New Approaches To Moving Target Search

Stan Melax
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
melax@cs.ualberta.ca

Abstract

New methods for doing moving target search are presented. One algorithm, forgetful depth-first search, attempts to adapt the well-known depth-first algorithm to this problem domain. Also, a search technique called marking quickly acquires general knowledge about the search space. These methods are discussed and compared with other known methods. Experimental results show that forgetful depth-first search and marking give good performance.

1 Introduction

The moving target search problem and solutions to it were initially introduced by Korf and Ishida in their paper "Moving Target Search"[1]. Ishida followed up this work in "Moving Target Search With Intelligence"[2]. The problem is a variation of the common search problem. There is a graph with a problem solver looking for the goal node. What is different in this problem is that the goal node, which we will refer to as the target, also moves. Each entity knows the current position (node) of the other. Moves can only be made to adjacent nodes. What makes the problem interesting is that the solver must make its decisions in a small constant amount of time. To guarantee that it is possible to complete the search, the target cannot always make optimal moves away from the solver. This condition could be satisfied with a slower moving target. However the search problem is more interesting when the solver cannot depend on the target's speed being slower than its own. Instead the target makes "mistakes", such as occasionally making moves "toward" the solver.

Although the search problem and the algorithms that are discussed are applicable to any graph, the examples and experimentation presented in this paper uses grid type graphs with some of the nodes blocked-out to prevent passage. Such graphs allow for much variation and are commonly used for many search problems.

With a moving target, heuristic functions take on the form $h(a, b)$ -the predicted distance between states a and b. The research presented here generally assumes the existence of a reasonably good and admissible heuristic function. For the grid class of graphs Manhattan distance is used. The Euclidean distance can be used to break ties when multiple neighboring nodes have the same lowest heuristic.

One simple moving target search algorithm is for the solver to get on the target's trail and follow it. This only works for "slower" targets. At the opposite end of the spectrum is Floyd's algorithm[3] which computes $h^*()$ in $O(n^3)$ time. This algorithm can be revised to work by increasing the working matrix (starting from 0 or initial heuristic). This makes the intermediate computations useful (admissible). After this knowledge has been computed, the problem solver just follows the direct path to the target.

This paper reviews the work of Korf and Ishida, introduces forgetful depth-first search and marking, and analyzes (with experimentation) the various algorithms. This paper shows that the new methods perform very well.

2 Other Work

2.1 Korf and Ishida '91

In the paper "Moving Target Search"[1], Korf and Ishida introduce this search problem. In addition, they present an algorithm by which a problem solver can find the target.

2.1.1 Original Moving Target Search Algorithm

Their basic moving target search algorithm (BMTS) is an extension of the trivial "greedy" algorithm in which the problem solver always moves to adjacent node with lowest heuristic value (ties broken randomly). The reason it does not get stuck forever in a "local minimum" (Figure 1) is that it modifies its

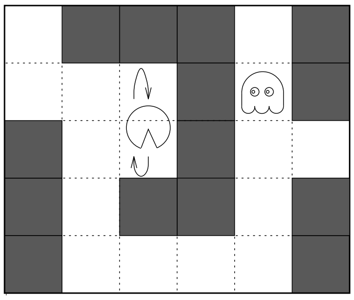

Figure 1: Local minimum (heuristic depression)

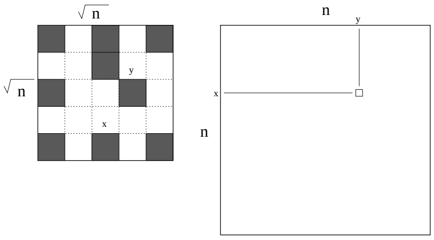

Figure 2: All pairs shortest path matrix

heuristic information using simple deduction. When the solver is at a non-goal node s_1 its true heuristic value (with respect to the target's node t) must be at least one greater than the least of its neighbors s_2. Therefore, when the actual heuristic does not reflect this,it is updated $(h(s_1,t) = h(s_2,t) + 1)$. The solver does this each time it moves. Furthermore, the target's moves (from t_1 to t_2) are monitored. If the heuristic value changes by more than 1 $(h(s,t_1) - h(s,t_2) > 1)$ a similar heuristic update is made. The problem solver builds an n by n matrix whose entries $h(x,y)$ represent a lower bound on the shortest path length between nodes x and y (Figure 2). This representation of knowledge is the same as that used by Floyd.

2.1.2 Heuristic Depressions

The BMTS problem solver has great difficulty with heuristic depressions. Figure 3 shows how the problem solver typically overcomes a misinformed heuristic situation such as a local minimum. The target is sitting still just on the other side of the wall at node t_1. The problem solver will move back and forth along the other side of the wall increasing the heuristic function $h(X,t_1)$ for states X on the problem solver's side of the wall. $O(k^2)$ moves are required to get out of a heuristic depression of k nodes. This inefficient "thrashing" behavior occurs because the solver increments heuristic information (usually by 1 or 2) one node at a time. Figure 3 also illustrates another problem. If just before the solver moves around the barrier the target moves from state t_1 to t_2, then the

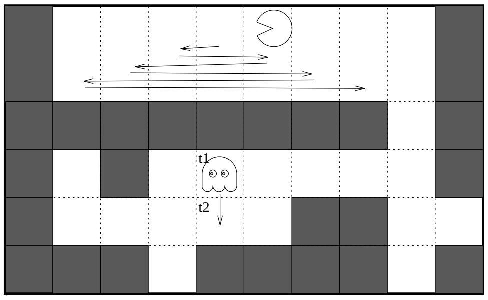

Figure 3: Adapting the heuristic function

solver will have to repeat the learning that previously took place. This second problem is characterized as "loss of information" due to target's movement. In a follow up paper, discussed later, Ishida addresses these problems.

2.1.3 Analysis

BMTS is complete under the assumption that the target makes mistakes. The solver keeps increasing his heuristic function (matrix) which, of course, cannot increase past the all-pairs shortest path matrix. Learning also depends on the rate in which the target makes "mistakes". The worst case is $O(n^3 T_{err})$ where T_{err} is the period of error (number of moves per mistake that the target makes).

BMTS differs from Floyd's[3] because it concentrates its learning on correcting the heuristic values that pertain to the current positions of the target and the problem solver. In practice most of the heuristic values are never used. Experimentation shows that actual search times are significantly better than the worst case.

2.1.4 Completeness Questioned

The proof of completeness for BMTS relies on the assumption that occasionally the target moves in such a way that the heuristic distance between the problem solver and the target does not increase. Given this, it seems reasonable for one to conclude that this criterion would be satisfied if they use a target that, instead of occasionally skipping moves, would move semi-randomly or occasionally make moves that common sense dictates are "toward" the problem solver. However this is not so. In other words, the target can move in such a way that reduces both the Manhattan distance (initial heuristic) and the shortest path distance (optimal heuristic h*()) but, according to the problem solver's current heuristic information, the heuristic disparity increases. Although this sounds possible, it seems likely that a target's random move should have a good if not equal chance of decreasing heuristic disparity for any given heuristic information. Unfortunately this is not the case. Experimentation discovered a situation in which BMTS

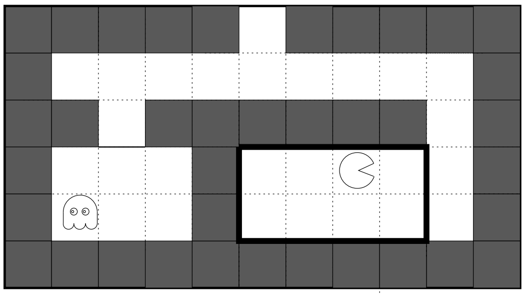

Figure 4: Stuck

builds up its heuristic information such that any legal move to an adjacent node that the target makes will increase the heuristic disparity. Furthermore this is not an unreasonable or isolated setup.

Figure 4 shows a situation where the solver is never able to work its way out of a heuristic depression. The target is programmed to move randomly but only stay within the 3 by 2 area. Even after a very long time the problem solver never moves out of the heuristic depression marked by the rectangle. The solver spends most of its time toward the left side of the depression. In general, the size of a heuristic depression that a solver can tackle does depend on the size of the area that the target moves about.

Note that this problem only applies to bipartite search spaces (no odd cycles). Possible remedies include using a slower target (in which case just following the target is better) or not using this class of search spaces. Fortunately, the commitment enhancement introduced by Ishida (next section) will restore completeness to searches with random targets that do not skip moves.

2.2 Ishida '92

In "Moving Target Search With Intelligence"[2], Ishida presents improvements to his and Korf's original algorithm BMTS. Ishida introduces commitment and deliberation to reduce the "information loss" and "thrashing" problems discussed in the previous section.

2.2.1 Commitment

In BMTS the solver learns with respect to the target's current position. If the target did not move then the solver could concentrate its learning and get out of heuristic depressions sooner. Ishida achieves this effect using commitment. The solver ignores some of the target's moves, and instead concentrates on a "goal" state which is occasionally updated to the target's current position.

This goal state is updated whenever the solver reaches it or when the solver makes a number of consecutive moves toward it. This parameter (the number of moves toward the goal) is called the degree of commitment. The ideal value for the degree of commitment will depend on the search space. By experimentation on 100 by 100 search spaces with varying densities, Ishida finds 10 to be a good value. Ishida shows that for the more difficult search spaces, (those with 35 percent of the edges removed,) the performance improves by 5 to 10 times over the original algorithm.

2.2.2 Deliberation

Even with commitment, the problem solver still "thrashes" inside depressions as it incrementally updates its heuristic information. Ishida introduces a mechanism called deliberation so that when the solver enters a depression it switches to an off-line search (similar to A^*) in order to expand its search in all directions. To maintain the real-time constraint, the solver can only expand one node in off-line search per regular move.

The solver limits the time spent deliberating to a maximum number of nodes (called the degree of deliberation), unless it finds a way out of the depression sooner. For easy search spaces the use of much deliberation is inappropriate since it allows the target to get away as the solver remains stationary. Ishida's experiment with the value of 25 nodes seemed to be the most promising with the more difficult search spaces. The performance doubled in comparison to using commitment alone. With respect to the original algorithm the improvement is between 10 and 20 times.

2.2.3 Overall Learning

Commitment and deliberation significantly improve BMTS. However even with these revisions, after the problem solver's immediate goal is satisfied, the learning that has taken place is not likely to be beneficial again. For example consider a long grueling search where the problem solver learns its way out of a heuristic depression but later finds itself in the same place. If at this time the target is in a different position from where it was during the first time that this happened, the problem solver will have to learn its way out of this heuristic depression once again. In short, learning seems to get lost in the huge n by n heuristic matrix representing the problem solver's knowledge. The essential reasoning process of Ishida's revised problem solver remains the same as its predecessor. Its knowledge domain still deals with the collection of pairs of states. Ishida's improvements are in the efficiency of the algorithm. Learning is concentrated and localized, not generalized.

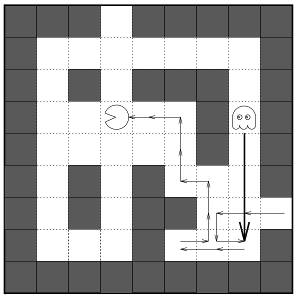

Figure 5: Target wanders onto searched nodes

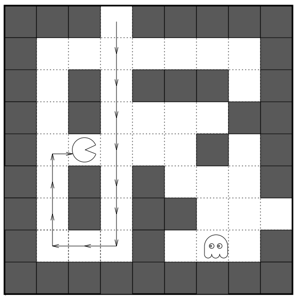

Figure 6: Closed list inhibits movement

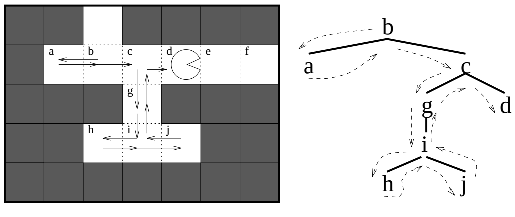

Figure 7: Problem solver's path and implicit DFS tree

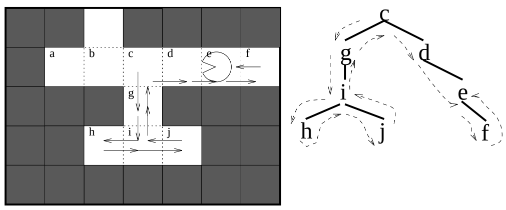

Figure 8: Same search 3 moves later (with the first 3 moves removed)

3 Forgetful Depth-First Search

This paper presents an algorithm with a radically different paradigm for searching for the moving target. The objective is to capture the way depth-first search (DFS) efficiently searches areas and quickly gets out of heuristic depressions.

3.1 Problems with Traditional DFS

Unfortunately, using normal depth-first search to hunt a moving target would not be an effective strategy. One problem is that old information becomes invalid. Nodes that have already been searched (those on the solver's closed list) are supposed to be places where the target is not. However this is not true; the target may move onto these nodes after they are searched (Figure 5).

Another problem is that the closed list constructed by the problem solver inhibits movement. This could block efficient paths to the target (Figure 6).

3.2 Forgetful Algorithm

These setbacks are overcome using an algorithm called forgetful depth-first search (FDFS). The solver has limited memory and cannot support a continually growing closed list. When the solver moves onto a node it will add it to his data structure. This means that some previous piece of information must go. The victim is the oldest piece of information, which is the root of the search tree. The new root of the search tree is the first child of the previous root. This pruning occurs once for each move the solver makes whether advancing onto a fresh node or backtracking over old ones. Figures 7 and 8 show the problem solver's path and the changes in the DFS search tree as the search progresses.

3.3 Implementation

Such an algorithm seems complex and raises doubt as to whether it can be done in real time. Fortunately there is a simple implementation. The solver keeps track of a list of visited nodes that allows duplicates. Whenever the solver moves he adds the new node onto the front of the list and removes the oldest node at the back of the list. When the solver is looking at its neighboring nodes to determine which to move to, it selects one that is not currently in its list if possible. If the problem solver cannot move to a node not on its list, it moves to the predecessor of the oldest occurrence on the list of the node it is currently at. This mechanism for deciding which node to move to, along with maintaining the list of the last few nodes, achieves the desired forgetful depth-first search strategy. Figures 7 and 8 also show the list and how it represents the implicit DFS tree.

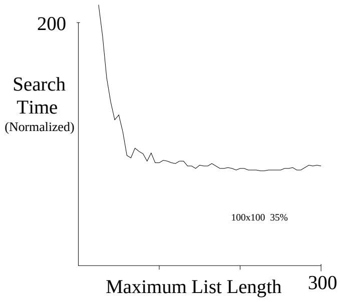

Figure 9: Search times for varying list maximums 100x100

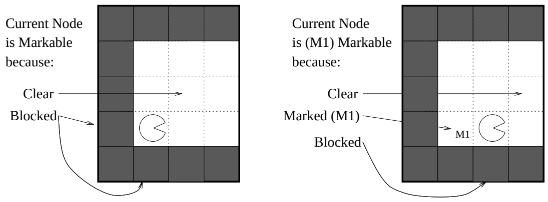

Figure 10: Marking nodes

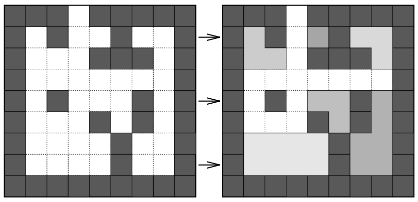

Figure 11: Search space marked into regions

3.4 Target Trespasses on Solver's History

One special case to consider is when the target wanders onto any of the nodes that appear in the solver's list of nodes it has visited recently. Once again, a situation arises that is similar to the one shown back in Figure 5. The problem solver must correct his current information.

One method of dealing with this is to remove the target's node and all prior nodes from the solver's list. However, the list of nodes that remain now represent a path (not necessarily the best path) from the solver to the target. Recall that the solver avoids nodes in the list. The solution adopted is for the solver to empty his list completely when the target wanders onto any of the nodes on it.

3.5 Amount of Memory

An issue that arises with respect to using forgetful depth-first search is how much memory should the solver have, i.e. how long should the list be. Experimentation has been done to determine a good maximum length for the solver's list. Figure 9 shows results of normalized search times versus the maximum allowable length. 100 by 100 search spaces were used in the experimentation. In this case, as well as other experiments (with other sizes), the search time tends to level out when the maximum list length equals the dimension of the search space. Therefore this was chosen as the default length for the forgetful depth-first search algorithm.

4 Marking

Any real time learning technique strictly based on using and updating the heuristic function falls victim to the huge n^2 representation. Algorithms such as forgetful depth-first search may have an excellent strategy for hunting down the target, however no learning occurs. Consequently, the performance of the solver does not improve in the long term. The marking mechanism introduced in this section attempts to provide the problem solver with the ability to learn efficiently and to maintain that knowledge in a higher level representation than the traditional all-pairs shortest path matrix.

4.1 Flat Marking

The objective of marking is to factor the search space into sections so that during the decision process the solver can deal with collections of states instead of individual states. In particular, the problem solver can mark a state with marking i if it is useless for the problem solver to move to that state unless the target is at a state with the same marking i.

As stated, the prerequisite to marking a node is that the solver's ability to track the target is not inhibited by doing so. Formally, a node can be marked if it does not lengthen the shortest path not involving such marked nodes between any two of the node's unmarked neighbors. For grid-type graphs, Figure 10 shows examples when nodes can be marked. A unique marking number is distributed when a node is marked and none of its neighbors are marked. The corner node is marked because one can travel from one of its neighbors to the other in two moves without going through the corner node. When marking a node with a marked neighbor normally the same marking value is used. The next node (to the right of the corner node) can now be marked (with the same marking as the corner node) because its non-marked neighbors have an alternate shortest path between them. Figure 11 shows how a search space would typically be marked after a problem solver explored it.

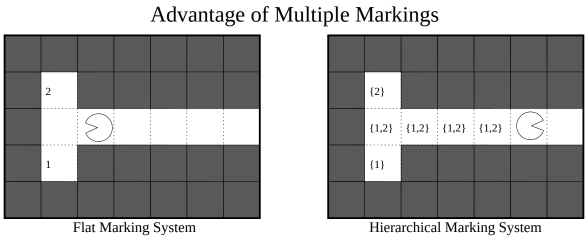

Figure 12: Multiple markings

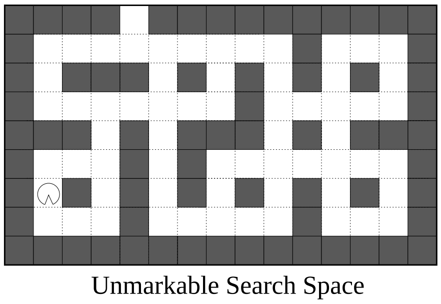

Figure 13: Unmarkable search space

4.2 Hierarchical Marking

Unfortunately there are some intuitive cases which this flat marking system cannot handle. Figure 12 shows an example where a long "dead-end" hallway cannot be conveniently marked off. The problem is that adjacent nodes have different markings. The solution is to mark the new node with a combination of its neighbors' markings. In other words, each node has a set of markings. Unmarked nodes can be thought of as marked with all markings (complement of the empty set).

There is a potential danger in violating the real-time constraint of the search since a single node can have many markings. Comparisons of large sets is not a constant operation. Fortunately the size of such multiple markings are typically small even for large spaces. Even so, the real-time constraint can be guaranteed if the number of markings per node is limited by a constant.

4.3 Marking Limitations

Marking unfortunately does not solve every problem. Figure 13 shows a search space for which this marking strategy is helpless. The graph is two-connected and every 2 by 2 collection of nodes has at least one of them blocked out. Consequently the problem solver cannot mark anything. On the other hand, simple human intuition would easily divide the space into appropriate sections. A more advanced mechanism that can look at more than just one node's neighbors is needed to do marking of this nature.

4.4 Using Markings

The problem solver uses marking information as it decides where to move. It is useless for the solver to move onto a neighboring node that is marked unless the target's node is marked similarly under the flat marking system. When the hierarchical marking system is used, the solver should not move onto a node unless its markings are a superset of the target's node's markings.

This marking technique may augment any moving target search strategy as easily as adding a heuristic function. Marking does not replace existing heuristic functions or learning techniques. By itself, marking would not be an effective strategy. Instead marking often reduces the number of neighboring nodes that the solver must choose from. Because marking is based only on deductive inferences, it is unlikely that adding it will ever make a problem solver less effective.

Even this marking strategy is not immune from the fact that learning takes time. This means that the solver using marking will still enter "markable" heuristic depressions. However, it will only enter them at most once. Consequently, marking will make more of a difference in search times when the search is long and difficult. In this setting the solver will visit areas of the search space many times enabling it to take advantage of previous learning.

5 Experimentation

Experimentation of the various algorithms was done to help understand and compare them.

5.1 Variance in Experimentation

One note concerning the experimentation is that there are large amounts of variance in the time needed to complete the search. One place where variance shows up is in the difficulty of search spaces. Two graphs with the same percentage of nodes randomly blocked out may differ by an order of magnitude in the average time needed for the problem solver to reach the target. Variance also occurs for different runs using the same search space. Sometimes when search parameters are varied slightly there is a large change in the search time; at other times, there is no difference. Such variance results from the high degree of sensitivity that can exist for individual moves. If the solver makes a "wrong turn" he may end up in a large heuristic depression instead of on a direct path to the target.

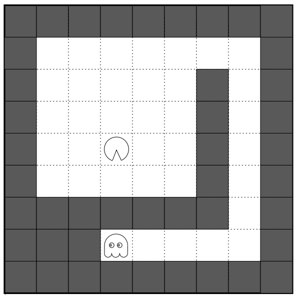

Figure 14: Stationary target search scenario

Algorithm	First Round	Round 10
BMTS	105	17
FDFS	37	37
Marking	37	23

Figure 15: Search Times

5.2 Small Space and Stationary Target

The first collection of experiments presented is based on the graph in Figure 14. The problem solver must work its way out of a heuristic depression in order to find a non-moving target. The main purpose of this experimentation is to become familiar with the nature of each of the algorithms. Algorithms tested include Korf and Ishida's original moving target search algorithm (BMTS), forgetful depth-first search (FDFS), and the hierarchical marking technique combined with forgetful. The results are shown in Figure 15.

The second column of Figure 15 shows the time required for each of the algorithms to find the target. Clearly BMTS does not do very well. This problem solver keeps moving back and forth until it finally builds up its heuristic information. With the stationary target, adding marking to forgetful depth-first search does not improve performance.

The third column of Figure 15 shows the number of moves required a few rounds later (acquired knowledge is maintained between rounds), which helps illustrate the learning that occurs during these searches. For every round after the first 10, BMTS performs optimally heading directly from the start state to the goal state. During later rounds FDFS (which does not adapt) requires the same number of moves that it did in the first round. This is where marking pays off: after a few rounds the number of moves required drops but not quite as much as it did for BMTS.

RAND	mean	med	min	max	over
BMTS	11174	10485	101	>20000	2964
CMTS	2110	915	109	>20000	91
IMTS	1700	512	109	>20000	142
FDFS	876	388	93	>20000	19
Marking	604	383	93	>20000	1
AVOID	**mean**	**med**	**min**	**max**	**over**
BMTS	9176	7300	135	>20000	1844
CMTS	2037	1162	123	>20000	20
IMTS	1476	512	112	>20000	77
FDFS	808	474	105	>20000	40
Marking	619	461	105	>20000	7

Figure 16: Algorithm Performance

5.3 Performance Evaluation

The previous experiment illustrates the behavior of the various problem solvers. The experimentation in this section attempts to measure the ability of these algorithms.

100 by 100 toroidal (wrap-around) search spaces with a density of 35 percent were used. (Each node has a 35 percent probability of being blocked out.) Although not representative of all possible graphs, these search spaces are challenging and appropriate for this moving target search problem. At the beginning of each search the solver and the target are placed a maximum distance (50 rows and 50 columns) apart.

Ten thousand randomly generated graphs (using the above parameters) were used for testing each of the problem solvers. The mean, median, and range are recorded. Searches lasting longer than 20000 moves were halted and 20000 was recorded for the search. The number of searches exceeding this limit is also recorded.

Two types of targets were tested against. One of them (RAND) behaves completely randomly giving equal probability to each neighboring node when deciding its next move. The other target (AVOID) gives higher probability to nodes away from the problem solver thus tending to avoid the solver.

Experimentation was done for BMTS (Ishida's and Korf's original algorithm), the same with Ishida's commitment technique but no deliberation (CMTS), Ishida's algorithm with commitment and deliberation (IMTS), forgetful depth-first search (FDFS), and forgetful with the marking technique. The results are shown in Figure 16. Note that the mean search times for BMTS are deflated because of the large number (almost 30 per cent) of truncated searches.

For CMTS and IMTS the degree of committment used was 10 which is the same value that Ishida suggests in his paper. For the search spaces used in this experimentation, adding deliberation with a maximum off-line search of 25 nodes (as suggested

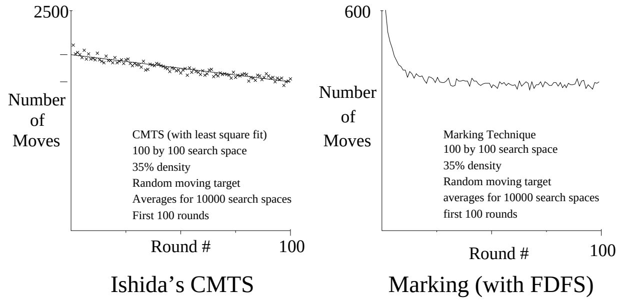

Figure 17: Long term learning

by Ishida) yielded little improvement. Allowing for more deliberation, up to 250 nodes, resulted in better performance for IMTS. The results for IMTS shown in Figure 16 are based on this. Using more deliberation was beneficial because the search spaces were very dense and had huge heuristic depressions. Recall that the search spaces were generated by blocking out nodes instead of just removing edges which is the method Ishida used.

5.4 Long Term Learning

Experimentation was done over a number of rounds to observe general learning on the part of the solver for algorithms that acquire knowledge about the search space. After a search the solver and target were placed back in their initial positions, the solver was allowed to retain any knowledge it had acquired from the previous search, and the search was carried out again.

The hierarchical marking technique combined with forgetful depth-first search and Ishida's moving target search algorithm with commitment (no deliberation) were tested for improvement over 100 rounds. The experimental setup is the same as the previous section, ten thousand times 100 by 100 spaces at density of 35 percent. The random moving target was used.

Figure 17 shows the performance of marking and Ishida's algorithm (CMTS). The data on the CMTS graph is fitted (using least squares) with a straight line. The progression along the x axis is the round number which can be misleading because the amount of time spent learning during a given round is proportional to the number of moves taken in that round. Both algorithms show improvement over time. Note that the scales on the y-axis differ. CMTS has much more room for improvement than marking whose performance is good from the beginning. Also, the data shows that the two algorithms learn at different rates. For marking, the search time drops quickly in the first few rounds and then show no improvement in later rounds. On the other hand, CMTS will likely maintain its slower rate of improvement until it approaches optimal performance.

Algorithm	**Knowledge Repn**
Floyd	all-pairs $\epsilon O(n^2)$
Follow	target's trail
BMTS	all-pairs $\epsilon O(n^2)$
Commitment	all-pairs $\epsilon O(n^2)$
Deliberation	all-pairs $\epsilon O(n^2)$
FDFS	own history
Marking	Regions $\epsilon O(n)$

Figure 18: Primary knowledge base of algorithms

Addable Technique	**can be combined with**	**parameters requiring tuning**
Floyd	Any	none
Commitment	BMTS	degree
Deliberation	BMTS	degree
Marking	Any	none

Figure 19: Comparison of algorithm enhancements

6 Discussion

Figures 18 and 19 summarize various points on the algorithms discussed.

Clearly the original moving target search algorithm does not perform well compared to the other algorithms discussed in this paper. The sensitivity to target's movements and the slow incremental learning hamper this algorithm. However the research by Korf and Ishida laid the foundation for this field of study and also led to Ishida's improved algorithm.

Adding commitment makes a substantial improvement. The learning is focused for maximum benefit. The solver gets out of heuristic depressions much quicker. Unfortunately, learning is "local" and helps little in the long term. In Section 5.4, which shows the benefit of learning over time, Ishida's problem solver improvements come slowly. Eventually (many rounds later) it will behave optimally.

Deliberation effectively reduces the thrashing problem and therefore helps the solver get out of local minimums. Ishida's techniques also require parameters, degree of commitment and deliberation, that must be fine tuned for the current search setting. Overall, deliberation and commitment are very good techniques. Unfortunately they are specifically designed to work with the original moving target search algorithm. They can not readily be added to other algorithms (forgetful depth-first search for example) with the guarantee of performance improvements.

Forgetful depth-first search is simple, requires little overhead and memory, and seems to be a reasonable approach to doing moving target search. The performance results from the experimentation done in this paper are very promising. For the search settings used here, FDFS clearly beats CMTS and IMTS. A criticism with FDFS is that it is not "$O(n^3)$ complete". (Although this is curable for any algorithm by augmenting it with Floyd's algorithm.) On a theoretical note, the forgetful depth-first search algorithm (without any marking technique) would be usable in a dynamic search setting where edges are added and deleted from the graph as the search progresses. Korf and Ishida's algorithms cannot handle edges being added

Marking proves to be a beneficial addition to the forgetful depth-first search algorithm. Although its learning may not be as complete as knowing the true distance between all pairs of nodes, it comes much sooner and is very general. The marking technique can be easily added to any algorithm, it requires no parameters to be tuned, and it is very unlikely that adding marking will have a negative effect on an algorithm's performance.

7 Acknowledgements

The author wishes to thank his supervisor Dr. Jonathan Schaeffer and colleague Sean Halliday for their helpful discussions during this research.

References

[1] T. Ishida and R. E. Korf: "Moving Target Search," IJCAI-91, pp. 204-210, 1991.

[2] T. Ishida: "Moving Target Search With Intelligence," AAAI-92, pp. 525-532, 1992.

[3] S. Baase: "Computer Algorithms, Introduction to Design and Analysis," 2nd edition, Addison Wesley, 1988.