of states in the problem. In practice, however, the complexity can be lower because usually a function exists which computes the original heuristic estimates, and it is necessary to store in memory only those values which differ from the computed ones. For instance, in the grid problem of Figure 3, the function to compute the distance

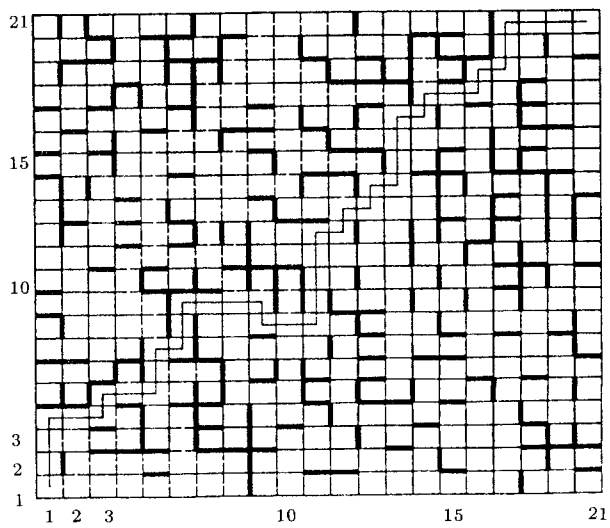

Figure 3. A sample grid problem and its optimal path found by LBA*.

from cell (i, j) to the goal state, cell $(21, 21)$, is $(2^{*}21 - i - j)$. The total number of states in this problem is $21^{*}21 = 441$, however, the algorithm reaches the optimal path by updating only 140 of them. Therefore, the heuristic estimates of 301 states can be computed rather than being retrieved from the memory.

To derive the time complexity for LBA*, it is assumed that all numbers used in the problem are positive integers. With this assumption, the worst-case time complexity of LBA* is $n.s$, where n is the total number of states and s is the final cost from the initial state to the goal state. This worst-case may happen when the initial heuristic estimates for all states are zero (i.e. there is no information at all) and the edge cost from every state to its neighbours is assumed to be one. The latter assumption will lead to only one unit increment in each updating process. Thus, the $n.s$ figure is the worst-case since every state has to be visited s times.

In reality, the actual worst-case will only be a portion of the figure $n.s$, because all states except the initial state will have a smaller "true" cost to the goal state than s. For most problems, where initial information for heuristic estimates is obtainable and edge costs vary from one state to another, the number of visits required before the final path is found could be very low.

Figures 4 and 5 demonstrate (using the example in Figure 3) the relationship between the amount of underestimate and the number of backtracks for every cell. In number terms, total amount of underestimate over all the cells and the total number of backtracks are 1324 and 119, respectively. Comparison of these two numbers reveals that Figure 4, on average, is ten times higher than Figure 5. Besides, it can be seen in Figure 5 that very few cells were subjected to backtracks, and the only cells with notable backtracks are located in the neighbourhood of the optimal path. For instance, cells $(21, 17)$, $(21, 16)$ and $(21, 15)$, which have the largest heuristic underestimates in Figure 4, have been backtracked infrequently compared to their heuristic underestimates.

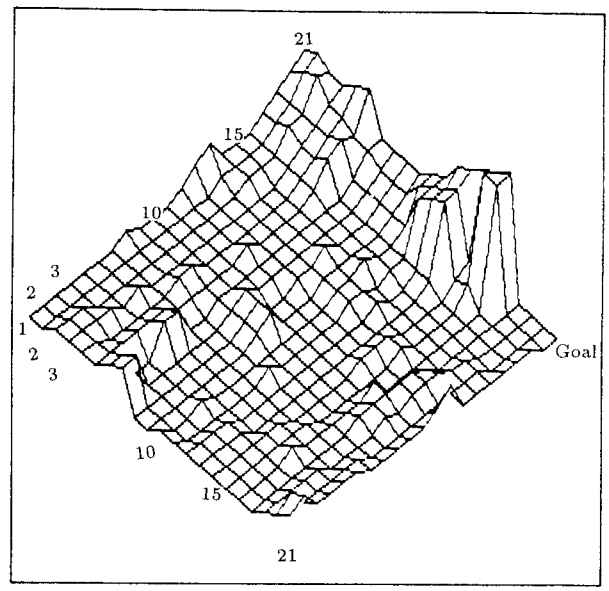

Figure 4. A 3-D representation of the initial heuristic underestimate of every cell in the sample grid problem.

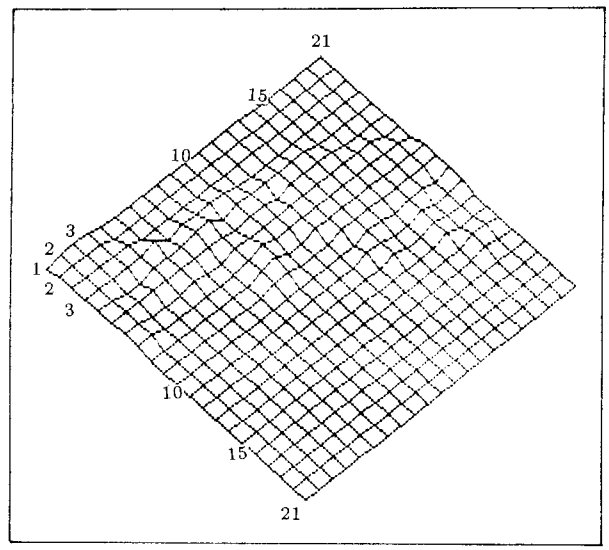

Figure 5. A 3-D representation of the number of backtracks done by LBA* in every cell of the sample grid problem.

The application of LBA* and the continuous application of LRTA* are compared in order to find the optimal solutions for these grid problems with different structures.

Over 20 square grid problems were tested and LBA* consistently outperformed LRTA*. LBA* found all optimum solutions in a single

Table 1. A comparison between performances of LBA* and LRTA* on 20 sample grid problems.

Instance			LBA*		LRTA*		Comparison
No	Size	Barrier Percent	Number of Cells Visited	Average Update per Visit	Number of Cells Visited	Average Update per Visit	Ratio of LBA* to LRTA* Speed
1	10	15	26	0.30	98	0.08	3.7
2	10	25	40	0.55	208	0.13	5.2
3	10	35	44	0.59	278	0.09	6.3
4	10	45	54	0.66	270	0.20	5.0
5	15	15	42	0.33	182	0.07	4.3
6	15	25	60	0.53	236	0.28	3.9
7	15	35	66	0.57	276	0.28	4.1
8	15	45	70	0.60	434	0.22	6.2
9	20	15	62	0.38	290	0.08	4.6
10	20	25	56	0.32	272	0.16	4.8
11	20	35	148	0.74	1310	0.36	8.8
12	20	45	430	0.90	2794	0.41	6.3
13	25	15	114	0.57	806	0.10	7.0
14	25	25	136	0.72	1104	0.13	8.1
15	25	35	148	0.74	1311	0.36	8.8
16	25	45	327	0.85	4730	0.69	14.4
17	30	15	138	0.57	1390	0.08	10.0
18	30	25	194	0.70	3374	0.05	17.0
19	30	35	426	0.87	7876	0.09	18.4
20	30	45	319	0.81	9180	0.67	28.7

problem solving trial, where LRTA* required more than one trial as expected. With 5 different square sizes 10, 15, 20, 25 and 30, and 4 different barrier percentages 15%, 25%, 35% and 45%, the ratios of states visited to find the optimal solution by LRTA* to that by LBA* range from 3.7 to 28.7. The trend is clear that, in general, the larger the size of a problem, the larger the ratio will be. Table 1 shows this comparison in detail.

This table consists of 8 columns. Columns 1, 2 and 3 show information about tested instances including size and the barrier percent. Columns 4 and 5 indicate how LBA* has functioned; and columns 6 and 7 show the functioning of the continuous application of LRTA* to find optimum solutions. In column 8 it has been shown how much faster the optimal solution of an instance has been found by LBA*. Notice that any backtracking, as well as any forwarding, causes a cell to be visited and, hence, the total number of visits in columns 4 and 6 indicates the total number of backtracks plus the total number of forwarding steps.

CONCLUSION

The main feature of LBA* is the circulation of updated heuristic estimates during the search process by utilising a backtracking routine

which causes the optimal solution of a problem to be found. Contrary to LRTA*, in LBA* updated heuristic values can be used in the same problem in which these values are learned. LBA* guarantees the improvement of the heuristic estimate of each state from which backtracking is done, since it starts with non-overestimating heuristic values and always keeps these values non-overestimating.

Over random grid problems tested, LBA* consistently outperformed LRTA* with performance ratios between 3.7 to 28. An important point was that, in general, the larger the size of the problem, the larger this ratio.

REFERENCES

1. Korf, R.E. "Real-time heuristic search", *J. Artificial Intelligence*, **42**, pp 189-211 (1990).
2. Gheoweth, S.V. and Davis, H.W. "High performance A* search using rapidly growing heuristics", *Proc. International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp 198-203 (1991).
3. Nillson, N.J. *Principles of Artificial Intelligence*, Tioga, Palo Alto, California, USA (1980).
4. Pearl, J. "Heuristic: intelligent search strategies for computer problem solving", Addison-Wesley, Massachusetts, USA (1984).
5. Korf, R.E. "An optimal admissible tree search", *J. Artificial Intelligence*, **27**, pp 97-100 (1985).
6. Korf, R.E. *Learning to Solve Problems by Searching for Macro Operators*, Pittman Advanced Publishing Program, Boston, USA (1985).
7. Laveen, K. and Kumar, V. *Search in Artificial Intelligence*, Springer-Verlag, New York, USA (1988).
8. Korf, R.E. "Planning as search: A quantitive approach", *J. Artificial Intelligence*, **36**, pp 201-218 (1987).
9. Korf, R.E. and Ishida, T. "Moving target search", *Proc. International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp 204-210 (1991).
10. Brown Donald, E. and White, C.C. *Operations Research and Artificial Intelligence*, Klumer Academic Press, Massachusetts, USA (1990).
11. Kingston J.H. *Algorithms and Data Structure: Design Correctness Analysis*, Addison-Wesley, Sydney, Australia (1990).
12. Simon, H. "Two heads are better than one: The collaboration between artificial intelligence and operations research", *J. OR Interface*, **17** (1), pp 8-15 (1987).
13. Brook, R.A. "Intelligent without reason", *Proc., International Joint Conference on Artificial Intelligence*, Sydney, Australia, pp 569-595 (1991).
14. Firebaugh and Morris, W. *Artificial Intelligence: A Knowledge Based Approach*, Boyd & Fraser, Boston, USA (1988).