Cooperative path finding in Multi Agent Systems

Summary by Andrei Paraschiv, Master AI, Sem II

1. Introduction

Multi-agent path finding involves the navigation of multiple agents from their starting positions to each individual end position (Goal) by circumnavigating obstacles and other moving agents. Pathfinding is a common task in modern AI implementations, game industry and AI research: planning the motion of robotic arms without colluding or monopolizing resources, planning the schedule of a railroad system, moving AI units in real time strategy games. Coordinating a fleet of agents by tracking all agents and computing all possible solutions turns out to be a PSPACE-Hard problem.
Multi-agent path finding can be divided into three problem classes: *Cooperative path finding* - each agent has the full knowledge of any other agent plan and know their planned future movements, *Non-cooperative path finding* - agents have no knowledge of each other plans and must predict their future movements and *Antagonistic path finding* - each agent tries to reach its goal and prevent any other agent to reach theirs. In this summary we will present cooperative path finding algorithms.
A possible approach is the decoupled (distributed) approach, by splitting the problem into *N* independent or weakly coupled tasks for each agent. The agent can then perform a greedy search for a path to its goal taking into account the states of other agents.
Many approaches are based on the classic A* algorithm (Hart, Nilsson, & Raphael 1968) that is widely used for single agent path finding, by rerouting on demand or improving the heuristic.

2. Problem definition

Most path finding algorithms assume a grid map topology, the map is discrete and composed of atomic locations named tiles, movement can take place in eight directions, four cardinal and four diagonal. Each tile can be accessible (traversable) or blocked (by a permanent object or by another agent). Each agent can occupy exactly one tile at a time, and one tile can host only one agent at a time. Agents can move only on adjacent tiles, instantaneously, from one time step to the next. A movement can be performed only if the target tile is free at the current time and no other agent plans to move in that tile. The distance traveled is considered 1 for cardinal moves and $\sqrt{2}$ for diagonal moves.

3. The Standard admissible Algorithm

The Standard admissible Algorithm is the classic A* algorithm with an *n-tuple* of grid locations for each of the *n* agents. The algorithm takes the moves of all agents simultaneously into consideration at any time step, so for for each state we have potentially 9^n legal operators (eight movements and one wait). In order to generate the legal operators we deploy a backtracking search to find the solutions for the constraint satisfaction problem (CSP) at every node expansion. The cost of each operator is the number of agents *not stopped* on their goals.
This method can be used to solve trivial problems, since each node expansion can generate ~59k nodes to the open list for an instance with 5 agents.

4. Decoupled Planning Algorithms

Decoupled planning algorithms solve the problem by planning the path for each individual agent separately, thus reducing the complexity of the search by searching lower dimensional spaces. This way we get results faster, but the optimality of the solution is not guaranteed.

5. Coupled Planning Algorithms

Coupled planning algorithms search solutions for all agents searching the joint configuration space, thus guaranteeing that an optimal solution will be found, if one exists.

5.1 Local Repair A* (LRA*)

In this family of algorithms, each agent searches the route to his goal by using the classic A* algorithm, not taking other agents into account (Stout 1996). Agents then start following their routes until a collision is detected. Whenever the agent encounters an occupied tile, it recalculates the remainder of the route. This approach can lead to cycles, causing agents to get stuck in an infinite loop. In order to avoid this, we can add an "*agitation level*" that is increased each time the agent is forced to reroute. Proportional to this *agitation level* we add some noise to the distance heuristic in order to escape the cycle.
There are some severe drawbacks to this algorithm - bottlenecks can take a long time to be resolved, and can take a high computational toll.

5.2 Cooperative A* (CA*)

In this algorithm (Silver 2005) the problem is split into a series of single agent searches. These searches are performed in the three dimensional space, by adding time as the third dimension. Also we add a "*wait*" state to the list of possible agent states, in order to allow the agent to remain stationary. Each computed agent route is added to a *reservation table* and these entries are then considered as impassable by the subsequent agents and therefore avoided. A simple implementation of the reservation table would be to think of it as a three dimensional grid (two spatial dimensions and one time dimension) and to hash the triplets (x,y,t) implementing a hash table that would be used to avoid any marked entries. In CA* we can use any admissible heuristic, for instance the Manhattan distance.

5.3. Hierarchical Cooperative A* (HCA*)

This method uses the Hierarchical A* (Holte et al. 1996), by computing the abstract distances on demand. The hierarchy is referring to the series of abstraction of the state space, each more general than the previous one. These hierarchies are not restricted to a spatial hierarchy. In HCA* (Silver 2005) we use a single domain abstraction, ignoring the time dimension and the reservation table. The abstraction is a simple 2 dimensional map with all agents removed. In order to reuse search data in the abstract domain, we can use the Reverse Resumable A* (RRA*)search in the abstract domain.
The RRA* executes a modified A* in the reverse direction. We start from the agent's Goal G and move towards the starting position S, continuing the search until a certain node N is expanded. This way we have the optimal distance from N to G once RRA* completes. For RRA* we use the manhattan distance heuristics. We use the RRA* algorithm to compute the abstract distance from N to G. This way if the node N is already expanded we can have the distance right away, if not we resume RRA* until we expand N.

5.4. Windowed Hierarchical Cooperative A* (WHCA*)

If the destination of an agent is for instance in a narrow corridor, it can block other agents once it reached its destination. This is why agents must continue to cooperate even after reaching the goal, so that agents can move out of the way in order to allow other agents to pass. Another issue we have to take into account is the sensitivity of the algorithm to the agent ordering. The third problem addressed by this algorithm is that previous mentioned algorithms must compute the complete route to the destination in the large three dimensional state space.
WHCA* (Silver 2005) uses a windowed search by limiting the cooperative search to a fixed depth limited by the current window. Each agent searches for a partial route to its goal and

starts following it. At regular times the window is shifted forward and a recalculation of a partial route ensues. In order to ensure that the agent is headed in the right direction, only the cooperative search is limited by the window, the abstract search is executed to its full depth. This way other agents are considered just for the next w steps, based on the reservation table and are ignored for the rest of the search. Once w steps have elapsed, all other agents are ignored and the search space is identical to the abstract space.
The windowed search will continue once the agent reaches its destination, and the agent goal will no longer be to reach the destination, but to complete the window via a terminal edge. Additional benefits can be achieved through the fact that the processing time is spread across agents, and the RRA* results can be reused for each consecutive window. An initial RRA* is performed from its initial position S to the goal G, any subsequent RRA* search is resumed to take into account any new nodes N encountered.

5.5 Operator Decomposition (OD)

Operator decomposition (Standley 2010) is a variant of A* that offers an improvement on the standard admissible algorithm. OD constructs neighbors of a single node incrementally, and by doing so it delays the instantiation of nodes that are not optimal according to path cost. The algorithm operates in a decomposed time space. In one timestep, each agent is considered one after the other, assigning moves to every agent sequentially. Until all agents have an assigned move, the nodes are thought of as intermediate nodes. Standard nodes are nodes with a move assignment for all agents at the current time step.
Expanding a start node S1 for an Agent A1 results in a first set of intermediate nodes, one for every possible move of Agent A1. These resulting intermediate nodes are put in the open list, sorted ascendant by the cost. We continue with Agent A2, base on the lowest cost node where Agent A1 has already been assigned a move. We avoid collisions with the new position of A1 while we ignore all other agents that have not yet been assigned moves. The next series of intermediate nodes consist in one node for every possible move for A2. When an operator is applied on the last agent, a regular node is generated.

5.6 Flow Annotation Replanning (FAR)

The Flow Annotation Replanning algorithm (Wang and Botea, 2008) introduces a flow annotated search graph that was inspired by two-way roads. The movement of the agents is restricted to one direction along a row or column, two adjacent rows or columns have different directions of movement - like a two way road. This way the whole map is covered by virtual roads. FAR attempts to solve the Multi-agent path finding problem with a lower requirement on memory and CPU compared to other algorithms.
Similar to WHCA*, the FAR algorithm uses the reservation table and combines it with flow annotations that lead to less expensive searches since we do not have to add the time

dimension to the search space. Each agent has to reserve its moves before executing them, but Agents do not incorporate these reservations into their search, but wait until the agents that block them have moved out of the way - like a traffic light on a crossroad. FAR tries to deal with deadlocks (agents waiting on each other indefinitely) by diverging temporary a *critical unit* off its planned trajectory, breaking up the deadlock. The *critical unit* is selected based on the density of the computed paths that pass through it, and should be continuously updated since by an agent passing, the density of that node decreases by one.
The FAR annotations are in fact changing the initial undirected search graph to a directed graph by imposing restrictions of directions to the edges and thus reducing the probability of a head-to-head collision among agents. The annotation must be done in a way that any node remains reachable from all nodes from which it was reachable in the original graph.
The new graph starts with no edges. The first row of edges is then connected via west bound edges, the second row via east bound edges, alternating directions. Similarly, we proceed with the columns by alternating north and south bound edges. Sink nodes (with only inbound edges) and Source nodes (with only outbound edges) are assigned diagonal edges adjacent to them. Corridor edges get bidirectional connectivity. We then perform a standard A* search in this restricted graph.

6. References

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.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In AAAI/IAAI, Vol. 1, 530–535.

Silver, D. (2005). Cooperative pathfinding. In Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment (pp. 117–122)..

Standley, T. (2010). Finding Optimal Solutions to Cooperative Pathfinding Problems. In Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI).

Stout, B. 1996. Smart moves: Intelligent pathfinding. Game Developer Magazine April 1996.

Wang, & Botea. (2008). Fast and memory-efficient multi-agent pathfinding. In Proceedings of the international conference on automated planning and scheduling (pp. 380–387).