Grid-based Path-finding

Peter Yap
Department of Computing Science
University of Alberta
Edmonton, Canada T6G 2E8
peteryap@peteryap.com

March 18, 2002

Abstract

Path-finding is an important problem for many applications, including network traffic, robot planning, military simulations, and computer games. Typically, a grid is superimposed over a region, and a graph search is used to find the optimal (minimal cost) path. The most common scenario is to use a grid of tiles and to search using A*. This paper discusses the tradeoffs for different grid representations and grid search algorithms. Grid representations discussed are 4-way tiles, 8-way tiles, and hexes. This paper introduces *texes* as an efficient representation of hexes. The search algorithms used are A* and iterative deepening A* (IDA*). Application-dependent properties dictate which grid representation and search algorithm will yield the best results.

1 Introduction

Commercial games were a $9 billion (US) industry in 1999, and the rapid rate of growth has not abated [10]. In the past, better computer graphics have been the major technological sales feature of games. With faster processors, larger memories, and better graphics cards, this has almost reached a saturation point. The perceived need for better graphics has been replaced by the demand for a more realistic gaming experience. All the major computer games companies are making big commitments to artificial intelligence [3].

Path-finding is an important problem for many applications, including transportation routing, robot planning, military simulations, and computer games. Path-finding involves analyzing a map to find the "best" cost of traveling from one point to another. Best can be a multi-valued function and use such criteria as the shortest path, least-cost path, safest path, etc. For many computer games this is an expensive calculation, made more difficult by the limited percentage of cycles that are devoted to AI processing.

Typically, a grid is superimposed over a region, and a graph search is used to find the best path. Most game programs conduct path-finding on a (rectangular) tile grid (e.g., *The Sims*, *Ages of Empire*, *Alpha Centauri*, and *Baldur's Gate*). Each tile has a positive weight that is associated with the cost to travel into that tile. The path-finding algorithm usually used is A* [2]. A few games use IDA* (Iterative Deepening A*) [4], which avoids A*'s memory overhead usually at the cost of a slower search. It is worth noting that the commercial computer games industry "discovered" A* in 1996 [9].

Path-finding in computer games may be conceptually easy, but for many game domains it is difficult to do well [1]. Real-time constraints limit the resources—both time and space—that can be used for path-finding. One solution is to reduce the granularity of the grid, resulting in a smaller search space. This gives a coarser representation, which is often discernible to the user (characters may follow in contorted paths). Another solution is to cheat and have the characters move in unrealistic ways (e.g., teleporting). Of course, a third solution is to get a faster processor. Regardless, the demands for realism in games will always result in more detailed domain terrains, resulting in a finer grid and a larger search space.

Most game programs decompose a terrain into a set of squares or tiles. Traditionally, one is allowed to move in the four compass directions on a tile. However, it is possible to also include the four diagonal directions (so eight directions in total). We call the latter an *octile* grid and the former a *tile* grid. Once the optimal path is found under the chosen grid, smoothing is done on this "grid-optimal" path to make it look more realistic [8].

This paper presents several new path-finding results. Grid representations discussed are tiles, octiles and the oft-overlooked hexes (for historical reasons, usually only seen in war strategy games). This paper introduces *texes* as an efficient representation of hexes. The search algorithms used are A* and iterative deepening A* (IDA*). Applicant-dependent properties dictate which grid representation and search algorithm will yield the best results. This work provides insights into different representations and their performance trade-offs. The theoretical and empirical analysis show the potential for major performance improvements to grid-based path-finding algorithms.

2 Path-finding in Practice

Many commercial games exhibit path-finding problems. Here we highlight a few examples that we are personally familiar with. It is not our intent to make negative remarks about these products, only to illustrate that there is a serious problem and that it is widespread.

Consider Blizzard's successful multi-player game *Diablo II*. To be very brief, the player basically runs around and kills hordes of demonic minions... over and over again. To finish the game, the player usually exterminates a few thousand minions. The game involves quite a lot of path-finding, since each of these minions either chases the player, or (less commonly) runs away from the player. In the meantime, the player is rapidly clicking on the screen in an effort to either chase the minion, or (more commonly) to run away from the minion and company. All this frantic running and chasing requires path-finding computations. To complicate the matter, the player is allowed to hire NPCs (non-player characters, called hirelings) or play with other humans in an effort to kill more minions. This significantly adds to the complexity of the game in terms of path-finding.

Consider the scenario whereby a party of human players with hirelings is attacked by a very large horde of minions. From the path-finding point of view, this is a complicated situation. Being very sensible, the human players all independently run in different directions to escape the horde. In this state, path-finding is done on each fleeing player by interpreting the player's mouse clicks, path-finding must be done on each minion so that they give chase to the human players, and path-finding must be done on each hireling so that they flee with their respective employers. On a slow computer or in a network game, this computationally-intensive scenario reduces the game to a slide show, often leading to a player's untimely death, since the minions still attack even while the game appears "frozen" to the human players. One of the solutions applied by the game programmers was to magically teleport the hireling close to the player instead of calculating a path to move the hireling to the player. This solution is not satisfactory; sometimes the hireling is teleported close to the player, but inside a large group of minions. The most serious problem with teleportation is that it detracts from the fun and realism of the game.

As a second case, consider *The Sims*. Here the player controls a family in a household environment. Often, the house is cluttered with obstacles like furniture, making path-finding slightly tricky. Typical path-finding problems involve deadlocks when two Sims are trying to occupy the same space at the same time. A common situation is when a Sim has just finished using the bathroom and is trying to leave through the bathroom door. Simultaneously, another Sim desperately needs to go and rushes towards the bathroom. The two Sims collide at the bathroom door and a deadlock ensues. Often the player must personally resolve the issue. This situation could be avoided with better path-finding (and if the Sims could learn simple courtesy).

These case studies are representative of the difficulties encountered in many commercial games. Clearly, these problems must be resolved if we are to realize John Laird's vision of creating "human-level AI" in these characters [6].

3 Search Algorithms

A* is the classic artificial intelligence optimization search algorithm. It uses a best-first search strategy, exploring the most likely candidate while eliminating provably inferior solutions. Its effectiveness is based on having a good heuristic estimator, H, on the remaining distance from the current state to a goal state. If the heuristic is admissible (does not overestimate), then an optimal answer is guaranteed. On a grid, A* can been shown to explore a search space that is proportional to D^2, where D is the distance to a goal state [7].

Iterative-deepening A* (IDA*) is a memory-efficient version of A*. It eliminates the open and closed lists by trading off space for time. For many applications, space is the limiting factor and thus IDA* is preferred. However, since IDA* iterates and repeatedly explores paths, this may result in a horribly inefficient search that is still asymptotically optimal (e.g., DNA sequence alignment).

The speed of an IDA* search depends on the number of nodes it needs to examine. Analysis has shown that the size of the nodes to be searched is proportional to $O(b^{D-H})$ [5], where b is the average branching factor and H is the effect of the heuristic. Intuitively, this is because IDA* checks every path of length D, and at each depth, each node branches into b more nodes.

4 Tiles, Octiles, and Hexes

In this section an analysis is presented of the cost of path-finding with IDA* using tiles (4 degrees of movement), hexes (6), and octiles (8). We assume non-negatively weighted nodes.

A tile has four adjacent nodes ($b = 4$). Hence a path-finding search has to consider the four adjacent tiles to explore. Since one never backtracks along an optimal path, it is not necessary to consider the direction that the search just came from (i.e., it is not optimal to undo the previously made move). Hence, $b = 3$ (except for the start tile), and the number of nodes that need to be searched to find a solution at depth D for IDA* is proportional to $O(3^{D-H})$.

Now consider a hex grid with six degrees of movement. Using a similar argument, one might deduce that the branching factor of a hex grid is five. However, we can do better and reduce the branching factor to three. Assume that a hexagonal tile's neighbors are in the compass directions N, NE, SE, S, SW, and NW (see Figure 1). Consider moving in direction N from $tile1$ to $tile2$. What is the branching factor now at $tile2$? Moving back to $tile1$ does not have to be considered (backtracking). SE and SW also do not need to be considered, since if they were on the optimal path, one would move from $tile1$ in directions NE and NW, respectively, instead of going to $tile2$. In summary, at each non-root hex, we need only examine three hexes and hence there is a branching factor of three ($b = 3$).

The branching factor of both the tile grid and the hex grid is three. For comparison purposes, the area of the hex is made to be the same as that of a tile. Given the same distance, on average, a path represented by the hex grid is shorter than the path represented by the tile grid (see Figure 2, where the direct path from start to goal is given in bold, and an optimal path following a grid topology is given in regular font). It follows that because the hex path is shorter, one doesn't need to search as deep (i.e., it requires fewer steps to reach the goal node and hence D is smaller). It can be mathematically shown that given the same distance, if a tile grid searches with depth D then a hex grid will search with depth $0.81D$ on average [11]. Combining the branching factors and the depths of both grids, it follows that if the tile grid searches through $O(3^{D-H})$ tiles in a search, then the hex grid searches through $O(3^{0.81D-H}) \approx O(2.42^{D-H})$.

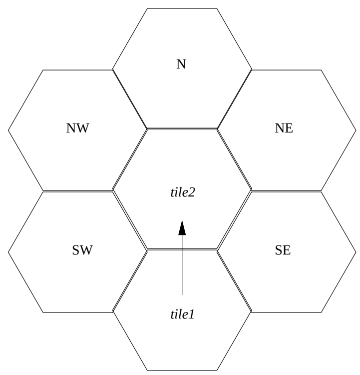

Figure 1: Branching factor of the hexagonal grid.

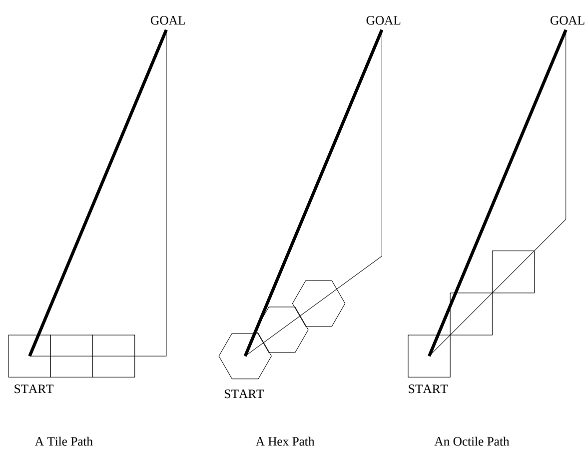

Figure 2: Optimal paths on different grids.

This result proves that a hex grid is exponentially faster than a tile grid for an IDA* search.

Now consider the octile grid, which has eight degrees of movement. Using similar arguments to that given above, we can deduce that the branching factor of an octile grid is five. However, a closer inspection shows that this is too high. With some enhancements, one can reduce an octile search to have an asymptotic branching factor of roughly 4.2 (5 for diagonal movements and 3 for non-diagonal movements). One can also mathematically show that if the tile grid is searched for D depth, then the octile grid is searched for $D/\sqrt{2}$ depth on average (see Figure 2). Intuitively, the depth for the octile should be less than that of a tile because one diagonal octile move is equal to two tile moves. Hence an octile grid searches $O(4.2^{\frac{1}{\sqrt{2}}D}) \approx O(2.77^D)$ [11].

In terms of IDA* search speed, hexes are better than octiles, and octiles are better than tiles. For A*, the asymptotic search speed is indifferent to the choice of grid (although, of course, D will differ).

5 Introducing the Tex Grid

In addition to the exponential search advantage that the hex grid enjoys over the tile grid, hexes have very nice geometric properties. For example, each hex is perpendicular and equidistant to each adjacent hex. Furthermore, a hex shares exactly one side with each adjacent hex. These hex properties provide a better topological representation of the path-finding problem for computer games. Consider Figure 3 where a unit wishes to move diagonally. The search needs to check if the two obstacles that pince the direction of movement are connected (like a mountain) or not connected (canyon) (top row of the figure). The middle row shows a possibly ambiguous tile representation of the two scenarios. Although this ambiguity can be resolved with some extra work, it can be entirely avoided by using hexes or texes (bottom row). For these reasons, it is common to see hexes used in war strategy games. Unfortunately, because of the regular hexagon's shape, the hex grid is harder to implement.

The *tex* grid (a tiled hex) is introduced which is topologically equivalent with a hex structure but uses tiles. One can imagine a tex grid as a tile grid such that the odd columns are moved up by half the height of a tile (see Figure 3 or Figure 7). A bricked wall is another example of a tex grid. Tex grids are more manageable and representative than hex grids since space is represented as rectangles. Additionally, each tex is equidistant and shares exactly one side to each adjacent tex. More importantly, texes have a branching factor of three. Theoretically, texes are only slightly slower than the hex grid on average: $O(3^{0.809D-H})$ instead of $O(3^{0.805D-H})$. Another obvious advantage that texes have over hexes is that every tex path is shorter than a tile path, whereas some hex paths are longer than some tile paths (but on average is shorter). All in all, tex grids are exponentially faster than tiles (and slightly slower than the hex grid), produce smoother and shorter paths, and are easy to work with. The attributes for the choice of grid are summarized in Table 1.

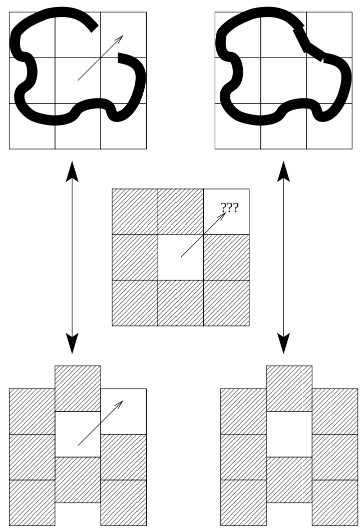

Figure 3: Two scenarios for diagonal moves.

Table 1: Summary of Grids

Grid Type	Adjacent Nodes	Branching Factor	Average Depth	A* Complexity	IDA* Complexity
Hex	6	3	$0.81D$	$O(D^2)$	$O(2.42^{D-H})$
Tex	6	3	$0.81D$	$O(D^2)$	$O(2.43^{D-H})$
Octile	8	4.2	$0.71D$	$O(D^2)$	$O(2.77^{D-H})$
Tile	4	3	$1.00D$	$O(D^2)$	$O(3.00^{D-H})$

6 Introducing the Vancouver Distance

In pathfinding search algorithms like A* or IDA*, we use a heuristic that estimates the cost of reaching the goal. This heuristic is generally the shortest distance between the current node and the goal node in the absence of obstacles. For the tile grid, this shortest distance heuristic is called the Manhattan distance. For the hex grid, we introduce the Vancouver distance. The Vancouver distance also works for the tex grid since it is topologically equivalent to the hex grid.

Given two nodes (x_1,y_1) and (x_2,y_2) under a hexagonal co-ordinate system, let

$$\begin{aligned} x = & \; \|x_1 - x_2\| \\ y = & \; \|y_1 - y_2\| \end{aligned}$$

$$correction = \begin{cases} x_1 \pmod 2 & , \quad \text{if } y_1 < y_2 \text{ and } x \text{ is odd} \\ x_2 \pmod 2 & , \quad \text{if } y_1 > y_2 \text{ and } x \text{ is odd} \\ 0 & , \quad \text{otherwise} \end{cases}$$

then the **Vancouver Distance**, or the number of hexes between the two points, is

$$max\{0, y - \lfloor (x/2) \rfloor\} + x - correction$$

The above result follows from three observations. Firstly, the hexes (x, y) on or under the 30° diagonal are exactly x hexes away from the origin. Secondly, the hexes above this diagonal are exactly $y - \lfloor (x/2) \rfloor$ away from the diagonal (see Figure 4). Thirdly, for two points such that one point is in an odd column and the other point in an even column, the heights of these two points will be different in the Cartesian plane even if they are the same on the hexagonal grid; as such, it is necessary we add a correction term to compensate. Using these facts, we can arbitarily set one node to be the origin and use symmetry to calculate the Vancouver distance between two nodes.

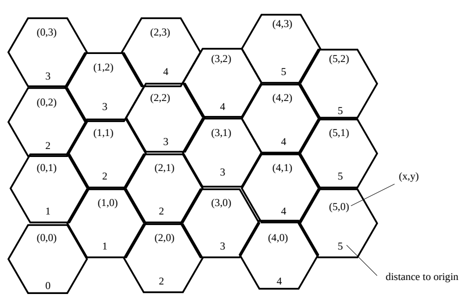

Figure 4: Vancouver Distance under this hex co-ordinate system. Note that we arbitarily arranged the hex grid so that the odd columns are above the even columns.

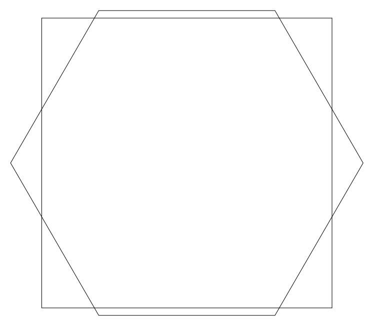

Figure 5: A tile/octile and an overlapping hex. Both have the same area but have different dimensions.

7 Empirical Results

This paper has shown the asymptotic theoretical results, but it is unclear how the various grid topologies behave for the grid sizes normally used in practice. This section contrasts the costs of IDA* searches on tile grids, against comparable tex grids (tiles/octiles and texes are the same size).

There are two reasons why we are empirically comparing tiles and texes, and not use octiles or hexes for comparison. Firstly, comparing hexes and any rectangular grid will not be fair because they are of different shape and size even if their area is the same (see Figure 5); this becomes a problem if we were to compare a MxN hex grid or a MxN tiled grid, as the hex grid would be taller and thinner than the tiled grid by an irrational proportion. As such, we are left with comparing the tex grid versus the tile or octile grid. Although all these grids are rectangular, they all have different topologies (due to their different branching factors). A convenient surjective mapping exists from the tile grid to the tex grid: for every pair of adjacent tile nodes, there exists a corresponding pair of adjacent tex nodes. The corresponding injective mapping does not exist, because not every pair of adjacent hex nodes has a corresponding pair of adjacent tile nodes. However, we can find a corresponding pair of *physically* adjacent tile nodes (see Figure 6).

All test cases where the tex grid has an unfair advantage over the tile grid were removed. For example in Figure 7, the tex grid can't be compared to the tile grid since the tile grid cannot reach A while the tex grid can. It is practically impossible to *fairly* compare these two topologically different grids; nevertheless, the results are presented below.

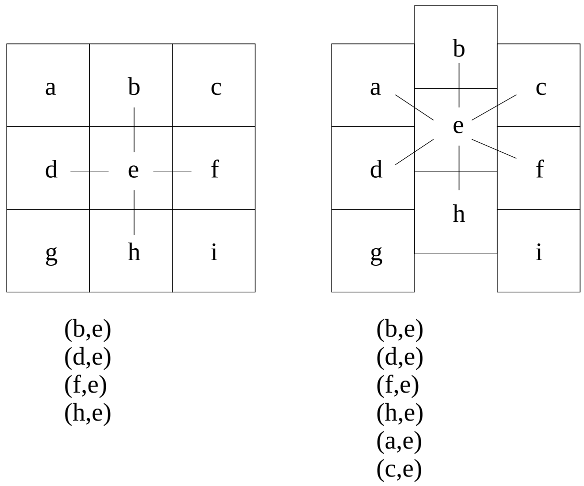

Figure 6: Nodes b,d,f,h are adjacent to node e in both the tile and tex grid. Nodes a and c are adjacent to e in the tex grid but not in the tile grid, but are physically adjacent. If we were to compare the octile grid and the tex grid using the same example, we would find that g and i are adjacent to e in the octile grid but neither adjacent nor physically adjacent in the tex grid.

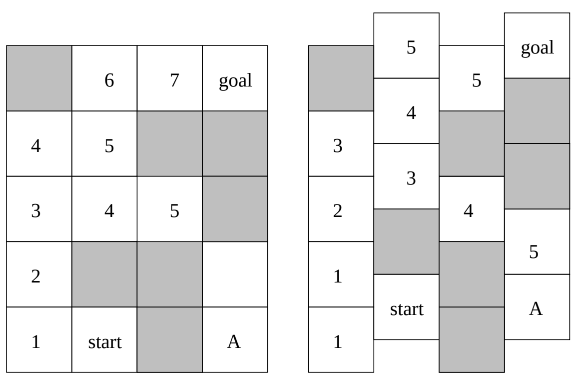

Figure 7: A tile grid and the corresponding tex grid.

Table 2: Empirical Results

Size	Trials	Obstacles	$\frac{TexPath}{TilePath}$	$\frac{TexNodes}{TileNodes}$	$\frac{TexTime}{TileTime}$
10^2	10^6	0%	0.809	0.769	1.172
10^2	10^6	10%	0.809	0.067	0.157
10^2	10^6	20%	0.810	0.049	0.090
10^2	10^6	30%	0.826	0.021	0.048
20^2	10^6	0%	0.808	0.492	0.974
20^2	6000	10%	0.808	0.012	0.020
20^2	1000	20%	0.813	0.019	0.034
30^2	10^6	0%	0.811	0.152	0.285

The tile grid and its comparable tex grid are compared in numerous independent trials (see Table 2). In each trial, a fixed number of obstacles are randomly placed on the grid; after that the start and the goal are randomly placed. Additionally, a path exists between the start and the goal in every trial. A glance at the $\frac{TexPath}{TilePath}$ column show that it reaffirms the theoretical prediction of 0.81. Note that the fraction of the tex path length over the tile path length ($\frac{TexPath}{TilePath}$) grows as the number of obstacles increases, this is not surprising given that the presence of obstacles restrict the advantages of texes or tiles (obstacles reduce the directions of movement).

The $\frac{TexNodes}{TileNodes}$ and $\frac{TexTime}{TileTime}$ clearly show that searches on tex grids examine less nodes and are faster. Note that the tex grid searches slower (but checks less nodes) than the tile grid when there are no obstacles, this is because the tex grid search checks, at the node level, that every move does not give an unfair advantage over the comparable tile grid search; clearly, the hex grid search would be much faster if the need for fair comparisons is removed.

The general trend in Table 2 is that Texes perform better than tiles when the grid size becomes larger or when the number of obstacles increases. This trend is most apparent in the 10x10 grids, whose numbers are more informative because the number of trials is large. In comparison, we only have a limited number of trials for the 20x20 grids since it takes exponential amount of time to gather those trials.

8 Conclusion

Path-finding is an important issue in many application domains, including computer games. It is worth noting that the results of this paper applies not only to computer games, but to any type of pathfinding on a grid. This paper introduces results that increase our understanding of the algorithms and data representations used:

1. Hexagonal grids provide a better topological representation of the underlying problem space. Each hex is equidistant and uniquely shares one side with each adjacent hex.

2. The tex grid retains the advantages of a hexagonal grid but is easier to implement.

3. While the choice of grid does not affect the asymptotic performance of A*, it does for IDA*.

4. It is mathematically proven and empirically shown that the hexagonal grid is superior to the conventional tile grid for IDA* searches. Furthermore, searching on a hex grid instead of a tile or octile grid will result in exponentially faster searches. It can also be proven that a hex grid is optimal in terms of search speed for all regular planar tessellations.

Hex grids provide a better topological representation than tile or octile grids. Moreover, for memory constrained domains that necessitate IDA*, the hex grid is the optimal grid choice. Finally, the implementation of hex grids is made easier with tex grids.

Current research involves analyzing the performance of different grid topologies and search algorithms in BioWare's *Baldur's Gate* series of programs.

9 Acknowledgments

Thanks to Mark Brockington for his invaluable insights into the path-finding issues in BioWare's products. I would like to express my deepest appreciation to Jonathan Schaeffer, this paper has changed and improved a lot as a consequence of his input. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Alberta's Informatics Circle of Research Excellence (iCORE).

References

[1] K. Forbus, J. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, pages 35–40, 2001.

[2] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernet.*, 4(2):100–107, 1968.

[3] S. Johnson. Wild Things. *Wired*, pages 78–83, 2002. March issue.

[4] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[5] R. Korf, M. Reid, and S. Edelkamp. Time complexity of iterative-deepening A*. *Artificial Intelligence*, 129(2):199–218, 2001.

[6] J. Laird and M. van Lent. Human-level AI's killer application: Interactive computer games. In *AAAI National Conference*, pages 1171–1178, 2000.

[7] J. Pearl. *Heuristics: Intelligent search strategies.* In Addison-Wesley, 1984.

[8] S. Rabin. A* Aesthetic Optimizations. *Game Programming Gems.* Charles River Media, pages 264-271, 2000.

[9] B. Stout. Smart moves: Intelligent path-finding. *Game Developer Magazine*, (October):28–35, 1996.

[10] D. Takahashi. Games get serious. *Red Herring*, (87):64–70, 2000. December 18 issue.

[11] P. Yap. *New Ideas in Pathfinding.* PhD thesis, Department of Computing Science, University of Alberta. In preparation.