A new approach to cooperative pathfinding

(Short Paper)

Renee Jansen
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
maaike@cs.ualberta.ca

Nathan Sturtevant
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
nathanst@cs.ualberta.ca

ABSTRACT

In the multi-agent pathfinding problem, groups of agents need to plan paths between their respective start and goal locations in a given environment, usually a two-dimensional map. Existing approaches to this problem include using static or dynamic information to help coordination. However, the resulting behaviour is not always desirable, in that too much information is hand-coded into the problem, agents take paths which look unintelligent, or because the agents collide and must re-plan frequently. We present a distributed approach in which agents share information about the direction in which they traveled when passing through each location. This information is then used to encourage agents passing through the same location to travel in the same direction as previous agents. In addition to this new approach, we present performance metrics for multi-agent path planning as well as experimental results for the new approach. These results indicate that the number of collisions between agents is reduced and that the visual fidelity is improved.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems

General Terms

Algorithms

Keywords

Cooperative path planning, Emergent behaviour

1. INTRODUCTION AND BACKGROUND

Consider a group of agents in a video game which try to traverse a game map. Many of them may be involved in repetitive tasks, such as patrolling between locations, or ferrying gold back and forth from a gold mine. Other agents may be traversing the map as well. Ideally these agents are able to coordinate their movements to avoid collisions. This is an example of cooperative pathfinding.

In the single-agent pathfinding problem, an agent is given a start and goal location and must find a path between them. The world is often assumed to be static, so that plans are guaranteed success. Most traditional algorithms, like A* and IDA*, assume this type of environment. In some cases it may be possible to use these assumptions in a world with multiple agents. If the world is sufficiently large, the costs of making this assumption will not be large.

When the world is sufficiently constrained, cooperative pathfinding is needed. In this case, multiple agents need to find paths between their respective start and goal locations. This problem is significantly harder, because the agents need to plan around both static and dynamic obstacles. It is not feasible to compute an optimal solution to the cooperative pathfinding problem. If there are u agents and each one can make b moves, the joint action space at each step is already $O(b^u)$.

There are three common approaches to this problem. The easiest approach is to consider other agents as static obstacles and to use individual solutions to the single-agent problem as a solution to the multi-agent problem. A second approach is to do cooperative planning over a limited window of search, and use a non-cooperative solution beyond that window. A third approach is to use flocking rules to modify the behaviour of agents to be more cooperative. We will describe these approaches more in the next section.

Cooperative and flocking approaches are promising, although they can be computationally expensive. They require that agents are able to share information about their location or movement direction. Considering other agents as static obstacles, on the other hand, does not seem promising, as agents will almost certainly move by the time a collision might occur. However, this approach is desirable in that each agent can plan and act individually.

One idea which motivates this paper is agent-centered search [4]. This is a paradigm where agents do not have access to global information about the world, but just some local window or radius around each agent. Constraining knowledge of the environment to a local window leads to the main idea of this research, although the ideas are not limited to this paradigm. Assume that the map is known, but the locations of all other agents are not known. If an agent's view of the world is overly restricted, most cooperative approaches break down. For example, cooperative planning will not work well if agents can only cooperate with adjacent agents.

The key insight for this paper is that agents should attempt to share dynamic information about the environment, as the static approach does not work well. Knowing that a tile is blocked is not interesting, because in a dynamic world this changes too quickly. We propose a new data structure, a *direction map*, which stores the direction an agent last moved at each location. This not only captures some of the dynamics of the world but can also be used during planning and results in emergent cooperative behaviour without explicit communication of plans between agents.

1.1 Related Work

A number of techniques have been used for cooperative behaviour. The first set of approaches is based on the idea that cooperative planning is not just planning in two dimensions, but planning in

Cite as: A new approach to cooperative pathfinding (Short Paper), Renee Jansen and Nathan Sturtevant, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 1401-1404.

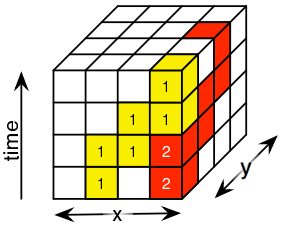


Figure 1: An example reservation table.

three dimensions, through both space and time. After agents plan, they mark a space-time data structure with their future locations. We illustrate this in Figure 1. Agent 1 is planning to move along the x-axis to the location where Agent 2 currently stands. Depending on the order of planning, Agent 2 either has already planned to move, or moves to get out of the way of Agent 1.

Dresner and Stone use this idea for a traffic management application [3]. In their domain, they are guiding traffic through a 4-way intersection. They show that this approach allows vehicles to move through the intersection faster than with stop signs or traffic lights, while collisions are avoided. This approach works well here because traffic is already organized into lanes, and therefore there is only a small area where cooperative planning is needed.

The same ideas have been adapted in the Windowed Hierarchical Cooperative A* (WHCA*) [6] and Cooperative Partial-Refinement A* (CPRA*) [7] algorithms. In these approaches agents are not restricted to traffic lanes, but freely travel around the environment. The drawback to this approach is that agents' cooperation is windowed, which can lead to a horizon effect, where agents plan just enough to push congestion past the windowed horizon. In this case, agents behaviour can appear to be quite bizarre.

A second common approach to cooperative behaviour is flocking [5]. In a flocking simulation, agents have three objectives. First, to avoid collisions with nearby members of the flock, second to match their velocity with that of nearby members of the flocks, and third, to stay close to their flock. The behaviour of this approach, when properly tuned, can be quite impressive. Similarly to reservation methods, a reasonably sized window of information about other agents must be maintained for this approach to work.

Cooperative behaviour can also be obtained by using ant-inspired approaches. An example is Ant System, which was developed by Dorigo *et al.*[2]. This approach uses virtual feromones to do optimization, for example in the traveling salesman problem. Similar to our approach, agents leave a piece of information behind when they pass through a state in order to do path planning. However, the goal of the approaches is different. In Ant System, the goal is to find an optimal path between a single start and goal location, whereas we consider the case where the agents have their own start and goal positions.

2. DIRECTION MAPS

The novel idea of this paper is a *direction map* (DM) which stores information about the direction that agents have traveled in each portion of a map. Agents then use this information during planning; moves which run counter to the *direction map* incur additional penalties so that agents are encouraged to move more uniformly across the environment. If the agents are able to form something akin to lanes (which were manually constructed for traffic management in [3]), the cohesiveness of their behaviour will increase.

UpdateDirectionVector(α, currentLocation, moveDirection)

1 $v1 \leftarrow GetVector(currentLocation)$
2 $v2 \leftarrow GetDirectionVector(moveDirection)$
3 $direction_{currentLocation} \leftarrow$
$\alpha \cdot v1 + (1 - \alpha) \cdot v2$

Figure 2: Pseudocode for updating the direction of a location

ComputeEdgeCost($weight$, e_{ab})

1 $v_m \leftarrow GetDirectionVector(e_{ab})$
2 $v_a \leftarrow GetVector(a)$
3 $v_b \leftarrow GetVector(b)$
4 $weight_a \leftarrow \frac{1-(v_m \cdot v_a)}{2}$
5 $weight_b \leftarrow \frac{1-(v_m \cdot v_b)}{2}$
6 return $cost(e_{ab}) + weight \cdot (\frac{1}{2} weight_a + \frac{1}{2} weight_b)$

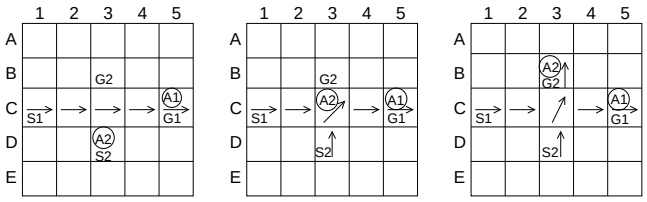
Figure 3: Pseudocode for computing weighted edge costs

First, consider how this can be done from a global perspective. Associated with each location in the world is a *direction vector* (DV). The first time a location is visited by an agent, the direction from which the agent moved into the location is stored as the DV for that location. Then, each successive time an agent enters or exits the location, the DV is updated to be a weighted average of the previous DV and the DV formed by the angle from which the agent moving through the location. After each update, the DV is normalized. This is demonstrated in Figure 2. A learning rate α is used to determine how fast the DV changes as agents traverse the environment.

These stored directions are then used to guide the agents during planning. The cost of traversing an edge e between locations a and b is the cost of the edge plus some function of the DVs for the locations that are being entered and left. This function can take many different forms. Intuitively there should be some penalty if we are leaving location a in a direction which is different from the DV, and similarly when entering location b. A simple approach, demonstrated in Figure 3, is to compute the vector an agent will be traveling between a and b, and then to take the dot-products of that vector with the DVs for both a and b. We normalize each dot-product to be between 0 and 1, where a value of 0 means the agent is following the direction that is stored in the node, and 1 means the agent is moving in the opposite direction. We take the average of the two normalized dot-products and multiply it by a weight which determines the actual cost of going in the 'wrong' direction. This weight is varied in our experimental results.

An example is given in Figure 4. Imagine that one agent, indicated by $A1$, has previously moved from location C1 to location C5. The arrows in Figure 4(a) indicate the directions for each location that were generated by this. Now, consider agent $A2$, who wants to move from its start position D3 to its goal location B3. It plans the route D3 - C3 - B3, so its first move is 'up'. This move changes the DVs stored in both D3 and in C3. Since no DV was stored in D3 yet, the DV is set to be the same as the direction in which the agent left, which is 'up', represented by the vector $[0, 1]^T$. For C3, the DV becomes a weighted average of the previous DV, $[1, 0]^T$, and the movement vector, $[0, 1]^T$. For this example, we assume that the two vectors are weighted evenly, *i.e.*$\alpha = 0.5$, which gives us $0.5 \cdot [1, 0]^T + 0.5 \cdot [0, 1]^T = [0.5, 0]^T + [0, 0.5]^T = [0.5, 0.5]^T$. The last step before the vector is stored is to normalize the DV, which gives $[\sqrt{2}/2, \sqrt{2}/2]^T$. This is the DV that is shown in location C3 in Figure 4(b).

The last step taken by agent A2 is from C3 to B3. The 'up'-



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

Figure 4: Example

direction the agent moves in again corresponds to the vector $[0, 1]^T$. The DV currently stored in C3 is $[\sqrt{2}/2, \sqrt{2}/2]^T$, and the two vectors are combined as above to give $0.5 \cdot [\sqrt{2}/2, \sqrt{2}/2]^T + 0.5 \cdot [0, 1]^T = [\sqrt{2}/4, \sqrt{2}/4]^T + [0, 1/2]^T = [\sqrt{2}/4, (\sqrt{2} + 2)/4]^T$. The DV is again normalized and stored.

As currently described, agents share a global direction map DM), which is counter to the notion of agent-centered search. However, it is simple to update this idea to be agent-centric. To do this, each agent maintains its own DM, according to what has been experienced when moving around the world. At each step, an agent updates its DM within a local radius. The observed DM is used for future planning, although it may become out-of-date after time passes. We assume that agents can store both the world map and a copy of their own DM.

3. PERFORMANCE METRICS

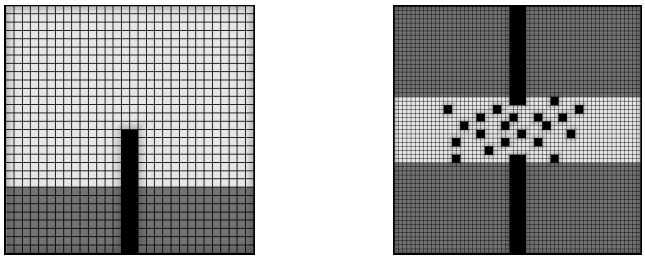
One difficulty with measuring the performance of a cooperative pathfinding approach is designing suitable metrics for the task. Although approaches which provide high visual fidelity are preferred, it is difficult to quantify this from a human perspective. We considered several different metrics, including the number of collisions during simulation, the number of turns made by each agent, the distance traveled, and the time taken to complete a task. Of these metrics, the one which could best distinguish different approaches was the number of collisions during simulation. The distance traveled and the time taken was similar for all approaches we used.

A new metric which arises from our approach is the coherence of the DM during simulation. For each location and direction vector in the DM, we can measure the dot product of the DV with the DV at location at which an agent would arrive if it traveled in the direction of the DV. In simple terms, this measures whether adjacent vectors in the location map point in the same direction, which is an indication of how coordinated the movement of the agents is. This, too, proved to be a useful metric into the performance of the cooperative pathfinding.

4. EXPERIMENTAL RESULTS

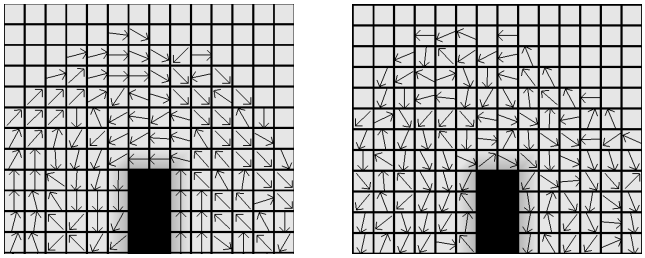
In this section we evaluate the performance of DMs. All experiments were performed in the Hierarchical Open Graph (HOG) testbed, which is publicly available [1]. We used a number of different maps for our experiments, two of which are shown in figure 5. The map shown in Figure 5(a) is a 32 x 32 grid, and the map in figure 5(b) is a 64 x 64 grid. The experiments consisted of a number of agents which patrol back and forth between two locations (chosen at random from the dark areas) on the map, with the number of patrols predefined. Each agent is assigned one patrol location in the left-hand side, and one on the right-hand side of the map.

We present a subset of our experimental results, although experiments with different parameters did not differ significantly from what we present here. For all experiments, agents must complete



(a) (b)

Figure 5: Examples of maps. Dark areas indicate possible start and goal locations for the agents.



(a) Weighted environment (b) Non-weighted environment

Figure 6: The arrows generated in the weighted and non-weighted environments

10 patrol loops between their start and goal locations, and the α parameter from Figure 2 is set to 0.8. The weight parameter from Figure 3 was either set to 5 or 10. Agents are able to see other agents within a small radius (4-5 steps), and A* is used for all experiments. We compare the agents using DMs to agents which just use static information about other agents during planning.

Figure 6 shows two DMs. Figure 6(a) is from agents that use weights from the DM to modify their search and demonstrates that the agents form clear lanes of travel. Figure 6(b) is from agents that update the DM, but do not use weights from the DM to modify their search. This DM shows more chaotic movement. Cooperative algorithms like WHCA* will not improve this behaviour, as the goal is to avoid bumping into other agents, not to move in a visually cohesive manner.

Our first experiment measures DM coherence. We measured the average coherence of the direction map during 50 different scenarios for agents that used the DM during search and those that did not. Agents could see other agents within radius 5, and the cost of traveling against the direction map was given weight 5. For agents not using the direction map for planning, we still updated the direction map to measure how they traveled across the environment.

In both cases, the DM coherence is high at the beginning, because a DM is fully coherent when it is empty. After an initial stabilization phase, the agents using the DM formed gradually more coherent paths. Agents which did not use the DM had much less coherence, although the coherence of their travel increases as the scenario draws to a close. This is because some agents finish their travel earlier than others, so the remaining agents are able to form more coherent paths.

Next we compare the number of failed moves between agents. Figure 8 shows the results, averaged over 50 trials. The weight on the cost of traveling against the DM was set to 10. This figure

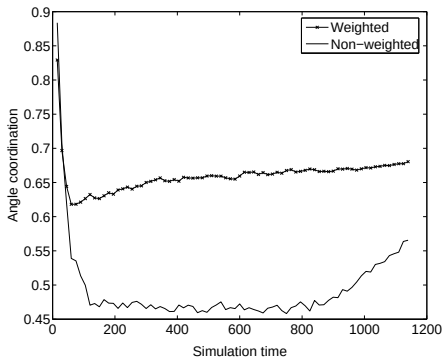


Figure 7: Angle coordination

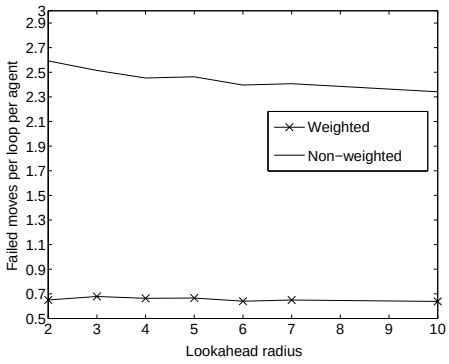


Figure 8: Failed moves per loop, per agent, for varying lookahead radius.

shows that the number of failed moves per agent per patrol loop is significantly larger for agents which do not use the DM than for those that do. Agents using the DM naturally avoid agents moving in the opposite direction.

Note that each failed move requires a re-planning step. Depending on the algorithm used, this re-planning can be expensive. But, it is also more expensive to plan using the weights from the direction map. Our experimental results showed that these costs offset each other, so both approaches expanded roughly the same number of nodes. We are investigating ways to reduce the nodes expanded when planning with the DM.

Finally, we compare the performance of agents that us a global DM to agent-centric DMs, maintained by each agent. We again report the average results over 50 scenarios. While agents could see other agents within a radius of 5, they only updated their local direction map with a radius of two.

Figure 4 compares the number of failed moves for the agent-centric DM case to the case with a global DM as well as the case with no DM. The figure shows that the local view increases the number of failed moves, which is to be expected because the agents' information about the world is incomplete. Although the number of failed moves with an agent-centric DM is higher than in the case with a global DM, it is better than using no DM.

5. CONCLUSIONS AND FUTURE WORK

This paper presents the idea of direction maps, which results in

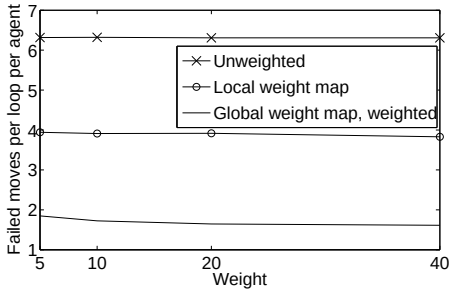


Figure 9: Failed moves per loop, per agent, for no weight maps, global weight maps, and local weight maps.

emergent cooperative behaviour without agents having to explicitly communicate their plans. As this is a preliminary study, there are many directions in which this research can be taken.

Direction maps work particularly well in an agent-centric environment, when agents have a very limited horizon of communication, as other approaches degenerate in this model. While it is clear that algorithms like WHCA* will degrade when the cooperation window is too small, we need to run larger tests to show this effect in practice. Another important question is what types of environments are best suited to each of the existing approaches for cooperative behaviour. We are also interested in understanding how direction maps might aid performance in *a priori* unknown worlds.

Finally, there are a number of enhancements and experiments which could be added to direction maps. For instance, the DM approach will fail when an agent is blocking a doorway and another agent needs to get through this doorway to get to its goal. This could possibly be solved by using direction maps along with WHCA*, since this algorithm allows for an agent to move out of another agent's way. There are a variety of different update rules that could be used as agents move through the environment, including a time decay on weights and update rules applied over a broader radius. This work is just a first step in exploring the possible uses of direction maps for inducing cooperative behaviour.

6. REFERENCES

[1] http://www.cs.ualberta.ca/ nathanst/hog.html.

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

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

[4] S. Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 22(4):109–131, 2001.

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

[6] D. Silver. Cooperative pathfinding. In *AIIDE*, pages 117–122, 2005.

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