Speedy and Weighted A* Implementations for the Grid-Based Path-Planning Competition

Maxwell C. Renke
University of New Hampshire
Durham, NH 03824 USA
mcl83@wildcats.unh.edu

Abstract

The Grid-Based Path-Planning Competition is an effort to facilitate comparison between existing approaches to path-planning through the use of widely available benchmark problems. Speedy and Weighted A* are two algorithms that perform suboptimal search and result in faster computation time in exchange for less than optimal final path costs. This paper will discuss the implementation of these algorithms and analyze the trade-off between optimal final path costs and computation time. The results of these algorithms will also be compared against existing approaches made available by the Grid-Based Path-Planning Competition. This paper also exists to fulfill a final project requirement for the University of New Hampshire Computer Science Department course CS830 Introduction to Artificial Intelligence.

Introduction

The Grid-Based Path-Planning exists in a wide variety of applications in both the academic and video game industries. In academia the focus is on finding the optimal (shortest) cost in the fastest time with the fewest resources. Additionally, path planning is required in virtually any video game that need characters to move in the world in a meaningful way. Speed is of the greatest necessity in these games as plans must be found in as short a time as possible as to not adversely affect game-play; but they must also be as correct as possible as to not seem unnatural to the player. Suboptimal search is an approach that exists as a trade off between optimal path cost and state space reduction which results in faster computation time. Suboptimal search has not been a focus in the Grid-Based Path-Planning Competition in the past. This paper seeks to provide suboptimal approaches to the competition and compare the results against existing optimal approaches to attempt to quantify the speed versus optimality trade-off. This paper is mainly focused on the total time to find the path and the suboptimality as a means of comparison between other entries.

Grid-Based Path-Planning Competition

The Grid-Based Path-Planning Competition provides a test framework and benchmark problems to compare entries using a variety of metrics. For each test a map is provided along with a scenario, which is a list of start and goal locations as well as the optimal length for the given path. These metrics include the total time to find the path, the total time to find the first 20 moves (some entires will only report the path one step at at time), the resulting path length, the suboptimality (which is the ratio between the resulting path length and the optimal path length for the given path), and whether the path is valid or not.

The Grid-Based Path-Planning Competition specifies the maps as no larger than 2048x2048 cells that are 8-way connected (i.e. diagonally moves are allowed) with some cells being blocked. A diagonal move is specified as sqrt(2) in an optimal path. Testing a scenario consists of the pre-processing phase and the testing phase. During the pre-processing phase an entry can load the map and store any information to be used during the testing phase. The metrics used to determine performance are taken from the testing phase where the scenario is loaded and the testing harness finds paths based on the given scenario. Speedy and Weighted A* do not make use of the pre-processing phase except to perform some initialization.

Benchmark problem sets provided by the Grid-Based Path-Planning Competition consist of several small examples, mazes, randomly filled grids, as well as grid maps from the video games Dragon Age Origins, Warcraft 3, and the original Starcraft.

Optimal versus Suboptimal Search

Optimal and suboptimal searches both attempt to reduce the search space as much as possible to reduce the number of state that need to be eventually expanded. Optimal searches do this much more carefully than suboptimal searches because an optimal search can only eliminate a state if it is guaranteed to not be part of the optimal solution. In contrast, suboptimal solutions can eliminate states much more liberally through the use of greedy actions. Greedy actions are actions that when examined locally are the best option and are chosen without consideration of the rest of the problem. This usually results a much smaller search space as compared to optimal searches and thus can find solutions faster.

Heuristics

Both optimal and suboptimal searching rely heavily on heuristic functions. Heuristic functions act as an estimate of the amount of search space that needs to be explored before the algorithm has found a valid path. The key difference between a heuristic used in an optimal versus suboptimal solution is the property of admissibility. An admissible heuristic will never over-estimate the remaining cost to reach the goal. Likewise an inadmissible heuristic is one that can overestimate the goal.

Octile Distance

One popular heuristic function is known as Octile Distance. This distance calculation takes into account that a valid move exists in only 8 possible directions with diagonal moves only being made at 45 degree angles. This heuristic is defined as

$$\max(x, y) + (sqrt(2) - 1) * \min(x, y)$$

This provides a better estimate than say Euclidean Distance for the Grid-Based Path-Planning Competition benchmarks.

A*

A* is an optimal algorithm that uses an evaluation function to determine which state in the state space to expand next in order to reach the goal node. The evaluation function for A* is

$$f(n) = g(n) + h(n)$$

where f(n) is the evaluation function, g(n) is the cost from the start state to the current state n, and h(n) is the heuristic function at state n.

Below is pseudo-code for A*.

```
g = goal state
closed list = empty
open list = start state
while open list is not empty
   retrieve state curr from open list with lowest f()
   if curr == goal state
      return
   get valid successor states from curr
   for each successor state
      if not present in closed list
         create new state s
         s.parent = curr
         s.g = curr.g + 1
         s.h = octile_distance(s,g)
         s.f = s.g + s.h          //evaluation function
         add s to closed list
         add s to open list
```

Note that a valid successor is a successor that is not blocked in the map and can be reached via non-blocked cells in the map. When the algorithm returns the path from the goal state to the start state can be found by following the *parent* pointers. This path can be reversed to give the path from the start state to the goal state (which is what the Grid-Based Path-Planning Competition requests).

Speedy

Speedy performs a search much in the same way A* does, but with a different evaluation function. Instead of using a g(n) and h(n) value, Speedy uses a new evaluation function

$$f(n) = d(n)$$

where d(n) is simply

$$\max(x, y)$$

This estimates the number of moves required to reach the goal considering adjacent moves and diagonal moves the same even though the moves cost different values in the final path. Thus, Speedy will expand only the nodes that get closer and closer to the goal state, greatly reducing the search space. Speedy is implemented in the same manner as A*, but with the evaluation function replaced.

Weighted A*

Weighted A*, as the name suggests, is a slight variation on the A* algorithm. Weighted A* uses the nearly the same evaluation function as A*

$$f(n) = g(n) + w * h(n)$$

where w is a value greater than 1 used to influence the heuristic. The heuristic in this case is still the Octile Distance heuristic but since the value of w*h(n) will always be greater than the true estimate to the goal, this heuristic is inadmissible and thus will result in suboptimal paths. However, this change will greatly reduce the state space. Weighted A* is implemented in the same manner as A*, but with the evaluation function replaced.

Implementation

Speedy and Weighted A* all use the same structure (A*, shown above) but their evaluation functions differ. Entries to the Grid-Based Path-Planning Competition are written in C++. The open list in the Speedy and Weighted A* entries are implemented used the C++ standard library heap, sorted by comparing the algorithm's respective evaluation function. The closed list is implemented in an integer array with each element in the array representing a grid cell in the map. This eliminates the need to store the state in the closed list as each element in the array only needs to express a single bit - whether the state is in the closed list or not. This simplifies the check during the algorithm and helps improve performance.

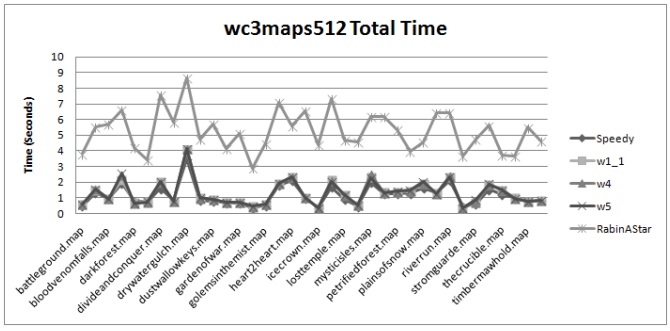
While the implementations for Speedy and Weighted A* achieve desirable performance when finding solutions even on very large benchmarking problems, there is a lot of performance that is still left on the table. Due to time constraints, specialized data structures were not utilized for the open list. While a heap does achieve O(log(n)) performance, this performance could be improved to constant time

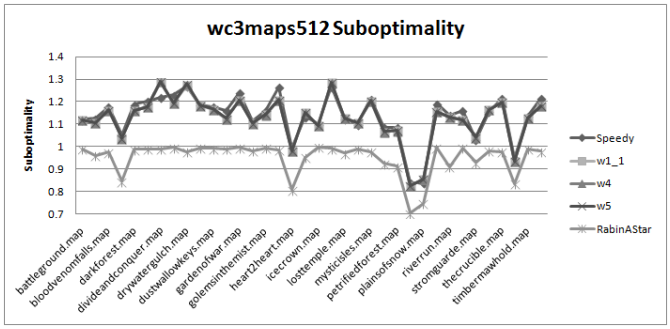
through the use of the bucketlist data structure. This data structure separates several ”buckets” (or arrays) on value boundaries and inserts states into each bucket according to their evaluation function. When an state is retrieved from the table the bucketlist need only return the state in the lowest numbered bucket, which is guaranteed to be the lowest f() value inserted. This brings the open list retrieval operation to constant time at the cost of perhaps some imprecision on the value boundaries. It should be noted that entries to the Grid-Based Path-Planning Competition that use a normal heap or a bucketlist are specified, as they have very different memory requirements and that is a factor in the competition.

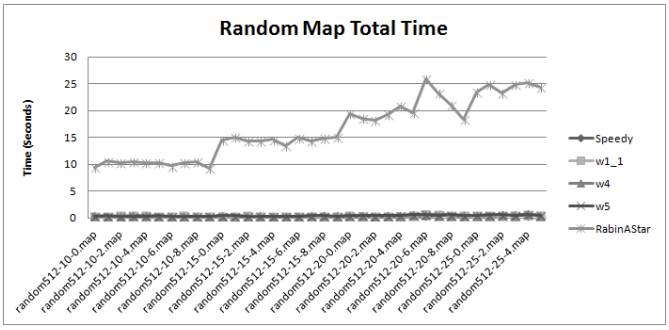
Performance Evaluation

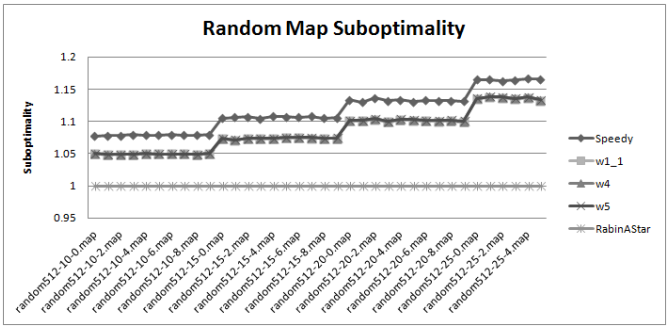
At the time this paper was written the deadline for the Grid-Based Path-Planning Competition had just passed therefore no results from the current competition can be included in this analysis. The results from past entries are included in this analysis and one particular entry, Rabin A*, was selected to be run against the same benchmarks as Speedy and the Weighted A* implementations. Rabin A* is a traditional A* implementation that focused on finding optimal paths (suboptimality $= 1$). The results below were collected on the University of New Hampshire Computer Science Department UNIX server *agate*.

Below are some graphs displaying the total time and average suboptimality of Speedy, several variations of Weighted A*, and Rabin A* against two distinct problem sets. The first problem set is a collection of Warcraft 3 maps labeled ”wc3maps512”. The second problem set is a collection of randomly generated maps labeled ”random”. The Weighted A* algorithms are listed as w1_1, w4, and w5 with weight values equal to 1.1, 4, and 5 respectively.









Below are tables showing an additional summary of other map instances, including maps from Starcraft (sc1).

Total Time - averaged across map instances

	Speedy	w1_1	w4	w5	RabinA*
wc3maps512	1.164	1.300	1.311	1.315	5.266
random	0.365	0.460	0.461	0.463	16.592
sc1	7.062	7.370	7.541	7.640	41.883

Suboptimality - averaged across map instances

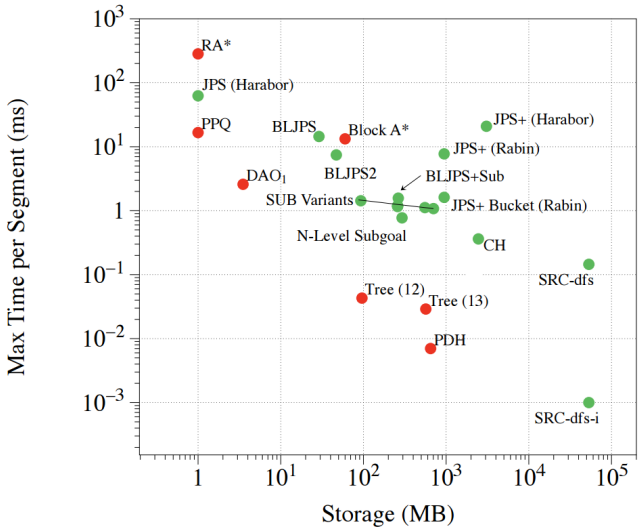
	Speedy	w1_1	w4	w5	RabinA*
wc3maps512	1.135	1.123	1.124	1.124	0.950
random	1.115	1.084	1.085	1.085	0.999
sc1	1.249	1.223	1.224	1.224	0.999

Conclusion

It is clear that there does exist a trade-off between optimal final path cost and computation time. The extent of this trade-off is what is interesting and algorithms such as Speedy and Weighted A* are clearly speed improvements to path-planning if one is willing to give up a certain degree of suboptimality. Since the Grid-Based Path-Planning Competition is focused on ”real world” applications an argument can be made that some degree of suboptimality can be tolerated if the speed benefit provided is of great benefit to the original application. This is certainly the case in video game applications, as well as perhaps some real world applications that are not as concerned with perfect precision.

Speedy and all permutations of Weighted A* find solutions faster than RabinA*. Speedy and Weighted A* (w=1.1) perform the closest to optimal in most cases. Weighed A* where w=4 and w=5 sometimes perform faster than Speedy and Weighted A* (w=1.1) but provide even less optimal paths.

Below is a graph from the 2014 Grid-Based Path-Finding Competition results page comparing various solution's memory usage and the maximum time per segment metric (total time divided by the number of map instances for Speedy and Weighted A*). From the results provided in this paper it is reasonable to expect Speedy and Weighted A* to appear in the bottom left corner of the graph, using about 1MB of memory and finding the path between 1 and 10^1 ms. This seems like a useful area to explore in the future for the Grid-Based Path-Planning Competition.



Speedy and Weighted A* are viable suboptimal search algorithms that greatly reduces the amount of time to find a path while only slightly affected optimality.

Extensions

As mentioned previously the implementations entered to the Grid-Based Path-Planning competition for Speedy and Weighted A* have some areas where performance could potentially be improved. The use of the standard library heap function in C++ is effective enough to get results but could be replaced with a more refined and customized implementation.

The use of a bucketlist was not explored in the context of Speedy and Weighted A*. A bucketlist implementation would certainly decrease the amount of time to find a path but it would potentially add some additional suboptimality to the algorithms due to the nature of the imprecision in selecting buckets. This additional trade-off could be examined.

Further results from the competition would have been a good detail to include in the report.

This paper only examines Weighted A* with weights of 1.1, 4, and 5. Other weights may have a different affect on the performance of these algorithms, and there is potentially that as the weights increase the suboptimality increases or the suboptimality may plateau.

Finally, attempting to find a suboptimal bound on Speedy Weighted A* has yet to be explored. If a bound can be found for these algorithms it can help in assessment of when the use of one of these suboptimal solutions is being examined for use in a real world application.

Final Remarks

In the University of New Hampshire Computer Science Department course CS830 Introduction to Artificial intelligence A* was discussed in great depth with multiple assignments given to implement A* for a variety of domains. This work aided in the implementation of Speedy and Weighted A* as entries to the Grid-Based Path-Planning Competition.

In terms of this paper as a final project I would like to comment on the scope and things I might have done differently. The scope of this paper is somewhere between too narrow and just narrow enough to return meaningful results. Unfortunately at the start of the project I did not have clear performance metrics or goals for which results I wanted to show. This made writing this paper difficult to complete. I was focused on getting a working entry for the Grid-Based Path-Planning competition rather than obtaining meaningful results from my research. While I believe I was successful in obtaining meaningful results I could have shaped my project around the results much better.

Acknowledgments

Special thanks to Professor Wheeler Ruml from the University of New Hampshire Department of Computer Science Department and to Dr. Nathan Sturtevant from the University of Denver, Grid-Based Path-Planning Competition.

References

Jordan Thayer, Wheeler Ruml *Bounded Suboptimal Search: A Direct Approach Using Inadmissable Estimates* Department of Computer Science, University of New Hampshire. 2011.

Christopher Wilt, Wheeler Ruml *Speedy versus Greedy Search* (SoCS-14). 2014

Ethan Burns, Matthew Hatem, Michael J. Leighton, Wheeler Ruml *Implementing Fast Heuristic Search Code* Department of Computer Science, University of New Hampshire. 2012.