Enhanced Real-Time Search for Dynamic Worlds

Dylan O'Ceallaigh

Wheeler Ruml

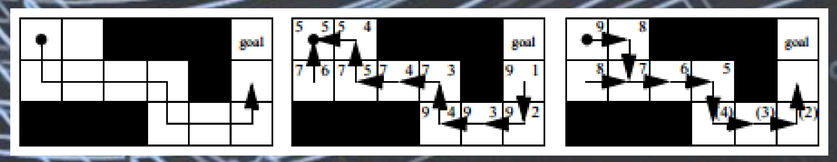
Dept. of Computer Science, University of New Hampshire, Durham, NH

Heuristic Search

Heuristic search is a subset of Artificial Intelligence for which the goal is to navigate a search agent from its initial state to some desired goal state. Heuristic search is differentiated from other methods of search by its use of heuristic estimates. Heuristic estimates (also called h-values) are estimates of the cost from one state to another, and are typically devised from a general rule of cost specific to the world being searched. This rule is referred to as the heuristic function. For the function to be deemed "admissible" (a property necessary to guarantee optimality or even completeness for many searches), it must never over-estimate the cost of traveling from one state to another. Most searches combine a state's h-value with the cost incurred to reach that state, also known as the g-value, to compute the overall cost of the state, also called the f-value.

Real-Time Search

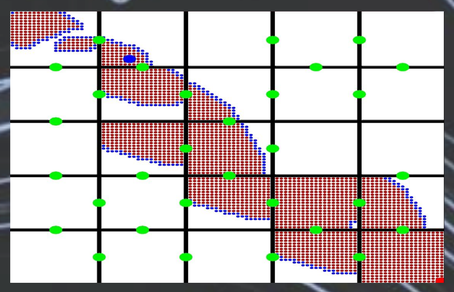
In many domains it is necessary for a search to behave in real-time. In order to do this, the search must be capable of returning an action within a predetermined time bound regardless of the size of the world in which the search is taking place. Because of this limitation, a real-time must return actions before a complete path to the goal has been found. In order to accomplish this, the current state-of-the-art real-time search LSS-LRTA* [1] uses a local search space to determine the most promising action to take according to its current understanding of the world, as represented graphically below:



From left to right:
An incomplete path found by LSS-LRTA*
The local search space as built using an A* lookahead
Path pointers from each state to the most promising successor, found by the Dijkstra subroutine
States are marked with f-values in the upper left and h-values in the upper right.

Dynamic Worlds

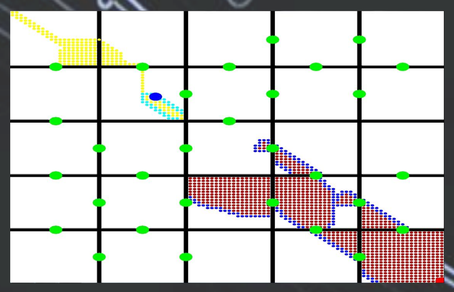
Many domains are designed to change on their own, regardless of the actions of the search agent. In such domains, a path which was previously assigned one cost might change to become more or less expansive. In order to handle this, a search must be able to update its understanding of the world accordingly. The D* family of algorithms [2] is designed specifically to handle such domains, and is depicted graphically below:



Depicted:
Room world, a dynamic environment used for the purpose of testing the proposed algorithm, as well as known dynamic searches. The small blue dots are the search frontier, while the small red dots are states expanded by the search. The large blue dot represents the agent and the large red dot represents the goal. The green dots depict doors which are opening and closing over time, thus providing dynamism to the search domain.

Bidirectional Search

It is often possible to search simultaneously both forwards from the agent and backwards from the goal. This can often improve search performance by sharing knowledge between the two halves of the search. A bidirectional search can behave in real-time as long as its forward half can behave in real-time. An example of bidirectional search is depicted below:



Depicted:
Room world once again, this time running the original RTD* [3] algorithm using LSS-LRTA* as a forward search and D*-lite as a backward search. The small yellow dots are states which the forward search has expanded while the small pale blue dots are states on the search frontier of LSS-LRTA*. The backward search is depicted using the same symbols as above.

Introduction

Background

A number of important domains, such as found in video game path planning and robot motion planning, call for real-time searches to be executed in dynamic worlds. Because of the nature of both real-time searches and dynamic worlds, directly applying an unaltered search in such a way often performs poorly. As such, it is preferable to adapt a search specifically to be able to handle dynamic worlds while maintaining the real-time search property.

One method of adapting a search for such domains is by using a real-time search and a search for dynamic worlds as the forward and backward halves of a bidirectional search, respectively. This approach was taken in the original RTD* algorithm, and is taken in the proposed ERTD* algorithm as well. This approach is not enough in and of itself, however, as any performance improvements are heavily reliant on the two searches connecting as quickly as possible. For this reason, the ERTD* algorithm attempts to adapt the forward search to dynamic worlds, so as to provide superior performance even before the two searches connect.

In order to accomplish this, the forward search portion of ERTD* utilizes concepts proposed in other state-of-the-art searches such as PLRTA*, as well as techniques which have been developed exclusively for the ERTD* algorithm. By applying these techniques to the current state-of-the-art algorithm RTD*, it is intended that ERTD* can be improved to the point that it outperforms other leading searches designed for the same domain.

Scope

As stated previously, real-time searches for dynamic worlds are prevalent in a number of different domains, the most common of which are found in video game design and robotics.

Because of their interactive nature, video games generally require their agents to behave in real time. Real-time strategy games, for instance, will obviously need to abide by the real-time constraint, and are generally too complex for actions to be computed offline prior to gameplay. Because players must be able to react in real-time, they must be able to depend on their agents to carry out commands as they are issued. Additionally, these games typically consist of a multitude of agents acting and moving at once. These agents will also generally interact as moving obstacles with one another, thus introducing the dynamic constraint.

While robotics problems may occasionally involve multiple robots navigating concurrently, this is not common. Moreover, when this does occur, the number of agents searching in parallel is generally small enough that the agents can effectively communicate with one another in such a way that they can trivialize themselves as dynamic obstacles. This does not hold, however, when an external agent such as a human being is navigating the same space. In this sense, robots outside of controlled environments will almost always need to be able to handle dynamic obstacles and, because they cannot stop the simulation as they are computing their next action, do so in real-time.

Implementation Goals

The primary objective of this research is to produce a real-time search algorithm for dynamic worlds which outperforms the RTD* algorithm. In order to implement such an algorithm, a three-pronged approach was devised.

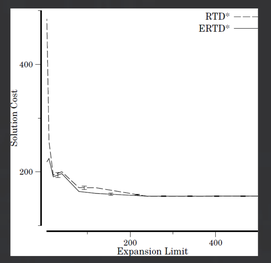
The first technique implemented was the use of a variable $\hat{h}$, which is meant to inflate the heuristic estimates of the forward search. Inflating the heuristic in this manner means that it can no longer be guaranteed admissible, but should at least be a closer approximation of the true cost-to-go. Jordan Thayer et al. [4] describe a method for learning an $\hat{h}$ inflation value such as this, but do so with a one step lookahead. Because ERTD* is a bidirectional search, it has access to states for which it knows the true cost-to-go before the agent ever reaches the goal. As such, we are able to build a much more accurate heuristic inflator, allowing for improved performance of the forward search.

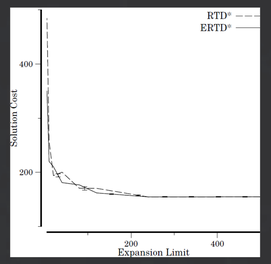
The second technique is borrowed from Jarad Cannon et al. [5]. This technique requires the heuristic estimates of the forward search to be partitioned into static and dynamic halves (one half that will change as the world changes and one which will remain the same). To do this, ERTD* partitions between the base heuristic estimate of a state and the learned estimate formed after revisiting a state multiple times. The learned portion is treated as dynamic, while the rest is treated as static. The technique then calls for the dynamic portion of the heuristic to be decayed over time so that if it was in fact influenced by a dynamic obstacle, it can relearn a correct heuristic estimate of the state after the dynamic obstacle has passed. One small modification in the ERTD* adaptation of this technique is that the heuristic is decayed based on the rate of change in the world, rather than over time. This is done so that the agent cannot get stuck in a loop "following its own tail" if there is no change in the domain.

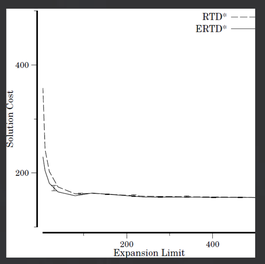
The third and final technique was to better appropriate the backward search to searching toward an agent which is constantly moving and attempting to connect with it. The reason for this technique is that there are sections of the backward search which can be "out of date" and untrustworthy. This technique was ultimately not implemented, as it was not clear how to modify the search, and because it was decided that given the nature of the search, this would not be too problematic.

Results

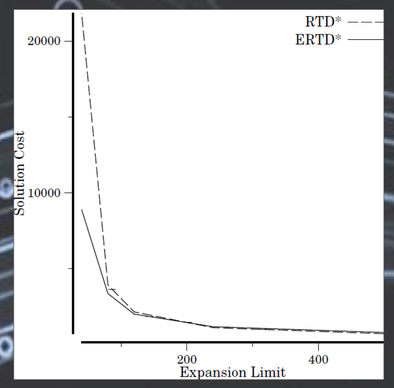
The figures below depict the performance of the RTD* and ERTD* algorithms over 50 instances of the room world. The forward search to backward search ratios are 1:4, 1:2, and 3:4 respectively.

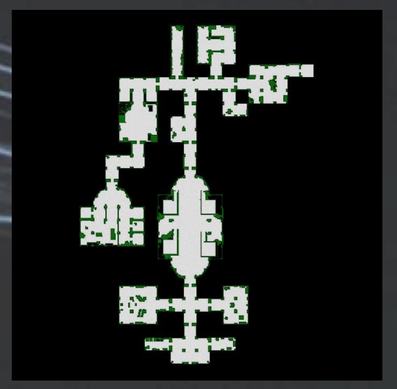






The plot below depicts the performance of the RTD* and ERTD* algorithms over a single instance of a well known static video game map (pictured on right) from the game *Dragon Age: Origins*, courtesy of Nathan Sturtevant [6]. The forward search to backward search computation ratio is 1:2.





Conclusions

ERTD* performs as well as or better than the leading state-of-the-art real-time search for dynamic worlds in all tested domains. Performance boosts are most noticeable in instances with a low expansion limit, suggesting that the algorithm should be used when expansions are particularly expensive or when the time bound in which an action must be returned is particularly short.

Acknowledgements

I am immensely thankful for the insight provided by my advisor, Wheeler Ruml, and for access to the original RTD* algorithm as provided by David M. Bond and Niels Widger.

References

[1] Koenig, S., and Sun, X. 2009. Comparing Real-Time and Incremental Heuristic Search for Real-Time Situated Agents. In *Autonomous Agents and Multi-Agent Systems* 18(3):313–341.
[2] Koenig, S., and Likhachev, M. 2002. D* Lite. In *AAAI/IAAI*, 476–483.
[3] Bond, D., Widger, N., Ruml, W., and Sun, X. 2010. Real-Time Search in Dynamic Worlds. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 16–22.
[4] Thayer, J., and Ruml, W. 2011. Learning inadmissible heuristics during search. In *Proceedings of ICAPS-11* 250-257.
[5] Cannon, J.; Rose, K.; and Ruml,W. 2012. Real-time Motion Planning with Dynamic Obstacles. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 33–40.
[6] Sturtevant, N. 2012. Benchmarks for Grid-Based Pathfinding. In *Transactions on Computational Intelligence and AI in Games* 4(2):144-148.