Multi-Agent Pathfinding and Variants within ILP framework

Veli Can ERDEM

January 4, 2017

1 Introduction

Multi-Agent Pathfinding problems consist of finding a solution for a path that satisfies the constraints specified within the problem. In general they have a starting position and a goal position and the solution provides the path between the start and the goal. This paper will first introduce MAPF with ILP solution, then provide a brief mention of the possible constraints in MAPF, how they are implemented in ILP and the completeness of the solution, then it will provide a experimental data on a specific problem to show the ILP solution's effectiveness.

2 Multi-Agent Pathfinding

The most general MAPF problem consists of finding a path between the initial position of agents and their respective goals. This is solved by finding an equivalence between a flow problem and the MAPF problem as shown in (**yulavalle15b**). The brief mention of the solution will be provided here.

Defining the problem A network $N = (G, c_1, c_2, S)$ consists of a directed graph $G = (V, E)$. $(c_1, c_2) : E \mapsto \_Z$ specifies the capacities and costs of edges, respectively and $S \subset V$ are the source and sink vertices. $S = S^+ \cup S^-$, respectively source and sink sets. For a v $\in$ V, $\delta^+(v)$ and $\delta^-(v)$ denotes respectively the set of edges going to the vertex and the set of edges leaving the vertex. A feasible (static) S^+ and S^- flow on a network is a map $f : E \mapsto \_Z^+$ that satisfies such constraints:

- Edge Constraint: $\forall e \in E : f(e) \leq c_1(e)$
- Flow Constraint at Non-terminal Vertices: $\forall v \in V \setminus S$

$$\sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)} f(e)$$

- Flow Constraint at Terminal Vertices:

$$\sum_{v \in S^+} (\sum_{e \in \delta^-(v)} f(e)) = \sum v \in S^-(\sum_{e \in \delta^+(v)} f(e))$$

This formulation is for a single flow problem. A multiflow problem considers multiple flows going to multiple sinks. Consider flow function f^i for each commodity i.

- Edge Constraint: $\forall e \in E, \forall i, \sum_{f_i(e)} \leq c_1(e)$
- Flow Constraint at Non-terminal Vertices: $\forall v \in V \setminus S \forall i$

$$\sum_{e \in \delta^+(v)} f_i(e) = \sum_{e \in \delta^-(v)} f_i(e)$$

- Flow Constraint at Terminal Vertices:

$$\forall i \sum_{v \in S^+} (\sum_{e \in \delta^-(v)} f_i(e)) = \sum v \in S^-(\sum_{e \in \delta^+(v)} f_i(e))$$

An MRP instance can be given as $(G, R, x_I : r_i \mapsto s_i^+, x_G : r_i \mapsto s_i^-$, To create the multiflow instance from MRP, the graph is expanded both by space and time, to instantiate the interaction of robots. In order to create the ILP instance, time is assumed fixed. Time expansion is made by creating 2T+1 copies of G's vertices, for example for a v it creates v(0),v(1),v(1)'...v(T),v(T)'. If there is an edge from v_i to v_j in the original graph, edges are created such that:

$$0 \leq t < T(v_i(t), v_j(t+1))$$

There are also edges created in between every two successive copies of v such that (v(0) v(1), v(1) v(1)'...

Loopback edges are created between $(x_G(i), x_I(i))$ to ensure that a robot will reach the goal point from the start point.

Suppose that for the constructed graph, x_{ij} denotes the robot i passing through edge $e_j \in G'$. There are n robots. From $1 \leq i \leq n$, e_i denotes the robot i's loopback edge. Following constraints hold:

$$\forall e_j, \sum_{i=1}^{n} x_{ij} = 1$$

$$\forall e_i 1 \leq i \leq N i \neq j, x_{ij} = 0$$

$$\forall v \in G', 1 \leq i \leq n \sum_{e_j \in \delta^+(v)} x_{ij} = \sum_{e_j \in \delta^-(v)} x_{ij}$$

Objective function is $\max_{\sum_{i=1}^{n} x_{ii}}$

In the article there is also a collusion detection mechanism mentioned, which is to make sure they won't collide head-on.

Consider the set such that

$$C = \{e_i, e_j : e_i = (v_k(t), v_l(t{+}1)), e_j = (v_l(t), v_k(t{+}1)), e_i, e_j \in E', 0 \leq t < t, (e_i, e_j)\}$$

$\forall c \in C, e_i$ and e_j holds such a constraint:

$$e_i + e_j \leq 1$$

$$e_i + e_j = 2$$

designates a head-on collusion.

3 Possible constraints of MAPF within ILP

3.1 Must go vertices

It is very easy to define whether an agent r must reach or should not reach to a vertex. Consider the vertex v ∈ G.

$\forall v \in V \sum_{t=0}^{T} \sum_{e_j \in \delta^+(v(t))} x_{ij} > 0$ considers v to be a must-go destination by r

$\forall v \in V \sum_{t=0}^{T} \sum_{e_j \in \delta^+(v(t))} x_{ij} = 0$ considers v to be a must-not-go node of r

Since aforementioned constraints do not conflict for $v_i, v_j \in V, v_i \neq v_j$ it can be used for every vertex.

Another constraint could be to limit the number of agents going to vertex v, regardless of the agents identity. This can be done by creating a constraint that designates whether an agent goes to a vertex v_k. The number of agents that is wanted to go to vertex v_k is g.

$\forall v_k$ such that $v_k \in V \sum_{t=0}^{T} \sum_{e_j \in \delta^+(v(t))} x_{rj} \geq c_{rk}$

$$c_{rk} \leq 1$$

$$\sum_{\forall c_{rk}} c_{rk} = g$$

The last constraint can be modified in order to limit agents such that at least g agents can go to v_k by modifying "=" with "≥".

If the vertex would be constrained such that at most g agents can go to a vertex another constraint should be used.

$$\forall v_k \text{ such that } v_k \in V, 0 \leq t \leq T, \forall e_j, e_j \in \delta^-(v(t))$$

$$x_{rj} \leq c_{rk}$$

$$c_{rk} \leq 1$$

$$\sum_{\forall c_{rk}} c_{rk} \leq g$$

3.2 Meeting points

Meeting problem is defined as finding a minimal path such that every agent will all reach to one of specific meeting points at the end. A complete problem can be defined within ILP such that it will provide a best solution from all possible meeting points.

Instead of goal points, meeting points are defined (x_G is changed to V_M). Most important difference is that loopback edges will be created from meeting points every starting point. For the ease of understanding, the loopback edges will be designated as e_{mn} such that $1 \leq m \leq (M)$ and $1 \leq n \leq N$ where M is the total number of meeting points and N is the total number of agents. e_{mn} is an abstraction for $e_{m*N+n-N}$

$$\forall e_i, 1 \leq i \leq N, i \neq j, x_{ij} = 0$$

will be replaced by:

$$\forall e_{mi}, 1 \leq m \leq M, 1 \leq i \leq N, i \neq j$$
$$x_{mij} = 0$$

$$1 \leq m \leq M$$
$$\forall e_{mi}, 1 \leq i \leq N$$
$$e_{mi} = e_{mj}$$

and

$$1 \leq i \leq N, \sum_{m=1}^{M} x_{mii} = 1$$

Above constraint can also be satisfied with only one instance satisfying it due to the first constraint which is satisfied when all loopback edges meeting points have equal flow passing through.

One thing to be considered is how will meeting vertices work within this algorithm. There are many options:

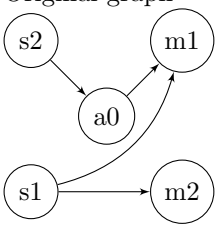
- An agent can't go out of a meeting place it entered.
- Only one agent can exit from a meeting place at a time instant.
- Agents can exit from a meeting place at the same time.

- Create a subgraph in meeting place having at least n (number of agents) vertices, that will represent the final state of agents when this place is designated.

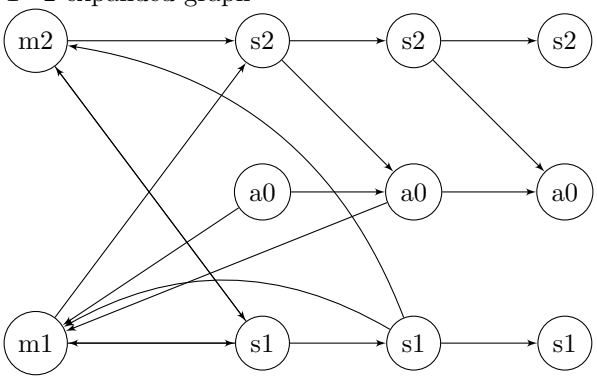
The option to select depends on the problem it will be used. If edges designate long streets, cars going into parking lot doesn't need to be in a queue in order to enter the park since it is negligible. However airports exceptionally need to track when a plane will enter or exit. In this solution, the entrance to meeting place will be assumed negligible and meeting places will be assumed as in option 1.

$$\forall v_m, v_m \in V_M, e_j \in \delta^+(v_m) f(e_j) = 0$$

Original graph



T=2 expanded graph



In this graph $f((m1, s1)) = f((m1, s2))$ and $f((m2, s1)) = f((m2, s2))$. There is also the sum constraint:

$$f((m1, s1)) + f((m2, s1)) = 1$$

This already satisfies $f((m1, s2)) + f((m2, s2)) = 1$

Notice that there isn't any need to create $v_m(t)$ for $v_m \in V_M$ in this formulation. Also because it is implicated in ILP formulation, (v_i(t), v_i'(t)) edges are omitted.

4 Performance of the algorithm

The algorithm consists of a graph, designated agent starting vertices and designated meeting points. In the implementation, a graph is designated by its length and its width. For ease, in the experimentation the length and width are equal, resulting in a square-like graph. On top of this graph, obstacles are placed (impassable vertices) using a ratio that is between 0 and 1. Then on this graph, agent starting vertices and meeting vertices are placed. In overall 4 variables are needed to generate a random graph: length/width, obstacle ratio, number of agents and number of meeting vertices.

It is a good thing to note that the vertices that are obstacles do not generate any constraint.

In order to find the starting T and to know whether the problem has a solution, a reachability check is made. If the meeting points are not reachable, a new graph is generated with same variables. If they are reachable, it stores for each meeting vertex the longest distance between itself and agent and the minimum of these distances is used for the solution.

It is trivial to see if a meeting point is reachable by every vertex, there is a solution.

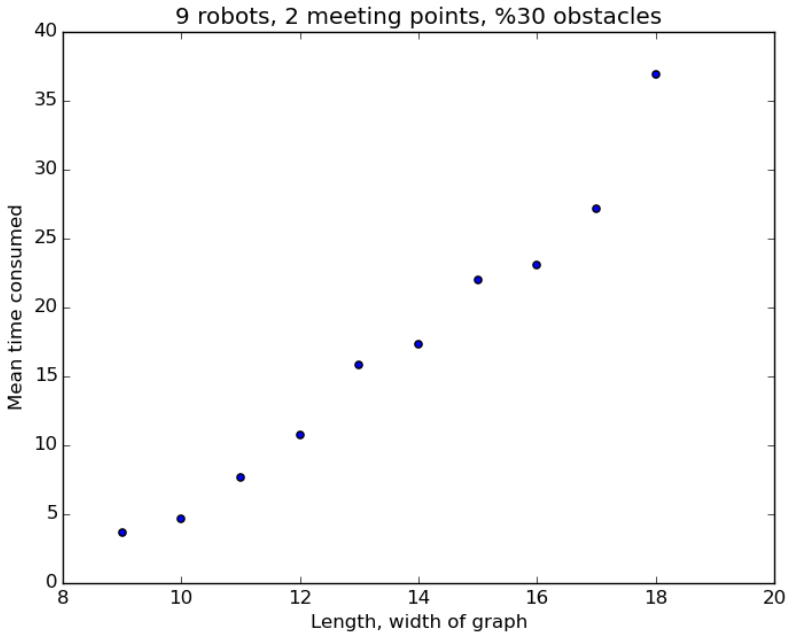
The analysis will only work the problems in which a solution is generated. It solves for optimal solutions, which means it doesn't use non-optimal heuristics.

4.1 Performance comparison by graph size

Consider the length of the graph as a variable l.

All graphs such that $9 \leq l \leq 18$ and having 9 agents, 0.3 of the graph being obstacles and 2 meeting points are analysed.

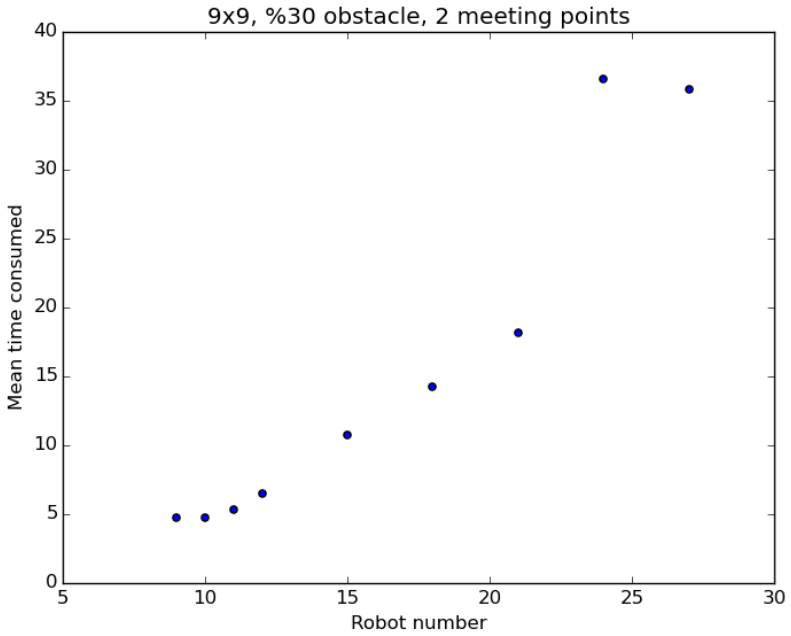
Theoretically the number of constraints is increased proportionally with the size of the graph, since for each general vertex a constant amount of constraints are generated.



Similar analysis using other variables resulted in a performance that is directly proportional to the size of the graph.

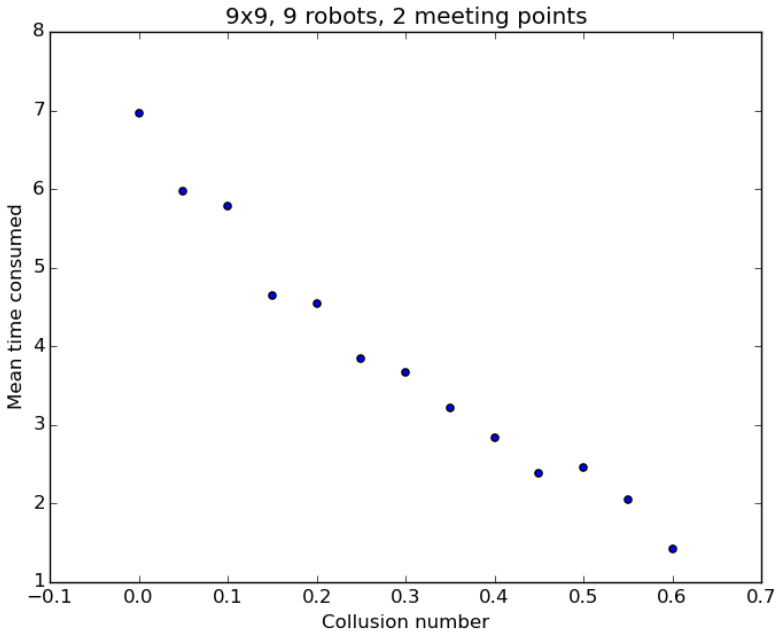
4.2 Performance comparison by number of agents

A graph of 9x9 size is analysed with the 0.3 obstacle ratio and 2 meeting vertices.



As it can be seen, while the increase has been linear when N $<$ 20, it grows exponentially after it. N = 27 means that robots occupy the half space of the graph. N = 30 gives 89 solve time, which is the double of N = 27.

4.3 Collusion test



As it can be seen, obstacles decreases the solve time linearly. Because of the reachability check, it may be due to the fact that solvable graphs have easier solutions. To check whether it solves better than other algorithms when there is a high obstacle ratio, the same graphs must be analysed with another algorithm.

References

[1] Jingjin Yu and Steven M. LaValle, *Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics*, 2015b.