Direction Maps for Cooperative Pathfinding

M. Renee Jansen and Nathan R. Sturtevant
Department of Computing Science, University of Alberta,
Edmonton, Alberta, Canada T6G 2E8
{maaike, nathanst}@cs.ualberta.ca

Abstract

Cooperative behavior is a desired trait in many fields from computer games to robotics. Yet, achieving cooperative behavior is often difficult, as maintaining shared information about the dynamics of agents in the world can be complex. We focus on the specific task of cooperative pathfinding and introduce a new approach based on the idea of "direction maps" that learns about the movement of agents in the world. This learned data then is used to produce implicit cooperation between agents. This approach is less expensive and has better performance than several existing cooperative algorithms.

Introduction

Cooperative pathfinding is an important topic for many computer games. In real-world interactions humans and animals have adapted mechanisms to avoid each other while moving. However, from a complexity standpoint it has often been much easier to treat all agents individually in video games, planning either as if the world does not contain other agents or as if other agents are static and non-moving.

There are two major approaches that have been developed to handle multiple agents in the world. The first approach has been cooperation through explicit planning and search (Silver 2005). This approach is computationally expensive, but is able to handle complex navigation problems where movement is constrained, such as in narrow corridors.

An alternate approach has been the development of flocking and flow-field techniques (Reynolds 1987; 1999), which dictate local behaviors for individual agents based on a small neighborhood of nearby agents. This method has proven very useful for simulating groups of agents moving in unison, whether crowds of humans or animals. The downside of this approach is that agents can lack a global picture of the world.

This research is concerned with the scenario where there are many agents involved in independent planning within the world. Each individual agent has unique locations that it is moving between. The goal is for agents to be distinct individuals with distinct plans, yet for implicit cooperation to arise in the simulated world.

We accomplish this task with the use of *Direction Maps*, a shared data structure, similar to flow fields, which provides joint information about how agents have been moving in the world, and leads to implicit cooperation during movement and planning. This approach is significantly cheaper than the explicit planning approaches and provides a robust system for movement and planning.

Background

The single-agent pathfinding problem can be formalized as a 4-tuple (G, h, s, t), where G is the search graph, h is a heuristic function, s is the start location, and t is the target location. This can be extended to multiple agents as (G, h, A, S, T), where $A = a_1 \dots a_n$ is the set of agents in the world, $S = s_1 \dots s_n$ is the set of start locations for each agent, and $T = t_1 \dots t_n$ is the set of target locations. Finally, it can be extended to a patrolling problem by requiring that agents repeatedly return to their start location after reaching the target location. While agents may patrol on any graph, in this work we assume that they are patrolling a 2-dimensional grid.

The single-agent pathfinding problem is well-understood, and there are a variety of techniques which have been developed for finding optimal solutions (Hart, Nilsson, and Raphael 1968), sub-optimal solutions (Botea, Müller, and Schaeffer 2004), and partial solutions (Sturtevant and Buro 2005). These single-agent approaches can be used for multiple agents if the dynamics of multiple moving agents are ignored.

A multi-agent approach to this problem is difficult. Consider the problem of multiple agents traveling between multiple start and goal locations in a 2-dimensional world. An optimal solution could be found by considering the joint actions of all agents at each time step. However, this means that the action space at each step is a^n for n agents and a possible actions per agent. If the solution depth is d the total cost of search would be $(a^n)^d$, which is clearly infeasible.

A relaxation of the full problem is to have every agent plan independently. In order to avoid collisions, the completed plans of other agents are taken into account whenever an agent needs to plan a path in the world. This approach is not guaranteed to be optimal, but the computation is more feasible than optimal planning. An even simpler approach is to assume that all agents are static, in which case they can be avoided as if they are part of the environment. This approach is attractive because of low computational costs.

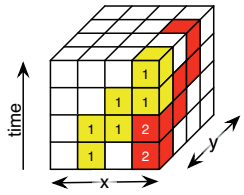


Figure 1: An example reservation table.

These approaches form two extremes on a spectrum. The first approach shares a large amount of data about other agents' plans. This data is continually changing and as a result, algorithms that use the data are complex. The second approach shares a much smaller amount of data, so algorithms that use the data are simpler. Obviously a simpler approach is preferred, but static data quickly becomes inaccurate in a dynamic world and may lead to poor behavior.

However, the trade-offs suggests a different approach. Planning with static data is simpler, but a static snapshot of where agents stand in the world gives very little information that is useful for cooperation. Therefore, instead of using static data about the location of other agents, we propose to build and use static data about the dynamics of the world. That is, data about how agents have historically moved through the environment. This approach is not only simpler than explicit cooperative approaches, but it is also robust and gives rise to emergent cooperation.

If agents share their plans globally and have access to all other agent plans, the 2-dimensional search space is turned into a 3-dimensional plan space, where time is the third dimension. Agents must ensure that their planned paths never pass through the same location at the same time as the path of another agent. This idea is demonstrated in Figure 1, which shows a data structure that could be used for communication between agents. In this figure the z-axis is time, and agents share where they plan to travel at each point in the future. Agent 1 has planned to move along the x-axis and Agent 2 has planned to move along the y-axis. Agent 1 is assured that Agent 2 will move out of the way in time so that they will not collide.

This approach has been used to solve a traffic management problem (Dresner and Stone 2004) as well as for cooperation within games in Windows Hierarchical Cooperative A* (WHCA*) (Silver 2005) and Cooperative Partial-Refinement A* (CPRA*) (Sturtevant and Buro 2006). This type of cooperative search can be very expensive, especially with inaccurate heuristics. WHCA* first builds a more accurate heuristic for the multi-agent planning problem by solving the single-agent problem. To further reduce computational costs, it performs a windowed search through the cooperative space, reverting to a single agent computation after the window is exceeded. This idea can be combined with Partial-Refinement A* (PRA*) (Sturtevant and Buro 2005) to form CPRA*, which uses abstraction and partial refinement to produce similar paths while reducing memory and computation overheads. One drawback to these approaches is that they look for the shortest path possible, when a slightly longer path might have a lower chance of failure. As a result, the movement of agents can be quite chaotic.

A problem which is similar to multi-agent pathfinding is the simulation of groups of agents traveling through a world, known as flocking (Reynolds 1987). Flocking simulations use a small set of rules for movement from which group behavior emerges. Flocking problems can be seen as a subset of the tasks we consider here, but the goal is usually more related to simulating group movement, as opposed to solving a pathfinding problem.

Cooperative behavior can also be achieved with steering techniques (Egbert and Winkler 1996; Reynolds 1999). These techniques assume that the agent is parameterized by a steering force vector, which describes the speed and direction of movement of the agent. Some steering behaviors that are similar to our approach are path following, in which agents follow a path but are allowed to deviate from it to some distance, and flow field following, in which the agent moves in the direction of the local tangent of a flow field. Our approach provides a mechanism to automatically build flow-fields, instead of requiring it to be provided by an external user, as is done in games such as Assassin's Creed (Bernard and Therien 2008). This has the potential to significantly reduce design times.

In the physical world ants are faced with real patrol tasks when collecting food, and they do this by following feromone trails left behind by other ants. Research in biology (Couzin and Franks 2003) has shown that rules for individual ant behavior lead to group behavior in which lanes are formed so that collisions are minimized. Their rules are derived from physical experiments with ants, and are similar to our approach.

In ants and other insects, a phenomenon called *stigmergy* is observed. This term is used to describe the indirect communication within groups of insects that arises because the insects modify the environment, for example by laying feromone trails (Dorigo, Bonabeau, and Theraulaz 2000). The behavior of real-life ants has inspired stigmergy-based algorithms for a number of applications such as optimization problems. In Ant System, for example, virtual feromones were used to do optimization in problems such as the traveling salesman problem (Dorigo, Maniezzo, and Colorni 1996).

Finally, this work bears some resemblance to potential field methods which have been used in robotics. An example of this is Arkin's work on robot navigation, which uses potential fields to guide the robots (Arkin 1987). The potential field consists of vectors which attract the robot towards a goal, and repel it from obstacles. This is combined with high-level behaviors to direct the robot's speed and movement direction.

Direction Vectors and Maps

This section describes implicit cooperative pathfinding using direction maps. This includes a method for learning the direction maps, as well as various methods for using the direction maps when planning. The learning and planning methods are intertwined. We first define a direction vector

(DV), a direction map (DM) and a movement vector (MV). Then we show how, given a complete DM, we we can plan using the DM. Finally, we show that the planning process in itself is sufficient for learning a DM, and formally define the DM learning process.

Definitions

To begin, we assume the world is divided into *cells*. Cells can be the result of dividing a map using any number of methods. For simplicity, we present this approach given that a map is divided into a grid, with each cell the size of a single agent, although other divisions are possible.

Associated with each cell in the world is a Direction Vector (DV). A DV is a vector with length between zero and one, and is a representation of the expected direction in which agents will move through a cell. The length of a DV can be seen as the strength of the correlation. A Direction Map (DM) is the complete collection of DV's for a map, one for each cell. An example of a DM and DV's is in Figure 2. DM-like concepts have been featured in other work, such as (Arkin 1987; Reynolds 1999), but they are used for different purposes and built in different ways.

One step of movement by a agent in the world is represented by a Movement Vector (MV). A MV always has unit length and, in a grid world, points in one of 8 directions.

Planning using DM's

Given a DM, it can be used to modify the planning process of any heuristic search algorithm. The DM is used to change the underlying costs of traversing the world, instead of modifying a search algorithm. For instance, A* expands nodes in order of f-cost, where $f = g + h$. Weighted-A* adds a weight to the heuristic function in order to encourage expanding nodes with lower heuristic cost. When planning with a DM, the opposite is done. The cost of traversing an edge (the g-cost) is modified by the DV at either end of the edge being traversed.

The goal of a DM is to encourage cooperative movement. So, movement which is congruent with the DV's in the DM should be encouraged, while movement in the opposite direction should be discouraged. The exact penalty for moving in the opposite direction of the DV is a parameter, w_{max}. Consider the example in Figure 2. A agent moving from (a) to (b) is moving in the same direction as the DV's at both (a) and (b). Thus, the penalty should be 0. However, if the agent is moving from (b) to (a), it will be moving against the DV's at both (a) and (b), so the penalty for this movement is w_{max}.

In general, when moving between cells a and b, let the underlying cost of the edge between a and b be w_{ab}, and let the direction vectors at a and b respectively be DV_a and DV_b. Finally, let MV_{ab} be the movement vector associated with movement between a and b. The dot-product of the DV and MV will range between -1 and 1, so the cost of moving between a and b is $w_{ab} + 0.25 \cdot w_{max}(2 - DV_a \cdot MV_{ab} - DV_b \cdot MV_{ab})$. It is important that the DV of both the initial and final cell are taken into account, because we want the cost of the edge to reflect the congruence with both DV's.

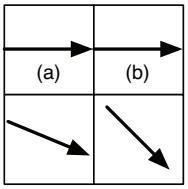


Figure 2: An example of planning with a DM.

In this formulation, cooperative pathfinding is achieved by changing the underlying problem definition. The goal is no longer to find the shortest path, but instead to find the lowest-cost path, following the general direction of other agents in the world. Cooperative pathfinding results not because this allows explicit cooperation, but because agents are unlikely to collide since all units passing through a given location will be traveling the same general direction.

As a side-effect, inexpensive agents can be added to the world which perform a form of flow-field following (Reynolds 1999) on the DM. This is achieved with a greedy one-step lookahead. Units will follow the direction map closely, and at little cost, as they are only expanding one node at each time step.

Learning Formulation

A DM can be learned from the movements of agents as they traverse the world. Although a static DM can be used for planning, a dynamic DM that is updated as agents move can also be used. The process of learning a DM and the individual DV's can be formulated in several different ways. The simplest explanation is that each DV is simply computing a recency-weighted average of the MV's of all agents that have passed through each cell on the map.

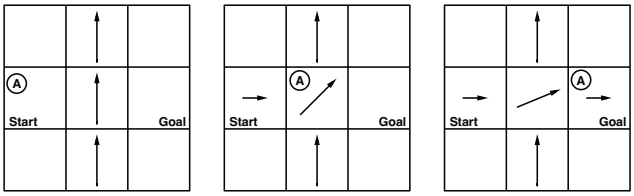
In practice, DV's can be broken into their x- and y-components and learned independently. In this formulation, given that the agent is following a movement vector, MV, at a cell with direction vector, DV, the DV is updated when the agent enters and leaves the cell:

$$DV_x \leftarrow (1 - \alpha)DV_x + \alpha MV_x$$
$$DV_y \leftarrow (1 - \alpha)DV_y + \alpha MV_y$$

In this formulation, α can vary between 0 and 1, controls the weighted average, and can be seen as a learning rate. For higher values of α an individual DV will change very quickly as agents move around the map, while with low values each DV will change more slowly.

Although this is a simple formulation of the learning problem, there is a formal justification of the learning process. With a small amount of algebraic manipulation, it can be shown that these equations are equivalent to the perceptron learning rule (Russell and Norvig 1995). In this formulation, there is a perceptron at each cell in the DM which is attempting to predict the direction that a agent will move through that cell. The perceptron will learn a DV that can then be used for planning.

We demonstrate the learning process in Figure 3. In this example α = 0.5. Imagine that a agent, indicated by "A"



(a) Step 1 (b) Step 2 (c) Step 3

Figure 3: Learning DV's.

wants to travel two steps to the right, to the state marked "Goal". As the agent takes the first step, moving directly to the right, the MV, v, is $(1, 0)$. The agent's current location does not yet have a DV associated with it, so as the agent moves, the DV stored there is

$$(\alpha \cdot v_x, \alpha \cdot v_y) = (0.5, 0)$$

The DV for the agent's new location, in the center of the map, will also be updated. The new value for this DV is

$$\begin{aligned}&((1-\alpha) \cdot DV_x + \alpha \cdot v_x, (1-\alpha) \cdot DV_y + \alpha \cdot v_t)\\ &= (0.5 \cdot 0 + 0.5 \cdot 1, 0.5 \cdot 1 + 0.5 \cdot 0)\\ &= (0.5, 0.5).\end{aligned}$$

As this vector has a magnitude greater than 1, we normalize:

$$DV = (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}})$$

On the next step, when the agent moves into its goal location, the agent updates the center location again. Using the same calculation as before, the DV stored here will be approximately $(0.85, 0.35)$. The last state that is updated is the goal state. Since no DV was stored here yet, we compute the DV similarly to the way we computed it for the agent's start location. The resulting DV is $(0.5, 0)$.

Instead of only updating visited states, the agent can also update neighbors of these states. This allows the agents to create wider corridors as they move. In our implementation, whenever an agent moves, it also updates all eight locations surrounding it using a smaller value for the learning rate.

Reducing Costs Using Abstraction

Planning using DM's is more expensive than using local-repair A*, because the heuristic takes into account the underlying topology, not the topology of the DM. A common solution is to reduce the length of the paths that are planned, which can be done with a simple abstraction mechanism. We divide the world into sectors, and, at a high-level, plan paths between sectors, an approach similar to (Sturtevant 2007). When refining the abstract plan into a low-level path, we then use the DM to plan.

We illustrate this process in Figure 4. In this example, the agent wants to travel from the top left to the bottom right of the map. At an abstract level, the map is divided into sectors, and the agent plans to travel through the three sectors, as outlined in the figure. In previous approaches, agents would

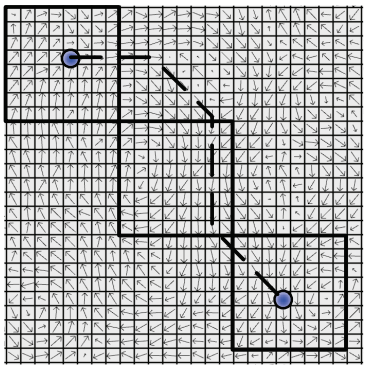


Figure 4: Using abstraction for planning.

normally travel to a particular point within an abstract sector when refining the abstract plan into a concrete path. The alternate proposed here is to find the shortest path to any point inside the next sector on the abstract path. This allows the agent to effectively follow the DM without traveling too far out of the way of the intended path. The final path which is followed is indicated by a dashed line.

There are a number of very minor implementation details which can cause this approach to fail; these most often occur when a agent is standing on or near the boundary between abstract sectors. The easiest fix is for a agent to always plan one step ahead in the abstract path. In Figure 4, this would result in the agent planning directly from the start to the goal. This improves the overall quality of the paths, but increases the computational costs.

Experimental Results

We have tested DM's in a variety of scenarios with many different parameters. A representative subset of the results is described here. In the experiments we place units on a map and assign two unique locations to each unit, asking the units to patrol between the locations a fixed number of times. We chose to evaluate this approach with patrolling tasks because they are common in many games. For instance, almost all RTS games have units patrolling back and forth between resources and a home base.

In this context we can measure a number of statistics about the unit performance. These include the average number of nodes expanded per patrol per agent, the total time taken to finish all patrols (simulation time, not CPU time), the average path length of a single patrol, and the average number of times per patrol loop that an action failed because another agent was in the way.

These metrics are useful, but the result of this work is a dynamic result; one that can visually be observed, but that is difficult to quantify experimentally. As a result, we define a new metric, coherence, which is a numerical measurement of how coherently agents travel in the world. The coherence of agent movement on a map can vary from 0 to 1, although movement through a map is naturally coherent, so low coherence values will never be found in practice.

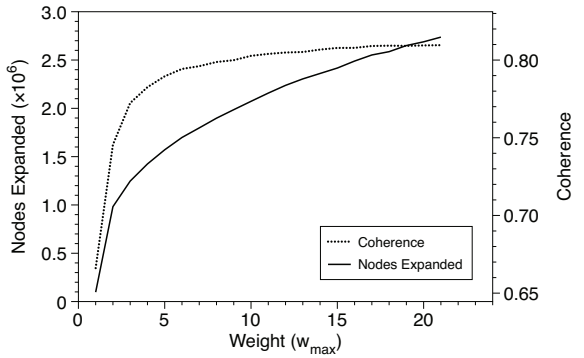
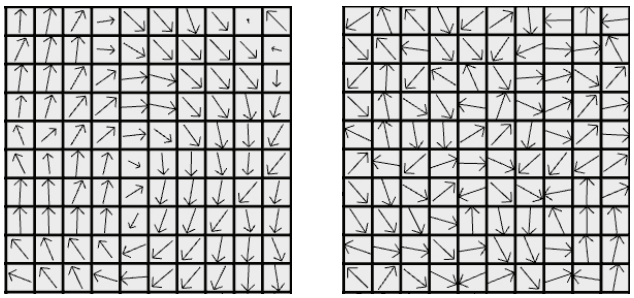


Figure 5: Relationship between coherence and w_{max}.



(a) Using DM (b) Not using DM

Figure 6: The DM's generated when we are and when we are not using the DM for planning

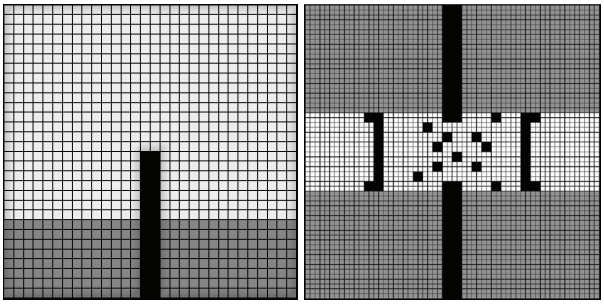


Figure 7: Two sample maps used for experimentation.

Coherence is computed as follows: for every DV, we define DV_t as the DV in the neighboring cell pointed to by the DV. We then compute the length of a new vector formed from the average of the x and y components of DV and DV_t. The coherence of the map is the average of these lengths over all DV's.

We begin by evaluating the learning process. In our first experiment, we place 100 patrolling units on an empty 64×64 map, and vary the weight w_{max} assigned when traveling against the DV. When planning, agents avoid static obstacles within a small radius, and use $\alpha = 0.6$. In Figure 5 we graph the results. The x-axis is w_{max}. On the right axis we plot coherence, shown as a dotted line. On the left axis we plot the number of nodes expanded. This graph shows that we can directly increase the coherence by assessing a larger penalty on units who travel against the DM, with a trade-off in the number of nodes expanded. The coherence graph has a fairly sharp 'elbow' indicating that most of the coherence gains can be achieved with a weight of 5 or less.

In Figure 6 we show a portion of the resulting DM in this scenario. In 6(a) the DM is used with a weight of 15, whereas a weight of 0 is used in 6(b). This is equivalent to not using the DM, although the DM still maintains a DV at each point. This clearly illustrates the effect the DM has on units' travel.

In our next experiments, we compare the performance of units controlled by direction maps to those controlled by WHCA* (Silver 2005). We present an evaluation based on the metrics in Table 1. For the DM, A* is used with a visibility radius 5, weight $w_{max} = 10$ and $\alpha = 0.4$. WHCA* has a cooperative a window size of 4. In this experiment we used the 32×32 map in the left part of Figure 7. 20 agents started in the bottom portion of the map and patrolled between opposite sides of the bottom of the map.

The table shows that WHCA* expands more than five times the number of nodes than the DM. The reason is that WHCA* performs its search including a dimension of time, whereas the DM approach only searches in two-dimensional (x/y) space. Units using WHCA* have a shorter path length, and thus take less simulation time to complete the scenario because WHCA* is specifically looking for short paths, whereas the DM approach puts more emphasis on coherent paths by following the DM. Since WHCA* maintains a reservation table, the number of collisions is also significantly smaller than in the DM case.

From this data, it may seem that WHCA* should be used if the approach is affordable. But, this data does not compare the coherence of the units movement. We compare the coherence of several approaches in Figure 8. The x-axis of this graph is simulation time. At time 0, the DM is blank, so the curves change as more learning takes place. In practice, the DM could be learned offline and loaded at runtime, avoiding the time required for learning.

The best coherence is achieved by using DM's with the neighborhood update rule. This approach more gradually achieves coherence because the neighborhood update rule results in many more cells having low-weight DV's, resulting in a lower overall coherence score. Surprisingly, WHCA* has the worst performance, even worse than regular A* with no DM. This is because WHCA* is only trying to find a fast route, not a coherent one.

In our last set of experiments, we compare planning costs using abstraction with the costs using regular A* and without using abstraction. We present results for 64×64 versions of the two maps in Figure 7 in Tables 2 and 3 respectively. 50 units start on each side of the map in the darkened areas and patrol back and forth from opposite sides of the map.

	WHCA*	DM
Nodes expanded	5813.43	1065.74
Simulation time (s)	83.02	110.55
Path length	70.25	76.44
Failed moves	0.33	1.98

Table 1: Comparison of WHCA* and direction maps

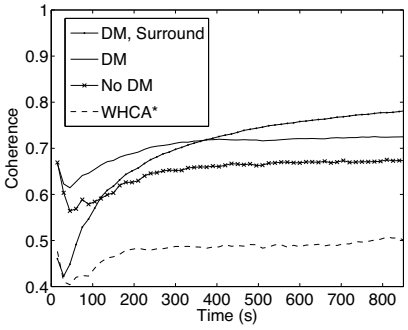


Figure 8: Comparison of map coherence for WHCA* and direction maps

The heuristic in the first map is less accurate than in the second map, hence the larger number of nodes expanded. In the best case abstraction reduces the number of nodes expanded to roughly the same as A*, however there can be significant variation depending on the sector size.

	A*	DM	DM-8	DM-12
Nodes Exp.	4000.82	5438.95	1426.57	3127.52
Sim. Time	5003.94	4644.99	4730.37	4737.54
Path Length	149.49	150.45	151.95	150.94
Failed Moves	21.55	8.08	7.54	8.96

Table 2: DM's with abstraction (map1)

Conclusions

In this paper we have presented the idea of direction maps and shown how they can be used to induce cooperative pathfinding when used during planning. This approach fills the gap between flocking-like approaches, and other approaches which are more planning pased. We have also shown how direction maps can be learned automatically as units traverse the world. The learning rule for direction maps could also be used to automatically learn flow fields.

In the future we plan to extend these ideas to provide a more hybrid approach. Direction maps are well suited to environments in which all units are continually moving. But, we would like to combine the idea of planning with direction maps with cooperative A* to provide an even more robust planning system.

	A*	DM	DM-8	DM-16
Nodes Exp.	2187.73	4001.19	2078.51	2795.56
Sim. Time	4993.83	4382.55	4580.88	4491.18
Path Length	140.00	131.60	135.66	131.77
Failed Moves	19.92	6.59	7.81	7.18

Table 3: DM's with abstraction (map2)

References

Arkin, R. 1987. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *IEEE International Conference on Robotics and Automation*, volume 4, 264–271.

Bernard, S., and Therien, J. 2008. Creating believable crowds in assassins creed. *Game Developers Conference*.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *J. of Game Develop.* 1(1):7–28.

Couzin, I. D., and Franks, N. 2003. Self-organized lane formation and optimized traffic flow in army ants. *Proceedings of the Royal Society of London, Series B* 270(1511):139–146.

Dorigo, M.; Bonabeau, E.; and Theraulaz, G. 2000. Ant algorithms and stigmergy. *Future Generation Computer Systems* 16(9):851–871.

Dorigo, M.; Maniezzo, V.; and Colorni, A. 1996. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics* 26(1):29–41.

Dresner, K., and Stone, P. 2004. Multiagent traffic management: A reservation-based intersection control mechanism. In *The Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 530–537.

Egbert, P. K., and Winkler, S. H. 1996. Collision-free object movement using vector fields. *IEEE Computer Graphics and Applications* 16(4):18–24.

Hart, P.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4:100–107.

Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21(4):25–34.

Reynolds, C. 1999. Steering behaviors for autonomous characters. *Game Developers Conference*.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall.

Silver, D. 2005. Cooperative pathfinding. In *AIIDE*, 117–122.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *Proceedings of AAAI*, 47–52.

Sturtevant, N., and Buro, M. 2006. Improving collaborative pathfinding using map abstraction. In *AIIDE*, 80–85.

Sturtevant, N. R. 2007. Memory-efficient abstractions for pathfinding. In *AIIDE*, 31–36.