Using Swamps to Improve Optimal Pathfinding (Extended Abstract)

Nir Pochter, Aviv Zohar and Jeffrey S. Rosenschein*
School of Engineering and Computer Science
The Hebrew University of Jerusalem, Israel
{nirp, avivz, jeff}@cs.huji.ac.il

ABSTRACT

We address the problem of quickly finding shortest paths in known graphs. We propose a method that relies on identifying areas that tend to be searched needlessly (areas we call *swamps*), and exploits this knowledge to improve search. The method requires relatively little memory, and reduces search cost drastically, while still finding optimal paths. Our method is independent of the heuristics used in the search, and of the search algorithm. We present experimental results that support our claims, and provide an anytime algorithm for the pre-processing stage that identifies swamps.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Graph and tree search strategies*

General Terms

Algorithms, Experimentation

Keywords

Search, Pathfinding, Pruning

1. INTRODUCTION

Many real-time applications search for shortest paths in known graphs. Examples include strategy games where multiple units traverse a large board, as well as robotics applications where robots are required to navigate, planning their path through some environment, usually using some variant of A* [1] or IDA* [2].

We introduce a method that prunes the search graph by removing areas where search is usually wasted, thus lowering the overall search cost. Our method guarantees that paths found are optimal, even after the pruning. We automatically identify areas we call *swamps*, and efficiently store information about them in the graph. Then, while searching for shortest paths between two nodes of the graph, we block the search as it tries to unnecessarily enter those regions. The pre-processing stage is done using an anytime algorithm, in which we locate swamps in a grid; i.e., the algorithm gives better results the longer it runs. The detection process can thus be run in the background, to improve the results of future searches in the graph. We empirically evaluated our method on 2D four and eight neighbor grids with randomly-placed obstacles, where search is performed using the A* algorithm with an admissible, consistent heuristic. The results demonstrate the usefulness of our approach and provide information regarding the efficiency of our method.

1.1 Swamps

Intuitively, a swamp is an area in the graph such that any shortest path that passes through it either starts or ends inside that area, or has an alternative shortest path that does not pass through.[1] We define this notion more formally below (see Figure 1 for an example).

DEFINITION 1. *A swamp $\mathcal{S}$ in an undirected graph $G = (V, E)$ is a group of nodes $\mathcal{S} \subseteq V$ such that any 2 nodes v_1, v_2 which are not part of $\mathcal{S}$ have a shortest path that does not pass through $\mathcal{S}$. A swamp-region $\mathcal{R}$ is a set of connected nodes that is a swamp.*

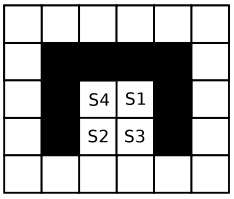


Figure 1: An example of a swamp-region. Nodes filled in black are obstacles. Nodes $\{s_1, s_2, s_3, s_4\}$ form a swamp-region.

2. DETECTING, EXPLOITING SWAMPS

To detect swamp-regions we make use of the following lemma:

LEMMA 1. *Let $\mathcal{S}$ be a set of nodes in V. If for any two nodes v_1, v_2 on the external boundary of $\mathcal{S}$ there exists a shortest path between v_1, v_2 that does not pass through $\mathcal{S}$, then $\mathcal{S}$ is a swamp.*

To detect a swamp-region, we select a connected group of nodes S and trim it to a swamp-region, by checking all shortest paths between pairs of nodes on its boundary. If a pair of nodes contradicts S being a swamp-region according to Lemma 1, we trim S, and repeat until we get a swamp-region.

It is trivially possible to define the entire graph as a single large swamp. This, however, will not be beneficial when trying to exploit

*Aviv Zohar is also affiliated with Microsoft Israel R&D center, Herzlia, Israel. All authors partially supported by Israel Science Foundation grant #898/05.

Cite as: Using Swamps to Improve Optimal Pathfinding (Short Paper), Nir Pochter, Aviv Zohar and Jeffrey S. Rosenschein, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

[1] A slightly more restrictive alternative is to define a swamp as a group of nodes that is *never* used in any shortest path. That definition has nicer properties in some sense, but yields significantly smaller swamps and is thus less useful in practice.

the swamps for more efficient searches. The same goes for very small swamps, that contain just one node. In the first case there are no searches that start and end outside the swamp, and in the second case the graph is barely pruned. We will try to increase the benefits we get from swamps by using a swamp that is completely partitioned into different swamp-regions, any subset of which is still a swamp. For this purpose, we add the following definition:

DEFINITION 2. *A swamp-collection $\mathcal{C}$ is a set of swamp-regions, any subset of which forms a swamp together. $\mathcal{C} = \{\mathcal{R}_1, \ldots \mathcal{R}_k\}$*

To detect a swamp-collection, we apply our detection algorithm incrementally in the presence of swamp regions that have already been located. The resulting swamp-region can be safely added to the swamp-collection.

To utilize the information about swamps during search we apply the following algorithm:

Alg. 1: When searching for a path between nodes v_1 and v_2:
1. Let V be the set of vertices in the graph.
2. Let $\mathcal{C} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ be the full swamp-collection that was found in the graph.
3. Let $\mathcal{R}' \in \mathcal{C}$ be the swamp-region that v_1 belongs to, or $\emptyset$ if v_1 does not belong to any swamp-region.
4. Let $\mathcal{R}'' \in \mathcal{C}$ be the swamp-region that v_2 belongs to, or $\emptyset$ if v_2 does not belong to any swamp-region.
5. Search only in the nodes of $\left(V \setminus \bigcup_{i=1}^{k} \mathcal{R}_i\right) \cup \mathcal{R}' \cup \mathcal{R}''$

LEMMA 2. *Running Alg. 1 will always find the shortest path between nodes v_1, v_2.*

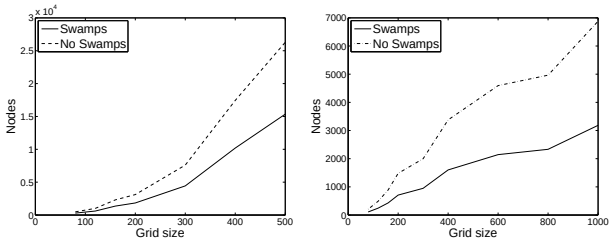
3. EXPERIMENTAL RESULTS

We ran experiments on four and eight neighbor grids, where each node can be either blocked or free. Nodes were blocked at random with varying probabilities in each test, using different grid sizes. In each grid, nodes were independently blocked with equal probability.[2] We then ran our swamp detection algorithms, and various searches, with and without the swamps, to measure performance. We will present here two different types of measurement: the time it takes to perform the search (machine dependent), and the number of nodes expanded (machine independent). Our implementation was in Java; experiments measuring run-time performance were done on a Pentium 4, 2.4GHZ machine, with 500MB of RAM. We ran our swamp detection algorithm on each generated grid, and ran 1,000 searches between pairs of points. Each search was repeated twice: once using regular A*, and once using the same implementation of A* but also using the additional information on the swamp that was detected in the pre-processing stage.

Our experiments demonstrated that using swamps results in a significant saving in search costs (Figure 2). The saving becomes more pronounced in larger grid sizes, where A* expands many more nodes than are strictly needed for the path. The density of obstacles is also a factor in the efficiency of the method. As the number of obstacles rises, so does our algorithm's savings.

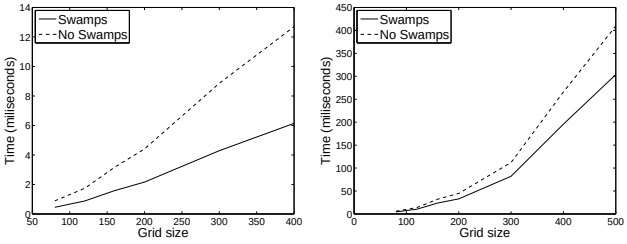
In addition to the number of expanded nodes, we also measured the time it took to detect swamps, and the time it took to execute searches with and without swamps. Figure 3 displays the comparison of search time for 4-neighbor and 8-neighbor grids. The figures show that the saving in the number of nodes expanded translated to a saving in search time. Another interesting measurement is the number of searches, on average, that it takes to make up for the time it took to perform the swamp pre-processing. This data for 4 and 8-neighbor grids, shown in Figure 4, demonstrate that the state pre-processing cost is returned after a few hundred searches, and that this number decreases as the size of the grid increases.

[2]Since 8-neighbor graphs are much more connected, as they contain more edges, we used higher obstacle densities for the 8-neighbor grids—otherwise, a search in these graphs is too easy.



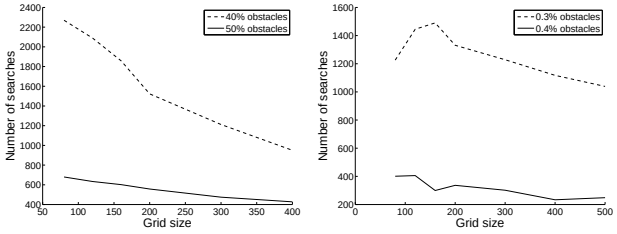
(a) 40 percent obstacles — four neighbor grid (b) 60 percent obstacles — eight neighbor grid

Figure 2: Expanded nodes (swamps, no swamps) and path size, 4/8-neighbor grid



(a) 60 percent obstacles — eight neighbor grid (b) 40 percent obstacles — four neighbor grid

Figure 3: Time it took to execute searches with and without swamps, 4/8 neighbors grids



(a) four neighbor grid (b) eight neighbor grid

Figure 4: The average number of searches needed to make up for the time it cost to pre-process the graph

4. REFERENCES

[1] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[2] R. E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artif. Intell.*, 27(1):97–109, 1985.