Pathfinding and MAPF

Pathfinding or pathing is the plotting, by a computer application, of the shortest route between two points. This field is heavily based on Dijkstra's algorithm for finding a shortest path on a weighted graph, and in general, in graph theory.

It's a fundamental and important field in AI that has been researched extensively. Can be found in GPS navigation, robot routing, network routing, videogames and many combinatorial problems like puzzles as well.

Multi-agent Pathfinding (MAPF) is known as the pathfinding problem where multiple agents are involved. Given a start state and a goal state for each of the ageents, the task is to find minimal paths for the different agents while avoiding collisions. Some of its uses are in traffic control, aviation and video games.

There are 2 approaches for MAPF problems, the Decoupled approach and the coupled approach.

The Decoupled approaches are those in which paths for each agent is calculated separately. While these kinds of algorithms tend to be quite fast, not always is there a way to determine whether it's optimal or not. These algorithms are used when the agent number is large or the target is to find a nice solution, rather than a perfect one. A prominent example of a decoupled approach algorithm is Hierarchical cooperative A* (HCA*).

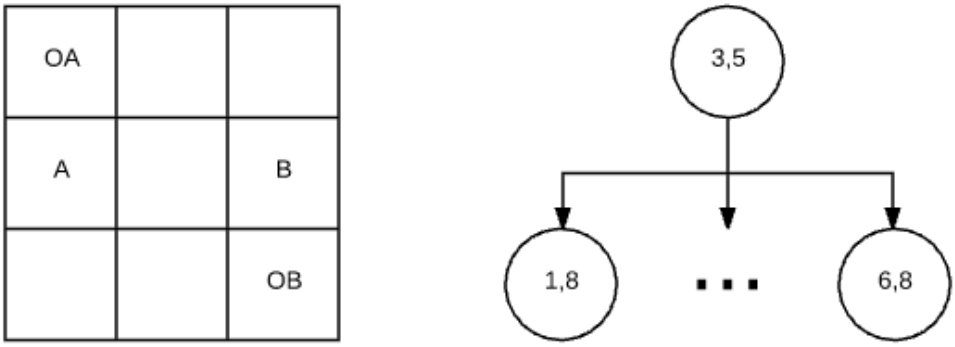
The Coupled approaches are generally those where the MAPF problem is formalized as a global, single-agent search problem, where paths are planned for all of the agents simultaneously. This approach tends to focus on finding the optimal solution to the problem, rather than finding it fast. The ICTS is one example of coupled approach algorithm.

A* in Multi Agent Search

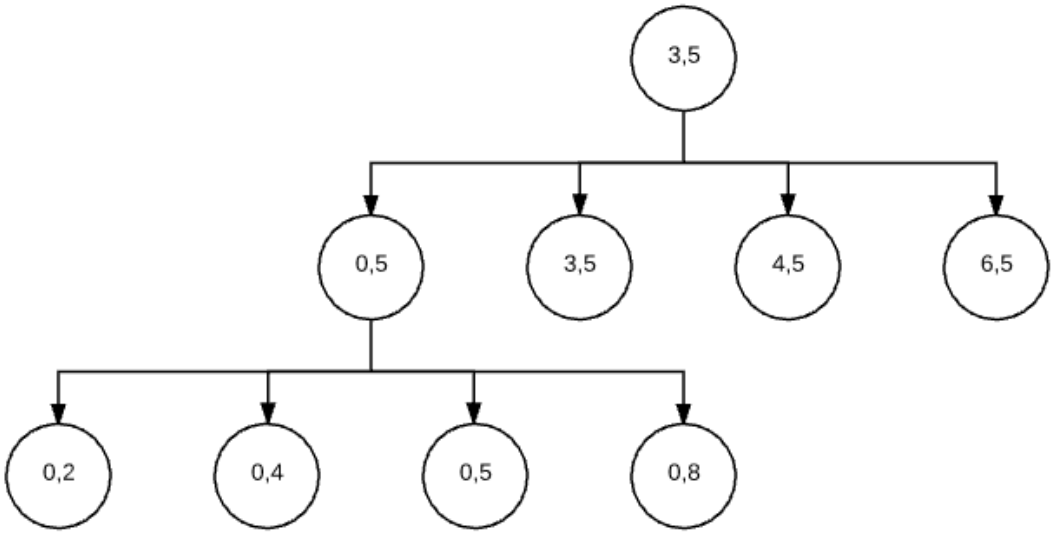
The A* is a graph search algorithm which finds always the shortest path between the origin node and an objective node. The A* algorithm uses an heuristic to reduce the amount of nodes that are expanded during the search. The algorithm always advances to the lowest value node, and this value is calculated with this formula: $f(n) = g(n) + h(n)$. Where $g(n)$ is the minimum cost found from the origin to the actual node, and $h(n)$ is the estimated cost from the actual node to the objective node.

To apply this algorithm in the multi agent problem we need to find an admissible heuristic for this case, an easy one could be the sum of all the distances between the actual position of each agent and their objectives.

Is possible to adapt the A* algorithm for Multi Agent path finding, in this case it will be expanded a node for each of the possible positions of all the agents that are involved. In this image we can see how will be the search tree for two agent searches:



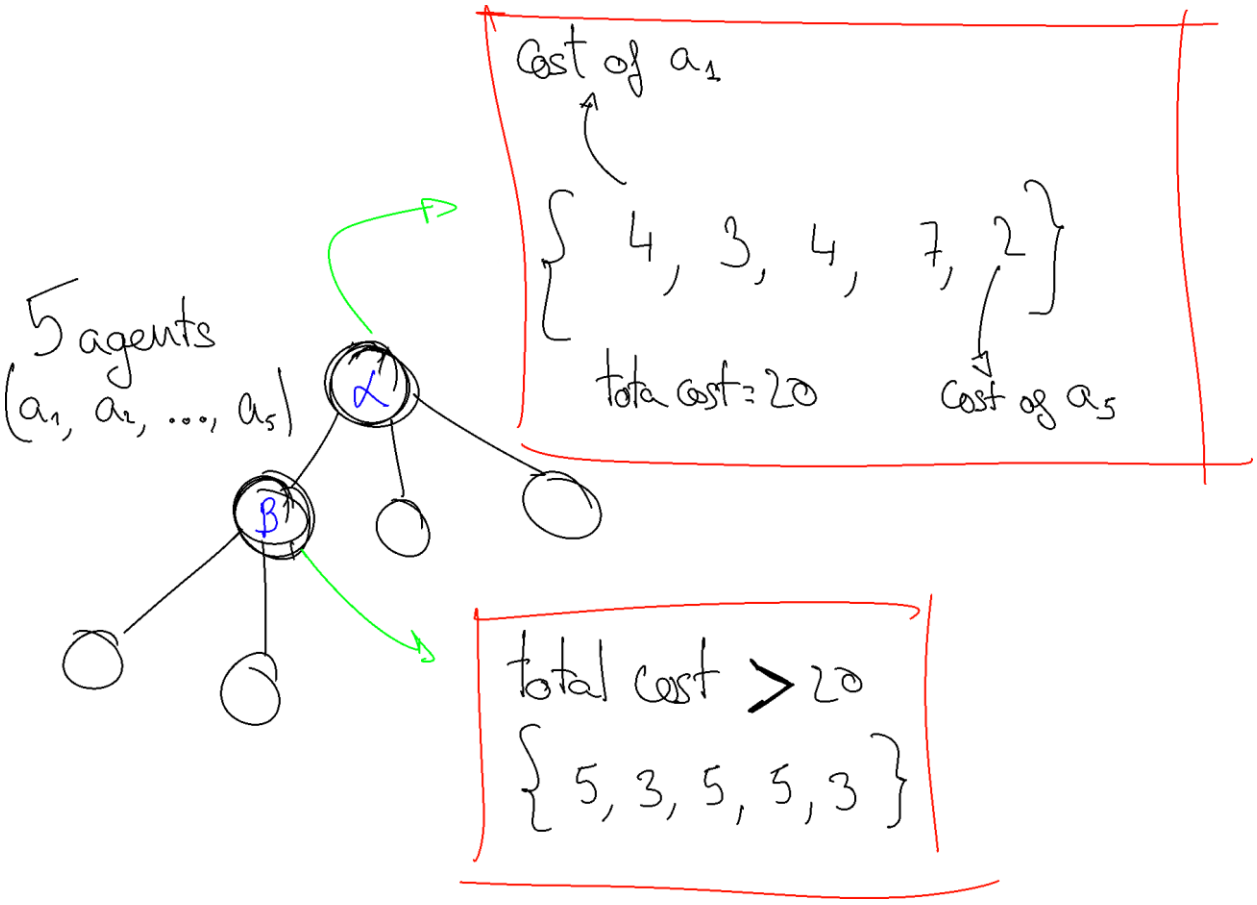
There is other possibility called **Operator Decomposition**, this method reduces the number expanded for each step. This is done by introducing intermediate steps between each of the full steps, each one of this steps will represent the movement of one of the agents. So in each step we will only expand 4 nodes, one for each of the possible locations of the agent taking into account.



ICTS (Increasing Cost Tree Search)

This algorithm is based on the understanding that a complete solution for the entire problem is built from a set of individual paths for different agents. It consists of 2 phases:

The high-level phase performs a search on the Increasing Cost Tree. Each node consists of a vector {C_1, C_2, C_3, …, C_k} where k is the number of agents in the problem. Each of these nodes represents all possible solutions in which the cost of the individual path of each agent A_i is exactly C_i. There is a unique node in the tree for each possible combination for costs. This tree is made in a way that the first node we find to be a solution, is going to be the optimal solution, that's because the cost of each node only increases the deeper you go.



For each of the nodes of the ICT, the low-level phase is invoked.

The low-level phase checks if there is a valid solution that fits the costs determined by the node. This is called for every single node until a node that is valid is found. In the example above, supposing that the node *alpha* is not a valid solution (the cause to this is probably that one or more of the agents break into each other), the high-level phase will go to the next node, *beta*, and will check if it is valid solution.

In order to agilicize the low-level phase, prior to the ICT search, we store all the possible way an agent has to go from the source to its objective, and how much is the cost, this way, the low-level phase can

know much faster if a solution is valid or not. The structure where this information is kept is called Multi-value Decision Diagram (MDD).

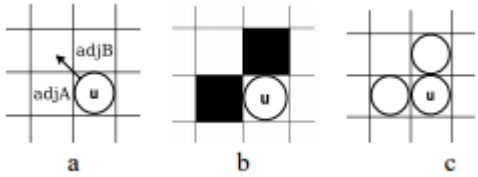
FAR (Flow Annotation Replanning)

FAR (Flow Annotation Replanning) is a method for multi-agent planning in grid maps. It is a new pathfinding method proposed by Ko-Hsin Cindy Wang and Adi Botea and uses A* algorithm in a multi-agent problem. This method was run on a collection of Baldur's Gate maps and was compared to WHCA*, another successful algorithm for multi-agent path planning.

Problem definition:

We assume that the topology is represented as a grid map. This map will be divided in atomic locations called tiles and they can have up to eight adjacent tiles, four cardinals and four diagonal. Each tile has two states, accessible and blocked and can only host one agent at a given time.

An agent can only move if the destination tile is accessible at the current time and no other agent is going to occupy it in the next step. For diagonal movements there is an extra constraint, an agent cannot move to its diagonal tile if its two adjacent tiles are blocked or occupied.



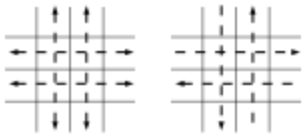
In a) the agent can make a diagonal movement because the adjacent tiles of the objective are free. In b) the adjacents are blocked and in c) they are occupied by other agents.

FAR method:

The goal is to achieve a scalable algorithm with low CPU and memory requirements.

FAR starts by preprocessing the grid map to abstract it into a flow-annotated search graph.

Grid maps usually are abstracted into undirected search graphs but having it directed gives us more control. After abstracting it additional rules are applied to preserve original maps connectivity so that any two locations reachable on the original map are still reachable in both directions in the directed graph. This resembles a driver navigating to the destination in a city since it has to obey traffic flow and take turns to reach its goal.



In a) we see a classical map and in b) an annotated map.

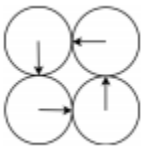
Just as the roads in real life, this design avoids head-on collisions between agents but side-on collisions can still happen in intersections so agents will have to communicate before moving. Our strategy to achieve this is to try to avoid replanning in the first place and if it has to be done we will try a local and cheap computation.

To reduce the chances of collisions keeping straight paths is favored. We achieve this modifying A* so that it favors the next tile that maintains the current path.

To provide communication between agents we make them know each other's intended moves. All agents make reservation for k steps and when each one of them moves those k steps another reservation has to be done. A unit will always make k moves except when the target node is less than k steps away. To reserve node(i) the agent has to be able to reach it at step i-1 and it can't be already reserved by another agent in the same step. This can make certain nodes very busy because multiple agents may want to use it. In these cases we first allow horizontal movements and then vertical movements alternating between the two each step. This creates a traffic light effect.

When a unit arrives to its target, it stays there using wait actions unless it blocks another agent's path. In these cases the agent takes a step away from its target and uses A* to return to the position where the block occurred.

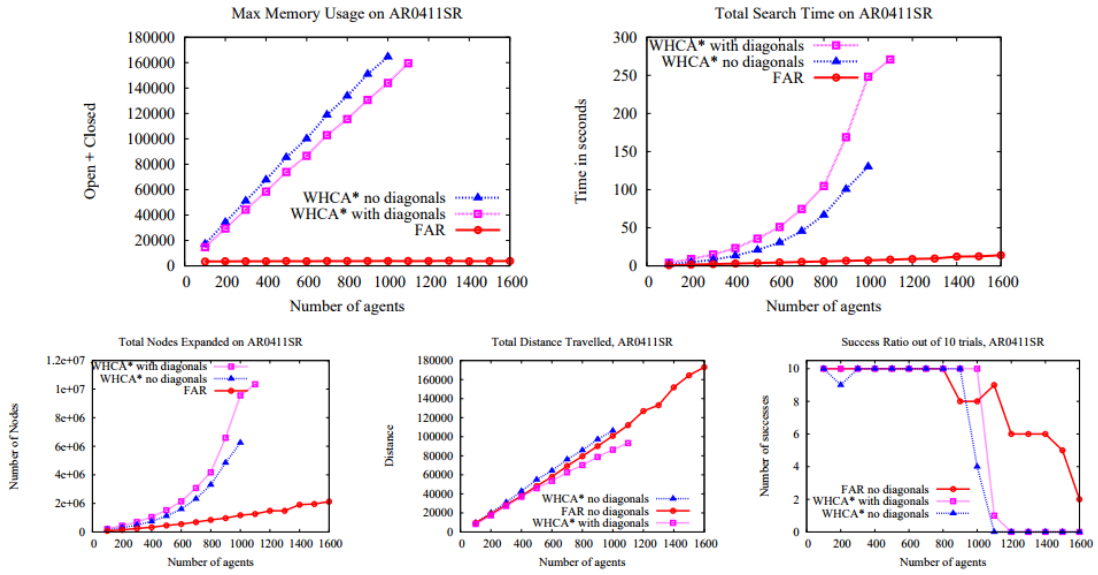
Replanning can also happen if an agent is critical in breaking a deadlock. A deadlock is a situation where agents wait for one another creating a cycle.



First, we need to identify where the deadlock is and which agent is the critical one. The critical agent is the one that is placed in the highest density node. The density of a node is calculated using a heuristic. The critical agent will take a step out of the deadlock and will replan how to get to that same exact location. This will take at least three steps and the rest of the agents will be able to move.

Conclusion:

FAR uses approaches such as flow annotation and local plan repair measures to solve difficult multi-agent pathfinding problems. It has some problems with single-with tunnels but in most cases it is quicker and uses less resources than WHCA*.



Comparison between WHCA* and FAR performance. From left to right, max memory usage, total search time, total node expansion, total distance travelled, success ratio out of 10 trials.

Bibliography

A search algorithm on Wikipedia*

Fast and Memory-Efficient Multi-Agent Pathfinding (Ko-Hsin Cindy Wang and Adi Botea)

Control flow graph on Wikipedia

Pathfinding on Wikipedia

Visual demo on Pathfinding

Thesis Summary: Optimal Multi-Agent Pathfinding Algorithms (Guni Sharon)

The increasing cost tree search for optimal multi-agent pathfinding (Guni Sharon, Roni Stern, Meir Goldenberg, Ariel Felner)