Backward Real Time Heuristic Search in Grid World Pathfinding

David M. Bond
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
dmf5@unh.edu

Abstract

The A* algorithm works well for problems where the solution can be computed offline. In the real time search paradigm, an algorithm must compute the solution online: computation time is not free. Real time heuristic search is search in which the goal is to calculate a plan as close to an optimal plan as possible with the constraint that the algorithm executes an action after each tick of computation time, so that the computation time does not inhibit the optimality of the path taken. In this paper, a new algorithm for real time heuristic search is proposed and examined in the domain of grid world pathfinding. This algorithm named Backward Real Time Search (BRTS) boasts similar performance to modern algorithms such as TBA* and, unlike many other modern algorithms, it does not require pre-computation of the search space and complicated state space abstractions. It beats TBA* by a small margin which is more noticeable at low computation limit values. BRTS also beats a classic variant of LRTA* by more than an order of magnitude. BRTS is a general approach to real time search that one can apply to many search algorithms from A* to Weighted A*.

Introduction

In search problems the A* algorithm [Hart, 1968] is guaranteed to return the optimal solution if an admissible heuristic is used. The A* algorithm assumes the planning and execution stages are independent of one another. In other words, if the planning stage takes a few minutes to complete, this time spent does not affect the execution stage: the computation time is not altering the optimality of the solution. This is reasonable in many applications. When a driver enters a destination address into a GPS unit, the fact that the GPS takes a moment to compute the optimal route has little affect on the total travel time. The planning and execution stages are essentially independent, with the planning stage being computed offline.

However, most likely in this situation, the driver would start moving down the driveway while entering the destination into the GPS. This means that while the complete route to the destination is still unknown, the driver utilizes the limited information currently known to begin the trip towards the destination. More specifically, the driver knows the first move must be to go down the driveway. In an example such as this, the final time gains are insignificant, but consider two other domains where the gains can be very significant, game path finding and robot controls.

If a player in a game gives a command to a unit and that unit sits around for ten seconds while computing the path to the destination, the player is certain to become irritated. When the player gives the command for the unit to move, the player expects the unit to begin executing the command. Also, note that while the path computation may not take ten seconds, games give very little processing time to the pathfinding unit before the algorithm must return control so other elements of the game can do their computation. Some games give 1-3ms of computation time. [Bj¨ornsson, 2009] This means the algorithm does not have the required time to compute a full plan to take.

Similarly, if a controller gives a robot a task and that robot does not begin executing the task right away, the time spent while stalled during the computation of the task plan is in effect adding no-op actions to the start of the plan. This makes an offline optimal plan potentially less optimal then a plan an inadmissible real time algorithm returns. A cleaning robot might have to sweep a floor and clean up cobwebs. The sweeping of the floor might be easy to plan and so there is no reason the robot should wait around doing nothing while planning how to clean the cobwebs. The robot can start sweeping and think about the compete path while working.

This is the definition of, and motivation for, real time heuristic search: calculating a search plan as close to an optimal plan as possible with the constraint that the algorithm executes an action after each tick of computation time so that the computation time does not inhibit the optimality of the path taken. In this paper, a new algorithm for real time heuristic search is proposed and examined in the domain of grid world pathfinding. This algorithm named Backward Real Time Search (BRTS) boasts similar performance to modern algorithms such as TBA* [Bj¨ornsson, 2009] and unlike many other modern algorithms it does not require pre-computation of the search space and complicated state space abstractions.

With the definition of real time heuristic search established, there are several important characteristics of real time heuristic search to consider. Firstly, in this search paradigm algorithm designers often forfeit completeness. Consider a robot moving around a map in which the robot can fall off a cliff. Given a limited search time, the robot may chose to go over the cliff. After which no matter what the robot learns and does the robot cannot reverse the action. This may prevent the robot from reaching the desired goal. An algorithm could compute a path offline by not choosing actions but this would defeat the purpose of real time search. Moreover, completeness is not an issue in domains where all actions have a corresponding reverse action. The cliff example is a problem since there is no way for the robot to climb the cliff. The grid world domain used in this paper has reversible actions. Every state in the set of states reachable from the starting state can reach every state in this set of states.

Secondly, admissibility is impossible to guarantee in real time search since it bases the actions taken upon local information. If that information is incomplete, as it often is, the solution may be suboptimal. This is a disadvantage of real time search and therefore, if an optimal solution is required and computation time is free, this is not the search method to use. The goal of real time search is to minimize this sub-optimality given the limits on local search and therefore the time constraints. Real time search is a close cousin of anytime search because of these considerations.

Domain: Grid World Single Agent Pathfinding

There are a plethora of domains in which real-time heuristic search is applicable. This paper uses a simple domain: four-way movement grid-world pathfinding with a single agent using Manhattan distance as the base heuristic estimator. This is domain is commonly found in video games with the possible modification that many game allow eight way movement. The four-way movement in this example domain is along the four cardinal directions with unit cost in every direction. All actions are reversible. The grid itself consists of tiles that are either passable or impassable. The start and end destination are randomly selected and a path is guaranteed between the two points. The two forms of maps used for empirical observations are randomly generated gird worlds with certain obstacle densities and grids attained from Warcraft maps. Unobstructed tiles on the map represent states.

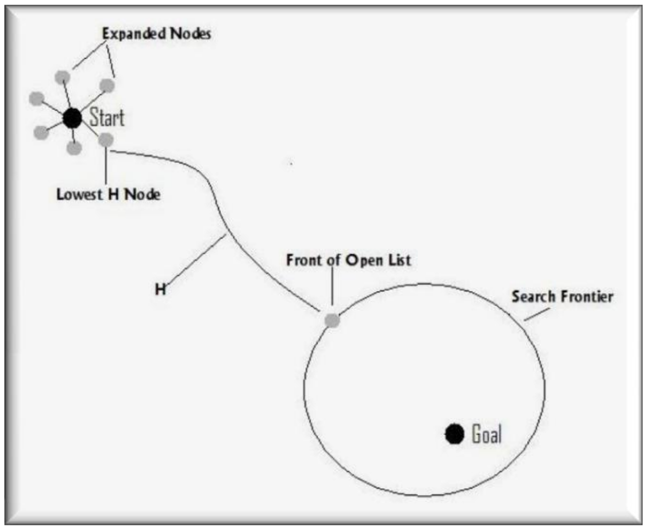


Figure 1

Backward Real Time Heuristic Search

LRTA* is a classic algorithm that solves real time search by repeating breadth first depth limited searches from the current state to the goal state throughout iterations of the planning phase [Korf, 1990]. One of the major disadvantages to LRTA* is how it must throw out all search done after every planning phase. This means it must redo expansions repeatedly through iterations of the planning phase. Backward Real Time Search solves this problem by persisting a single search throughout all iterations of planning. Given an initial state and a goal state BRTS begins searching from the goal state towards the start state. In this paper, the A* algorithm is used to search from the goal to the start state but in general any search algorithm is acceptable.

This search continues until the algorithm exceeds a certain computation limit in the planning phase. At this point, the algorithm expands the start (current) state once so it can commit to an action. The algorithm decides which action to commit to by taking all the generated children and making a heuristic estimate from these to head of open list of the search. The minimum estimate indicates this node is the most promising node towards the head of the open list. The head of the open list is along the most promising path from the goal state to the current state the agent is residing. Figure 1 shows a graphical example of this.

With this minimum node determined the algorithm now commits to the action taken to get to this node. It yields to execution stage with this action. Once the execution stage has completed the algorithm continues its search. The state the algorithm is now searching for is the new state in which the agent resides. This planning and execution process repeats until the search frontier passes the current state of the agent.

Once the search frontier has passed the current state, the algorithm knows a complete path to from the goal to the current state. The algorithm reverses and returns this path for execution.

A number of subtleties have been over looked so far. Firstly, each time the algorithm enters the execution phase the agent's current state changes. This in fact invalidates the open list since the heuristic estimates from previous current states are the metrics with which the algorithm sorted the open list. In grid world path finding, we consider the magnitude of this invalidation small. The algorithm presented here therefore does not resort the list and simply continues using the old open list. Since the nature of real time search specifies that it cannot guarantee optimality, this paper considers this an acceptable compromise to increase the speed of the algorithm.

Another subtlety overlooked is the possibility of loops traversed repeatedly. To understand this, consider Real-Time-A*. Korf points out how if LRTA* does not update the heuristic of the agent's previous state with the mini-min rule LRTA* will often fall into infinite loops iterating between a number of states. While BRTS will not fall into infinite loops due to the search frontier finally reaching the current state, and while BRTS will tend to fall into such loops less due to the movement towards the head of the open list, this is still an issue to consider. Agents will do "obviously stupid" actions when an implementer does not consider this issue.

The problem with using a solution such as Korf's mini-min rule is this algorithm does not have a consistent heuristic comparator between iterations. After every planning phase, the head of the open list may have changed quite significantly and therefore the stored heuristic values are often incorrect. The solution to this used in this paper's implementation is not quite satisfactory and so this problem is left for the reader to consider. In this paper's implementation of BRTS, each time the algorithm expands the current state, if all of the generated children have higher heuristic estimates then the current state, then the algorithm adds that state to a "dead-end" list. When the algorithm expands states, it generates a set of nodes. If any nodes on this list are on the dead-end list, the algorithm ignores them as if their actions did not exist. Doing this is not complete and at times can mean the agent does not have actions to take. To resolve this, the algorithm ignores the dead-end list in this case and the implementation operates as if the dead-end list did not exist. The rationale behind this method is to find dead ends in the search space and mark them as such. A potentially better solution for readers to consider would be to update the memorized heuristics after each planning phase to reflect the change in the open lists size. This is not an O(1) operation unfortunately.

BRTA* Pseudo Code

```
0   BRTA*( Start, Goal, ExpansionLimit )
1     OpenList ← 0
2     ClosedList ← 0
3     OpenList.Enqueue(0,Goal)
4     CurrentGoal ← Start
5     IterationExpansions ← 0
6     loop
7       if OpenList.IsEmpty() do
8         return pathNotFound
9       end
10      Current ← OpenList.Dequeue()
11      if CurrentGoal = Current do
12        return Current.Path().Reverse()
13      else if ++IterationExpansions
14          >=ExpansionLimit do
15        foreach State ∋ CurrentGoal.Expand() do
16          if F(State,OpenList.Peak()) <
17            F(Min,OpenList.Peak() do
18            Min ← State
19          end
20        end
21        Execute( Min.Path().FirstAction() )
22        CurrentGoal = Min.Path().FirstState()
23        if OpenList.Contains(CurrentGoal) do
24          return OpenList.Get(CurrentGoal).
25            Path().Reverse()
26        else if ClosedList.Contains(CurrentGoal) do
27          return ClosedList.Get(CurrentGoal).
28            Path().Reverse()
29        end
30        IterationExpansions ← 0
31      else
32        foreach State ∋ Current.Expand() do
33          if !ClosedList.Contains(State) &&
34            !OpenList.Contains(State) do
35            OpenList.Enqueue(
36              F(State, CurrentGoal), State))
37          end
38        end
39      end
40    end
41  end
```

Backward Real Time A* Pseudo Code

The column to the right contains the pseudo code for Backward Real Time A* (BRTA*), BRTS implemented with A*. This code does not deal with updating of the visited node's heuristic estimates. This function takes a start state, a goal state, and an expansion limit as its arguments. Lines 1-5 are mundane initialization code for A*.

The code then enters the main loop that does planning. After each planning phase, the algorithm yields a partial solution to the calling code so it can execute the potential solution. The executing phase then returns control to the algorithm and the planning continues. Skipping to the bottom, lines 32 through 38 show an ordinary A* expansion of the head of the open list with the heuristic estimate from the child state expanded to the current state.

Lines 11 and 12 show the case where the search frontier passes the current state during the planning phase. At this point, the algorithm returns the reverse path and the calling code executes it.

Lines 13-31 show when the expansion limit has expired and the algorithm must yield an action to the execution phase. Within this block lines 15-20 show where the algorithm expands the current state and the algorithm finds the most promising state. Line 21 shows where the algorithm yields control to the execution phase. It yields so the calling code can execute a single action. Line 22 shows the algorithm updating the current state

The next set of lines from 23 to 29 shows a special case to handle. This case is when an expansion of the current state pushed the agent past the search frontier. At this point, the algorithm has found the solution and must return the path or the algorithm will not find a path as the solution is now inside the search frontier.

Backward Real Time Search Properties

Time Complexity. The time complexity of BRTS in relation to a single planning phase follows the real time search property: the algorithm returns actions within a fixed computation limited time. BRTS limits its computation by only allowing a certain number of expansions per planning phase. One of these expansions must be reserved for the final expansion around the current state. The closed list should be implemented with a hash table to ensure near to O(1) insertion and retrieval. The one area in which BRTS fails to be real time in nature is with its open list. The open list presumably will be implemented as a heap which has O(1) retrieval but O(log n) insertion. While this violates the real time property, the effect of insertions should be minimal due to the log function.

In terms of overall time complexity BRTS is directly proportional to the number of planning phases until the search frontier and current state intersect one another. This is an improvement over LRTA* whose time complexity is nearly the same as the final path length. This is a direct consequence of BRTS saving search between iterations and LRTA* not saving search. BRTS will therefore tend to find a solution sooner when the search frontier intersects the current state since it has had many more iterations to expand this search frontier compared to LRTA* which only has had one iteration to expand the search frontier.

Admissibility. BRTS is not admissible since it must make decisions based on limited information. BRTS, however, does approach optimality in the limit as the computation limit is increased, as LRTA* also does. In fact, if the expansion limit is high enough to find the solution in the first planning stage BRTS is admissible if the search algorithm used is admissible. BRTA* is identical to reverse A* in this case and therefore admissible.

Completeness. BRTS is complete if all states in the set of states reachable from the start state can reach every state in this set. In order words if the domain does not contain irreversible operators, BRTS is complete. This also assumes the algorithm BRTS uses is complete. BRTA* is therefore complete in domains with reversible actions.

Space Complexity. The space complexity of BRTS is dependent on the algorithm used. BRTA* is therefore exponential in the worst case. LRTA* boasts linear time complexity. BRTS could guarantee this with a depth first search such as IDA*[Russell, 2003]

Domain Constraints. LRTA* works on any real time search domain. One of the problems with BRTS is it has limits on which domains it will work with. BRTS first assumes the knowledge of the exact characteristics of the goal state(s). If the algorithm can only specify a domain's goal as a goal predicate, the exact characterization is not possible and so BRTS is not applicable to the domain. Similarly, in domains such as vacuum world, the goal states are enumerable, but there is not a single goal state. In this case, the algorithm could add the enumerated goal states to the open list and then the algorithm could use some method to determine which state to expand first.

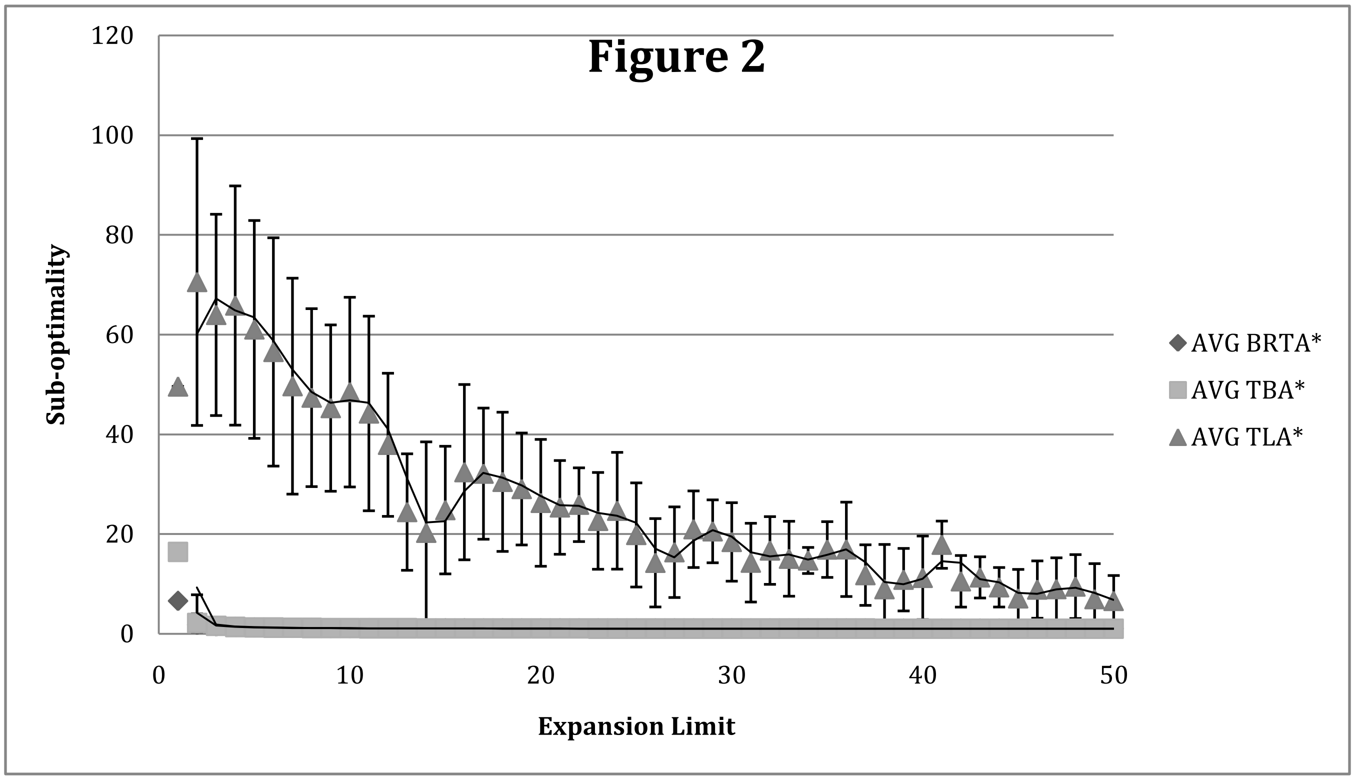
Another problem with BRTS is that it does not work in domains with a limited search horizon. Some video games have fog of war that prevents a player from seeing portions of the map until their units are near the fog. In other words until that portion of the map has been explored. Likewise, a robot might not know what is behind a door. In cases such as this BRTS could choose the most optimistic point towards the final goal along the search horizon limit as a goal state for BRTS and then restart the algorithm as the search horizon expands. LRTA* has a naïve advantage in such a domain because presumable LRTA*'s search depth limit will always be within the limited search horizon. This means it will never perform any differently than if it knew everything about the environment. Searches in general require that they can hypothesize all reachable states and so this limitation is not extraordinary for search algorithms in any paradigm.

Empirical Evaluations

This section presents empirical evaluations of BRTS compared to a classical algorithm and a state of the art algorithm.

Time Limited A*. Time Limited A*(TLA*) is a variant of the classical real time search algorithm LRTA* [Korf, 1990] that utilizes an expansion limited A*, instead of a depth limited breath-first search, during the planning phase. A* is run from the current state to the goal state until the expansion limit has been reached. The agent then travels along the most promising path. This process repeats updating the heuristic values of previously visited nodes with the Korf mini-min rule.

Time Bounded A*. Time Bounded A* (TBA*) is a state of the art algorithm that, like BRTA*, runs A* preserving the search between planning iterations [Bj¨ornsson, 2009]. TBA* runs this search from the start state to the goal state. When the algorithm reaches the expansion limit, the agent



begins following the most promising path. This path may change between iterations. If it does, the agent begins backtracking until the agent is on the new most optimal path. This in the worst case is back at the start state. TBA* can be enhanced by implementing detection of shortcuts between paths. The implementation of TBA* used in these evaluations did not have this enhancement.

Evaluation Methodology

To evaluate BRTS in comparison to TLA* and TBA* each of these three algorithms were run on the grid world path finding domain. The grid worlds came from scaled up maps from the popular game Warcraft. We randomly chose the agents start and destination locations. To compute sub-optimality A* was run offline and the optimal path length was stored. If the optimal path length was less than 75 or greater than 750 actions, the program ignored the map and moved on to the next. If any algorithm path exceeded 100 times the optimal path length, the program cut the algorithm off early. The computation limits used varied from one to 491 in increments of ten inclusive. We ran each algorithm crossed with each computation limit once for four maps. We then graphed these with the sub-optimality along the Y-axis and the computation limit along the X-axis. We define sub-optimality as the number of times the algorithms final path length was above the off-line computed optimal path length

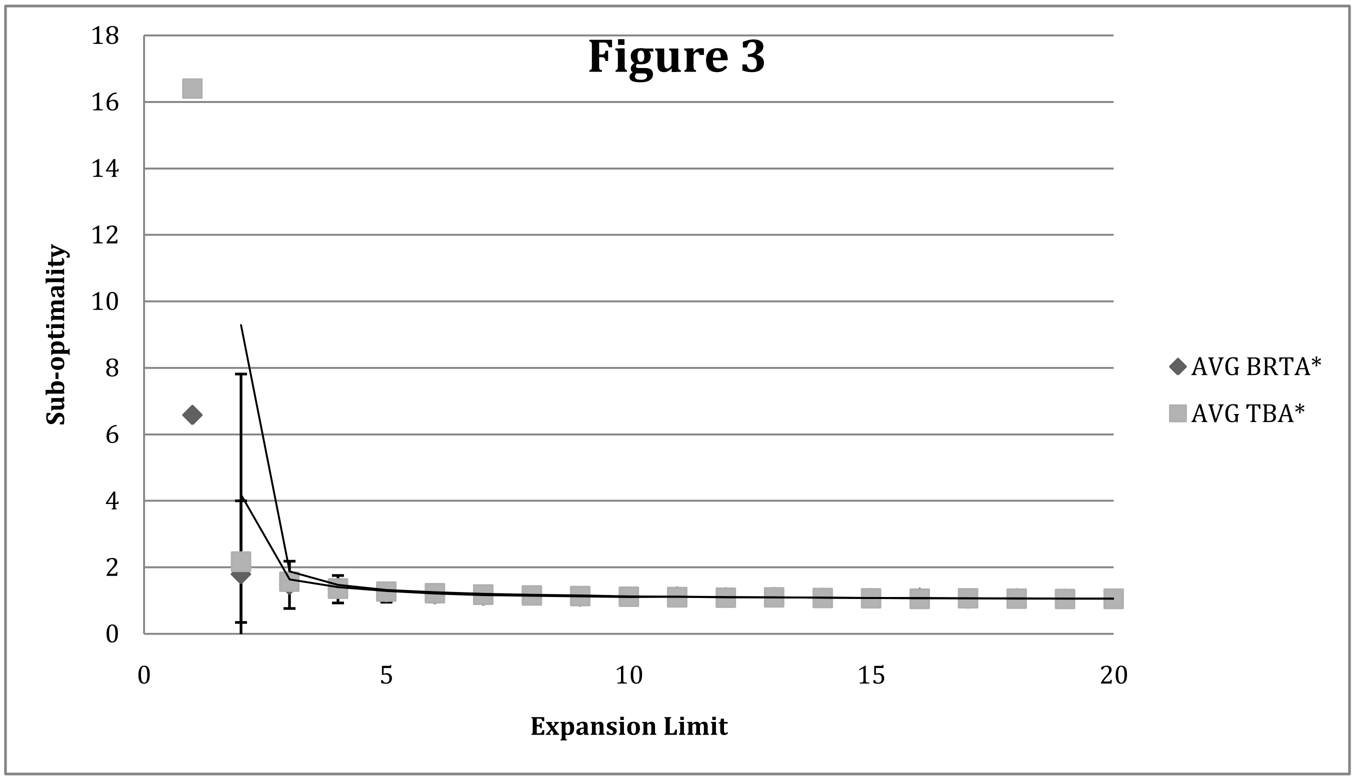
Results

The results of this research as seen in figures two and three shows that BRTA* has comparable performance to TBA*. While the confidence intervals are quite large the initial results indicate BRTA* beats TBA* at very low expansion limits by a small amount. Both of these algorithms are vastly superior to TLA* as one can see the sub-optimality of TLA* goes as high as 70 times and then approaches optimality at the limit. Figure three shows a zoomed in graph which shows that TBA* and BRTA* are near optimal with an expansion limit of five, indicating that these problems are not very complex A* computations.

Two interesting notes to make about these graphs is now TLA* consistently has a much lower sub-optimality at an expansion limit of 1 then at 11. Another abnormality is how BRTA* and TLA* do not have the same sub-optimality for an expansion limit of one. While one would expect BRTA* to diverge quickly at a computation limit of two, at one BRTA* and TLA* should be equivalent.

Future Work

The results seen in this paper are preliminary and there are a number of areas to look into in future research. Firstly, in this paper we compared BRTS to two algorithms. There are a number of other algorithms such as the vanilla flavor



of LRTA*, PR LRTA*, and D LRTA* which would be useful for comparison purposes.

There was also only one domain examined in this work. Domains such as cooperative path finding and tile puzzles would also be interesting to explore. Perhaps not re-sorting the open list, after changing the current state, has a greater detrimental effect in certain domains. We could also test BRTS more extensively on randomly generated graphs. While it we did test on these during the research, we did not produce any formal graphs showing its performance. These would be interesting to see.

With the open list in mind the sorting of the open list is another area of interest for its domain independent effect on the algorithm. The algorithm could resort the open list partially or fully, while ensuring the real time properties and perhaps this would provide a boost to the algorithm.

Next, the problem of learning updated heuristic values for visited nodes is an area in dire need of attention. As mentioned previously if BRTS does not learn these updated heuristic values in some way the agent maybe become stuck in loops. The current solution is unsatisfactory, and while it does work, the payoff for a solution in this area could be very interesting to see in high obstacle density domains.

One final area that is interesting to consider would be hybrid algorithms between LRTA* and BRTA*. Consider how LRTA* searches from the current node to the goal and BRTA* search from the goal to the current node. If an algorithm did a little of both of these perhaps it would have better performance. Doing 9/10th of the expansion limit on as BRTS expands, with 1/10th as LRTA* expands might prevent the agent from going into many "obviously" stupid dead ends. In a way, this is already what BRTS does by expanding its current state once. Perhaps a larger local search is in order.

Conclusion

In conclusion, Backward Real Time Search, or more specifically Backward Real Time A* is a promising algorithm. It boasts solutions similar to other start of the art algorithms in its most naïve form. It beats TBA* by a small margin which is more noticeable at low computation limit values. BRTS also beats a classic variant of LRTA* by more than an order of magnitude. BRTS also benefits from simplicity of implementations and ease of understanding. The most important aspects of future research include open list sorting methods and ways to learn updated heuristics. One should not think of Backward Real Time Search as a single algorithm for solving the real time search paradigm, but rather it is a general approach to real time search. Algorithms from A* to Weighted A* [Pohl, 1970] could be applied as the search method with each method providing a slightly different flavor of search with different characteristics.

Acknowledgments

Special thanks to Professor Wheeler Ruml from the University of New Hampshire Department of Computer Science Department who originally proposed the BRTS algorithm and to Dr. Nathan Sturtevant from the University of Alberta for providing the Warcraft maps to run the calculations over.

References

Bj¨ornsson, Y. Bulitko, V. and Sturtevant, N. 2009. TBA*: Time-Bounded A*. IJCAI.

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". IEEE Transactions on Systems Science and Cybernetics SSC4 4 (2): 100–107.

Korf, R. E. 1990. Real-time heuristic search. Artificial Intelligence 42:189–211.

Pohl, I., "Heuristic Search Viewed as Path Finding in a Graph," *Artificial Intelligence Journal*, Vol. 1, No. 3, Fall 1970, pp. 193--204.

Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, NJ: Prentice Hall, ISBN 0-13-790395-2