Individual Research Module: Plan Merging

Supervisors

Klaus Strauch

Torsten Schaub

Etienne Tignon

Aurélien SIMON

February 14, 2023

Contents

List of Figures

1 Introduction

Multi-agent pathfinding or MAPF [16–18] for short, is a fundamental AI problem that has a wide-range real application: GPS, video-games, routing, planning, traffic control etc. In few words, consider agents moving in an environment, going from an initial to a goal position; MAPF is about finding a path for each agents, such as their is no collision between them. Multiple approaches exist; Search-algorithm (CBS [15]) or reduction solving based ([3]). In this report we focus on an approach call "Plan Merging", this approach aims to solve MAPF problems by using two distinct steps; Computing a set of paths for each agents independently (which correspond to classic pathfinding / single-agent pathfinding [6]). We will refer to this step as Individual Path Finding, in short IPF. The second step, to find a solution avoiding collision using the previously computed paths.Formalization of each step will be described in their respective sections. The interest in this approach is due to pathfinding complexity which is lower than MAPF complexity [13]; the idea is then to use this property to expect saving computation time or space complexity. The approach in its definition is close to CBS, however, the planning part of CBS stops if a conflict occurs and re-iter the planning part considering the conflict previously encountered which is not the case for the IPF; the conflict handling would be in the merging section which could be a CBS algorithm for instance. The work achieved tries to formally define these two steps but also tries to provide workflow and different approaches with their formal definitions. The report includes a background part which formally introduce the notion of MAPF and also some definitions and notions that will be used in the whole report. The report then includes the Individual Path Finding section describes formalization of different cost or objective function for paths such as, functions based on the intrinsic property of a path, functions based on the path in the graph and functions based on other agents paths. The report then describes formalization of Plan Merging by introducing a witness solver, definitions, heuristics and approaches. And then, finally conclude.

2 Background

2.1 MAPF

The following definitions of Multi-Agent Path Finding (MAPF) follow the ones in [9]. MAPF is a triple (V, E, A) where V, E denotes a connected graph, V being a set of vertices and E a set of edges connecting them. Then A being a set of agents. For each agents $a = (s, g) \in A$, s is a vertex in V denoting the starting location and g is also a vertex in V denoting the goal location. We consider that every starting position and every goal position are disjoint. For each discrete time step $t \in \mathbb{N}_0$, an agent can either; wait at its current vertex or move to a neighboring one.

The output for MAPF problems is a a plan. A plan being a collection

$(\pi_a)_{a\in A}$ of finite walks in (V, E) where each walk π_a is represented by a finite sequence of adjacent or identical vertices in V from s to g for agent $a = (s, g)$. We use $\pi_a(t) = v$ to denote that agent a is located at vertex v at time step t. As consequences, for each $a = (s, g) \in A$, we have $\pi_a(0) = s$ and $\pi_a(|\pi_a| - 1) = g$ (where $|\pi_a|$ gives the length of walk π_a). Generally, for any $a = (s, g)$ and any $0 \leq t \leq |\pi_a| - 1$, we have $\pi_a(t) \in V$. In addition, we also have $(\pi_a(t), \pi_a(t+1)) \in E$ with $0 \leq t < |\pi_a| - 1$

A plan is considered as *valid* if, taken pairwisely, walks are collision-free. A vertex conflict occurs whenever two different agents occupy the same vertex at the same time step. Formally, we have $conflict(a, a', t)$ if given any $a, a' \in A$ and $t \in \mathbb{N}_0$, we have $\pi_a(t) = \pi_{a'}(t)$. An edge conflict (or swapping conflict) occurs whenever two agents exchange their position or are using the same vertex at the same time, which implies that edge conflict is defined on time step t and $t - 1$. We have $conflict(a, a', t)$ if given any $a, a' \in A$ and $t \in \mathbb{N}_0$, we have $\pi_a(t - 1) = \pi_{a'}(t)$ and $\pi_a(t) = \pi_{a'}(t - 1)$.

A plan $(\pi_a)_{a\in A}$ has a conflict, if a conflict (a, a', t) occurs in $(\pi_a)_{a\in A}$ for some pair $a, a' \in A$ of agents and a time step $t \in \mathbb{N}_0$.

Sum-of-costs and *makespan* of a plan $(\pi_a)_{a\in A}$ are respectively defined as such; $\sum_{a\in A}(|\pi_a| - 1)$ and $\max_{a\in A}(|\pi_a| - 1)$.

Furthermore, we also define in which MAPF problem specification the following work has been conducted. Since we would define object that require distance and/or coordinates such as rectangle or circles, we will consider that graphs have cartesian coordinate system, which means we can represent them as a grid. In addition we work on a non-anonymous MAPF variant with edge conflict and vertex conflict.

3 Individual Path finding

This section describes functions that can be used as cost/objective function. This means it can be maximized, minimized in order to obtain paths with different properties.

3.1 Formalization

Individual Path Finding (IPF) aims to compute for each agent, a non-empty set of paths. IPF can be summarized as "MAPF without collisions"; it takes the same input as MAPF (see 2), it however differ on the output. The output is a triple (V, E, τ) where τ is a set where the relation is bijective, considering an agent $a \in A$, we have $\tau[a]$ being a non-empty set of path, we can refer to $\tau[a]$ with γ or γ_a if necessary for better understanding or lighter notation. Let an agent $a = (s, g)$, for any $\pi \in \gamma_a$, $\pi(0) = s$ and $\pi(|\pi| - 1) = g$. Thus, every paths composing γ can have a different length.

We will then list and classify what can be the different kind of paths that can compose τ, what are their properties, advantages and inconvenient.

We also define the *pick* function which allows lighter notation; *pick* takes as input a set of paths γ and returns one of the paths in γ.

3.2 Properties of a Path

3.2.1 Path of defined length

A noticeable property is the **length** of the path, denoted as $|\pi|$. We can distinguish two cases; the length is the minimal possible or it is strictly higher than the minimal. The first case gives us a lower bound for searching a solution, and if a solution exists using only shortest path, it is an optimal solution. In the second case optimality is no longer guarantee. However, working with path of defined length, combined with knowledge of the graph, it can make the "field of action" for the merging algorithm wider, which can result of finding a solution, but also rise the number of possibilities which can lead to more computing difficulties.

Definition 1 (Shortest Path (SP))**.** Given an initial and a final position, an associated path π is in the set SP if π is valid and no path π' such as $|\pi| > |\pi'|$. Generally, SP the set of all of the shortest path possible for a given initial and final position. For lighter notation, $|SP^i|$ to the size of paths π composing SP

$2k$ paths are specific paths of defined length, they are of length $|SP| + 2k$ where k is a positive integer. The $2k$ section represents the fact that in order to "resolve" a conflict, if wait moves are not possible for any reason (specification, approaches, instance...), solving a conflict would require at least two moves. This kind of paths comes with an assumption; computing an additional path of $2k$ kind is (way) faster than trying to solve a conflict in the solver. Intuitively, this kind of paths can be quite interesting to solve Blocking agent instances.

3.2.2 Path containing bending(s)

Counting bending in a path require to work on a graph using cartesian coordinate system. From coordinates we can compare angles of the vectors made from the current vertex and the previous one, to the current one and the next one. If angles are equals, it means that the path is continuing in its previous direction (angles have the same direction), otherwise a direction change is identified. Formally, we can define a bending in an cartesian coordinate system of the path π at time step t as such:

$$u = \overrightarrow{(\pi(t-1), \pi(t))} \text{ and } v = \overrightarrow{(\pi(t), \pi(t+1))}$$

$$bending(\pi, t) = \begin{cases} 1 & \text{if } \widehat{u} = \widehat{v} \\ 0 & \text{otherwise} \end{cases}$$

Then, the number of bending in a path is

$$Bending(\pi) = \sum_{t=1}^{|\pi|-2} bending(\pi, t)$$

If a "waiting" move is considered as a vector of size 0, by definition, any vector of size 0 is "perpendicular" to any other one.

It is important to mention that approach for counting bendings makes sense for a grid scenario with defined and restricted directions; there high chance that bendings definition should be changed for scenarios working with continuous directions or a 3+ dimension. Given the number of bending, we can deduce information; with zero bending, we can trivially deduce that the initial position and the goal position are on the same column or row, furthermore the path is in SP.

3.3 Path using information coming from the graph

In this section, we will describe what information we can deduce from a path in the graph individually.

3.3.1 Degree of the path

The degree of the path is calculated by using the in and out degree of vertices (which is the number of edges linked to the vertex) that the path is going through; formally, we have for $v \in V$ and

$$degree^+(v) = \sum^{\forall v \in V} \begin{cases} 1 & \text{if } \exists (v, v') \in E \\ 0 & \text{otherwise} \end{cases}$$

$$degree^-(v) = \sum^{\forall v' \in V} \begin{cases} 1 & \text{if } \exists (v', v) \in E \\ 0 & \text{otherwise} \end{cases}$$

We can then calculate the degree of a path;

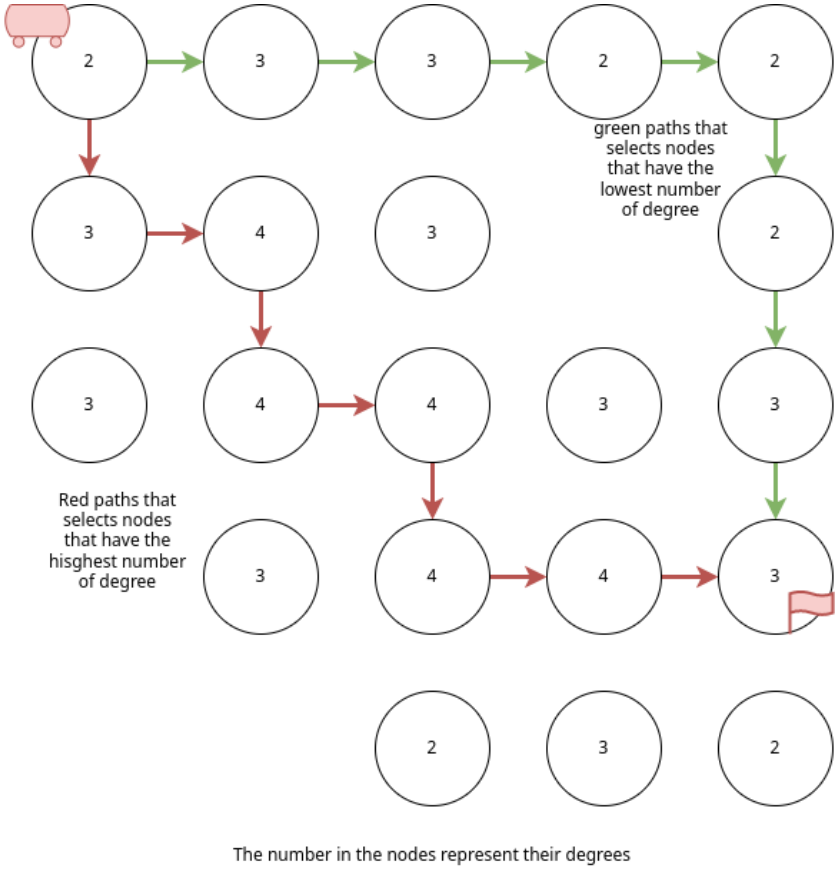
$$Degree(\pi) = \sum_{t=1}^{|\pi|-1} degree^+(\pi(t)) + degree^-(\pi(t))$$

Knowing this information, we can tell if a path is "going next to wall" or more "going through the middle" of the environment. We can of course imagine using only in or out degree information. $Degree$ works for a directed graph, notion of in out degree get lost for a non-directed graph, however we can easily change $degree^+$ and $degree^-$ by

$$degree(v) = \sum^{\forall e \in E} \begin{cases} 1 & \text{if } v \in e \\ 0 & \text{otherwise} \end{cases}$$

Figure 1 shows an example of two paths for a single agent. The red path is obtained by selecting nodes that have the highest degree while the green path is obtained by choosing nodes that have the lowest degree.

Figure 1: Example of paths based on the degree of vertices



3.3.2 Coverage of the graph

Coverage of a graph represents the proportion of the graph that is used by an agent. We can define graph coverage as such:

$$coverage(\pi) = \frac{|vertex(\pi)|}{|V|}$$

Graph coverage definition changes given the input;

$$coverage(\gamma) = \frac{|\bigcup^{\pi \in \gamma} vertex(\pi)|}{|V|}$$

Finally,

$$coverage(\tau) = \frac{|\bigcup^{\gamma \in \tau} \bigcup^{\pi \in \gamma} vertex(\pi)|}{|V|}$$

where $vertex(\pi)$ is a function that returns the associated vertices set of the sequence of vertices.

3.4 Selection of one path considering other agents

3.4.1 Path conflict

Potential conflict represents a function counting conflict occurring between agents considering time step. It takes as input a set of path τ; formally we define first

$$pc(\pi, \pi') = \sum_{t}^{min(|\pi|,|\pi'|)} \begin{cases} 1 & \text{if } conflict(\pi, \pi', t) \\ 0 & \text{otherwise} \end{cases}$$

Where $conflict$ returns the boolean value "true" if any conflict occurs between π and π' at time step t, return "false" otherwise. Furthermore, pc does not forbid π and π' belonging to a same γ.

We can then compare sets of paths:

$$pathc(\gamma, \gamma') = \sum_{\pi,\pi'}^{\forall \pi \in \gamma, \forall \pi' \in \gamma'} pc(\pi, \pi')$$

where γ an γ' are different sets of path, typically coming from $\tau[a_{\in A}]$ and $\tau[a'_{\in A}]$. We can then define

$$PathC(\tau) = \sum_{a,a'}^{\forall a \neq a' \in A} pathc(\tau[a], \tau[a'])$$

Another interesting function would be to compare a single path to a set of path γ or τ, indeed comparing all of the set of agents to all others, if their respective set of paths are large, the meaning of the comparison becomes blurry. We can in consequences define other functions that aims to keep interesting result not matter how much paths it takes as input:

$$pathc(\gamma, \pi) = \sum_{\pi'}^{\forall \pi' \in \gamma} pc(\pi', \pi)$$

and

$$pathc(\tau, \pi) = \sum_{\gamma}^{\forall \gamma \in \tau} pathc(\gamma, \pi)$$

Another point to mention is that the definition is not complete; if two paths have different length, a part of the bigger path will not be considered

3.4.2 Blocking agent

An agent a is called a "blocking agent" towards a', or agent a is blocking agent a', if at time step t, all the next vertices from $\pi_a(t)$ to $\pi_a(|\pi_a|-1)$ are included in the vertices from $\pi_{a'}(t)$ to $\pi_{a'}(|\pi_{a'}|-1)$, for instance see figure2 where red agent, at time step 1, is blocking the blue agent. Formally we have:

$$blocking(\pi, \pi', t) = \begin{cases} true & \text{if } since(\pi, t) \subset since(\pi', t) \\ false & \text{otherwise} \end{cases}$$

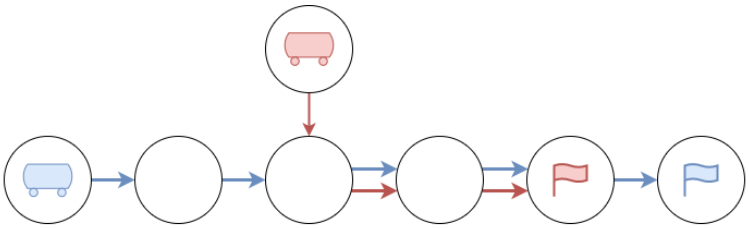
(We read "path π is blocking path π' at time step t"). Where the function *since* is define as:

$$since(\pi, t) = \bigcup_{i=t}^{i \to |\pi|-1} \{\pi(i)\}$$

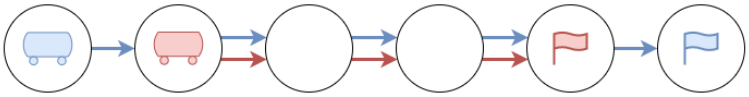
which represent the set of vertices in π from t until the end of the path .

It is called "fully-blocking" if a is a blocking a' at time step 0 (also see figure2).

Figure 2: Example of blocking agent



Example of red agent blocking blue agent a time step 1



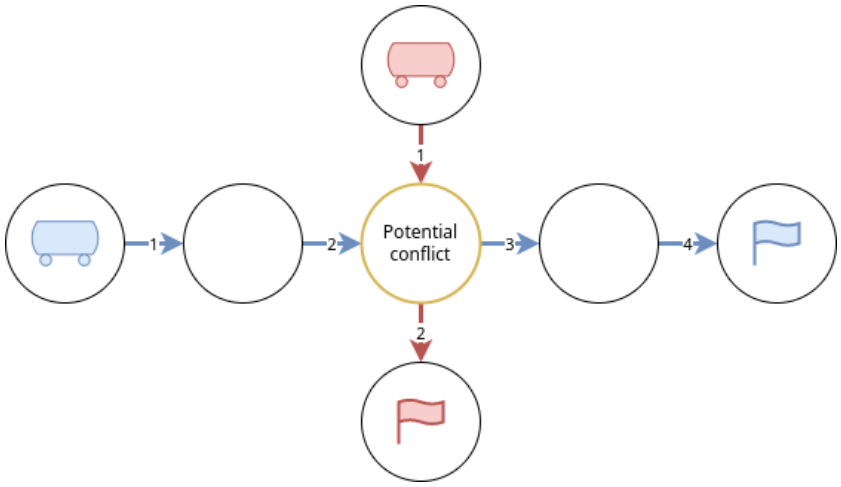
Example of red agent fully-blocking blue agent

3.4.3 Potential conflict

Potential conflict aims to count conflict occurring between paths without considering time step, in other words it delineate the union of common vertices between a all paths. it differs with path conflict by the fact that path conflict are defined according to time step, which is not the case for potential conflict. By definition, path conflict are specific cases of potential conflict, you can see

as instance the figure 3 which represent an isntance where no collision would occur, however there is a potential conflict represented by the yellow node.

Figure 3: Potential conflict



Formally, in order to define potential conflict, we have to define intermediate functions:

$$potc(\pi, \pi') = |vertex(\pi) \cap vertex(\pi')|$$

$$potentialc(\gamma, \gamma') = \sum_{\pi \in \gamma, \pi' \in \gamma'} potc(\pi, \pi')$$

$$PotentialC(\tau) = \sum_{a,a'}^{\forall a \neq a' \in A} potentialc(\tau[a], \tau[a'])$$

We do not separate vertex conflict and edge conflict since it is by definition, included in the calculus; considering an edge conflict occurring a time step t (and $t+1$) between paths π and π', by definition, it means that $\pi(t) = \pi'(t+1)$ and $\pi(t+1) = \pi'(t)$. Thus, *potc* functions will count an edge conflict as two potential conflict. However, it could be interesting to separate potential edge conflict from potential vertices conflict.

3.4.4 Conflict window

A conflict window is defined with a couple of vertices $(v, v') \in V$ in a graph with orthonormal coordinate system, we also introduce the function *coordinate* : $V \to (\mathbb{N}, \mathbb{N})$ which gives the coordinate associated to a vertex. Let C be a set containing all conflicts (these can be path or plan conflict), formally it can be define as

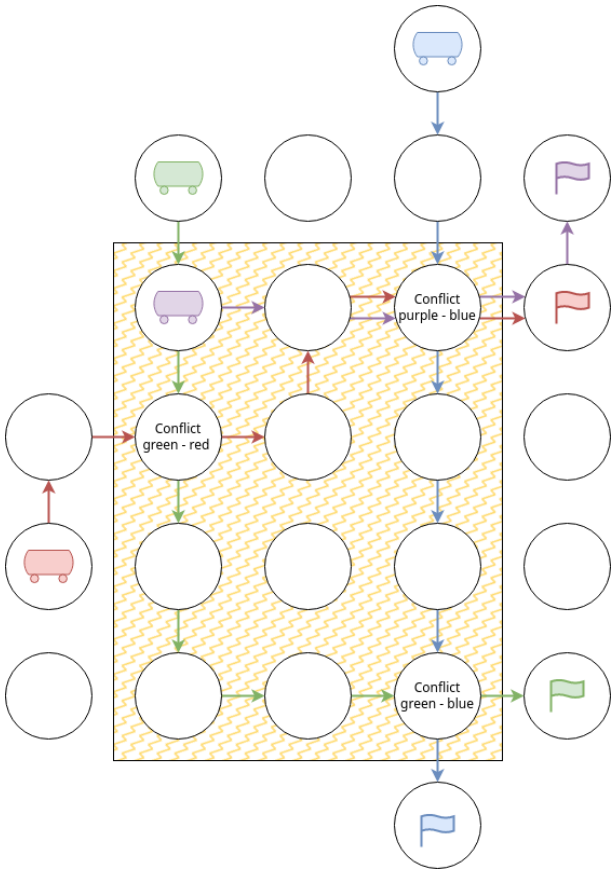
$$C = \{v | v \in V, conflict(v)\}$$

where $conflict$ is a boolean condition; true if a conflict exist at vertex v, else false. The next step is to "draw" a rectangle (see Conflict window figure). To do so, we use the the coordinates of vertices in C to construct this "rectangle". A conflict window is a set CW of vertices define as such:

$$CW(C) = \{v | v \in V, min_x(C) \leq x(v) \leq max_x(C), min_y(C) \leq y(v) \leq max_y(C)\}$$

We described the approach to build one conflict window, however we can easily imagine building multiple by dividing the set of conflict into multiple subset. We however require that taken pairwisely, $CW \cap CW' = \emptyset, CW \neq CW'$; we could end-up with overlapping conflict window, which reduce the usefulness. The example in figure 4 shows four agents where three path conflict occurs. It also shows the resulting conflict windows; we can see that angles of the conflict window are not necessarily path conflict.

Figure 4: Conflict window



3.4.5 Diversity

Computing diverse path is mainly done using two different diversity measure; based on the Jaccard similarity coefficient (see [7]), or on the the symmetry difference (also called Hamming distance) (see [8]) of two sets, we will respectively refer to them as $diversity_j(\gamma)$ and $diversity_\triangle(\gamma)$. Both compute diversity among two sets. The Jaccard similarity is defined as the size of the intersection of two sets divided by the size of the union of the two sets. In other words, it is a measure of how similar two sets are, based on the number of elements they have in common. we use them for computing diversity among two set of vertex issued from paths. The Hamming distance is a measure of the difference between two sequences of equal length. It is defined as the number of positions at which the corresponding symbols are different. The difference between the is that the Jaccard index focuses on the number of nodes that two paths have in common, while the Hamming distance focuses on the number of differences between the two paths. If we adapt the notation from the papers, we have

$$diversity_j(\gamma) = \sum_{\forall \pi, \pi' \in \gamma} JaccardCoefficient(vertex(\pi), vertex(\pi'))$$

$$diversity_\triangle(\gamma) = \sum_{\forall \pi, \pi' \in \gamma} SymmetryDifference(vertex(\pi), vertex(\pi'))$$

3.4.6 Distant

Let γ being a set of path of length n, where all of the path have the same initial and ending position. Let $\pi, \pi' \in \gamma$ two different paths. π and π' are considered "most distant" if no other path in γ have a higher distance. Distance between two paths is computed as such :

$$distance(\pi, \pi') = \sum_{k=0}^{n} dist(\pi(k), \pi'(k))$$

An intermediary function $dist(V, V) \to \mathbb{N}$ is necessary; it represents an arbitrary distance function between two vertices, it can be the euclidean distance, length of the shortest path between them... From *distance* definition, we can then define the distance from a path to a set of path γ:

$$Distance(\pi, \gamma) = \sum^{\pi' \in \gamma} distance(\pi, \pi')$$

Considering then a set of k distant paths $DP \subset \gamma$, $\pi = pick(\gamma)$, $\pi \notin DP$, π is distant towards DP (which means that if we add π to DP, DP is still distant) if no $\pi' = pick(\gamma), \pi' \neq \pi$ exist such as $Distance(\pi, DP) < Distance(\pi', DP)$.

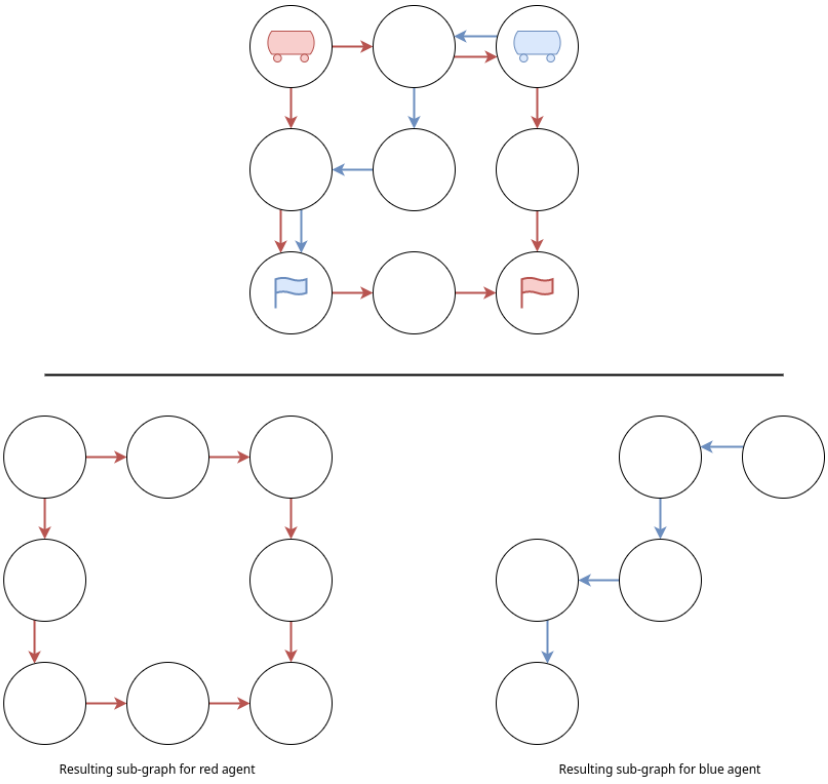
4 Merging

In this section we will define some approaches and heuristics in order to solve MAPF problems given a τ input. The expression "solving a conflict" (or similar) will occurs, it however do not represent anything formal; it represent a moment where an approach has a solution to reach a vertex that is after the conflicting vertex, in the sequence of vertices pi. Thus, it does not mean that no conflict may occurs because of the resolution.

4.1 Sub-graphs

Approaches in this section are using sub-graphs; one sub-graph is a graph based on G with the scope of a set of paths γ; for every γ in τ we have an associated sub-graph $\mathcal{S}(\gamma)$. In others words, n agents result in n sub-graphs. For instance, let an agent a_{red} and a_{blue} with their set of paths γ_{red} and γ_{blue} where $|\gamma_{red}| = 2$ and $|\gamma_{blue}| = 1$. The result of the decomposition of τ in sub-graphs is in figure 5.

Figure 5: Decomposition of τ in sub-graphs



Sub-graphs by themselves are not useful, approaches can extend the sub-graphs according to strategies. Intuitively, an "extension" represent additional

possibilities in order to solve a conflict.

4.1.1 Witness solver

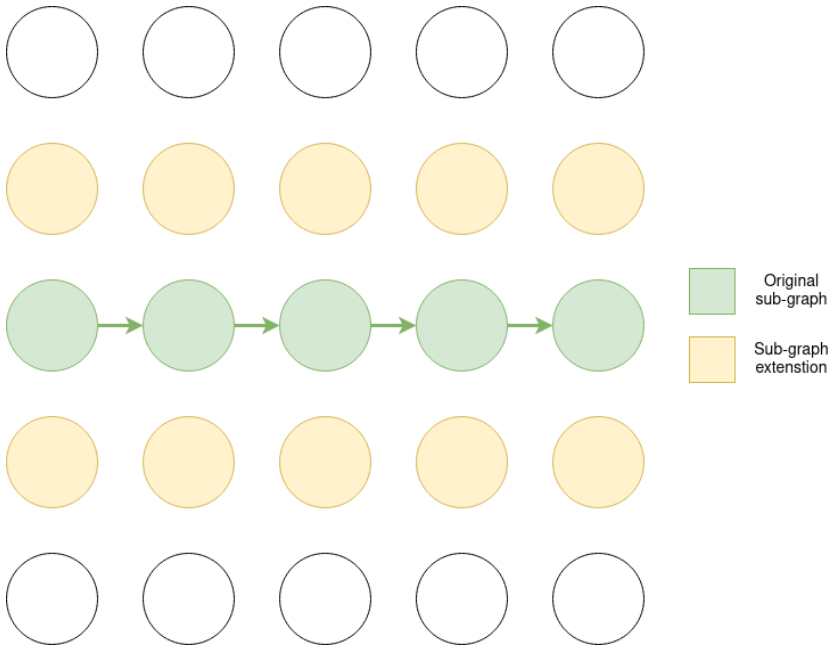
In this section, we will introduce a witness solver; a witness solver (WS) will be the basic solver that aims to solve MAPF instance given a τ input. It works as such; it converts the input τ in a set of n graph that are sub-graph of G as described in section 4.1. We then have two cases: there is a valid plan made out from τ that contains no path-conflict, or there is no valid plan that can be build ot of τ directly. In the first case, the merging process will give the valid plan as output, in the second case, the WS will try to solve the instance by using different approaches, strategies or heuristics until finding a valid set of paths that is a valid plan.

The first case occurs if it exists a set $\{pick(\gamma)|\gamma \in \tau\}$ which correspond to a collection of path, is conflict-free, we refer to this case as "straightforward solving".

4.1.2 Corridor

The corridor approach aims to help the solving process by extending the size of the sub-graph generated by a set of path γ. It extends the sub-graph by adding to the sub-graph the direct neighbor vertices (and their associated edges). Applying the corridor extension successively x times will result of a corridor of level x for instance:

Figure 6: Example of corridor



We can define corridor as such;

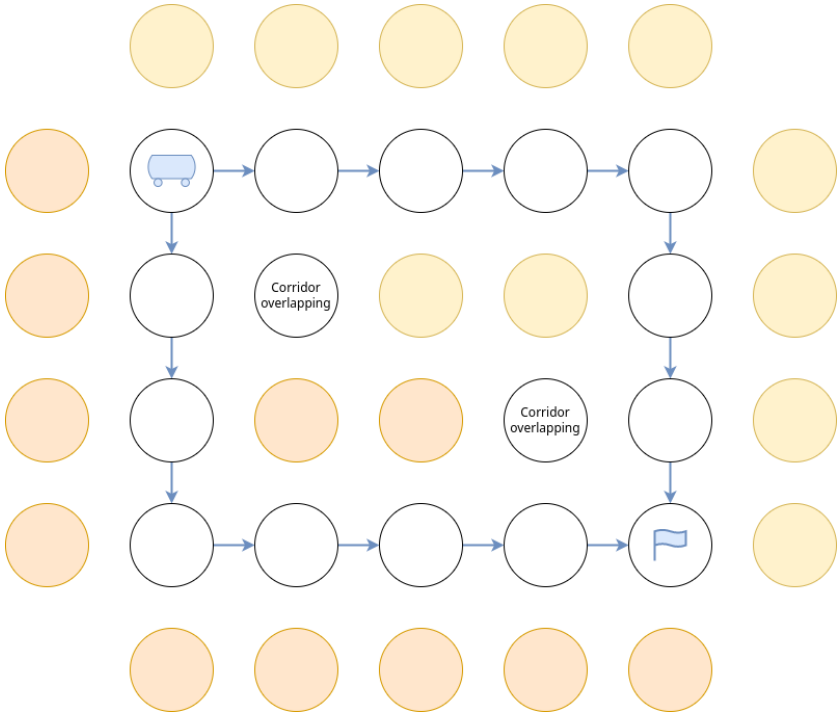
$$corridor(s, (V, E)) = \{v' | v' \in V, v \in s, (v, v') \in E\}$$

where s represent of set of vertices.

A corridor of k-level(s) is obtained by recursively use the corridor function k time on its output: e.g. $corridor(corridor(vertex(\pi), G))$ represent a 2-levels corridor. We can assume some elements about this approach; with a big enough k size the whole graph is covered. Using the witness solver would end up doing classic MAPF. Furthermore for large instances, a huge part of the extension would probably not be used ending up adding more space search for the witness solver. We can also identify instances where corridor extension would be probably interesting to use when the "dodge" movement required to solve a conflicts is situated in the graph "far away" (a distance relatively high) from where the conflict occurred. We can also assume that a set of distant paths would be an interesting set to work with; it potentially limits the number of "overlapping" vertices that would be added by the extension, see example shown in figure 7. Also, corridors may work better on low density instances and low number of conflicts and potential conflict since it is not adding so much depth of possible movement.

In practice, we can create corridor only for path that have a path conflict, we can also choose to create corridor only for one of the two agent involved in a conflict.

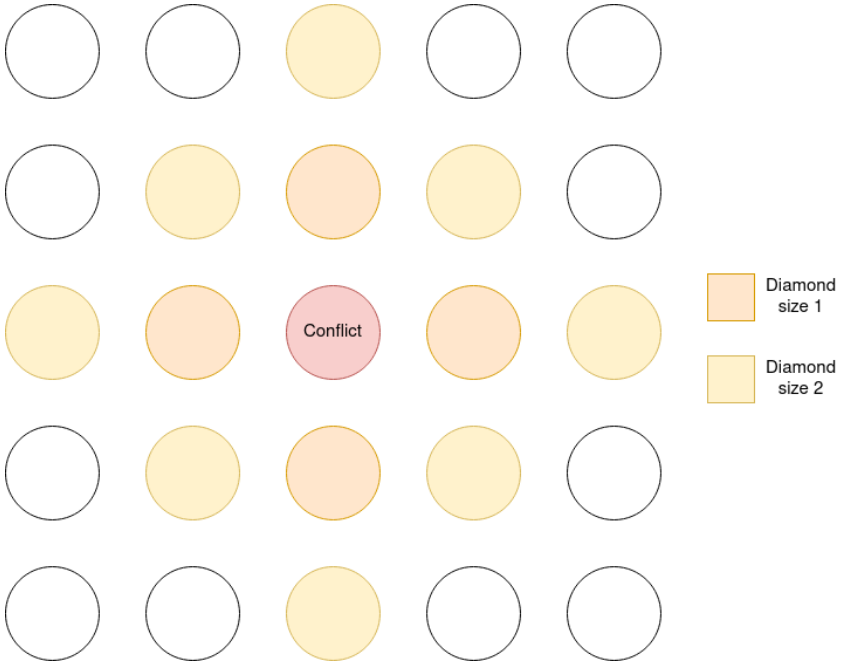
Figure 7: Most Distant paths and corridor extension



4.1.3 Diamond extention

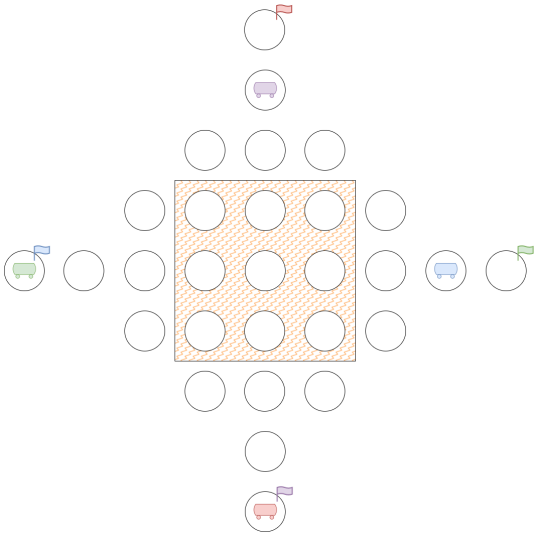
The diamond extension aims to extend the sub-graph by adding diamonds of vertices around potential conflict or plan conflict. As shown in figure 8, we can se different levels of diamond extension which will increase the size of the sub-graph, by extension the possibilities in order to solve the conflict.

Figure 8: Example of diamond of size 1 and 2



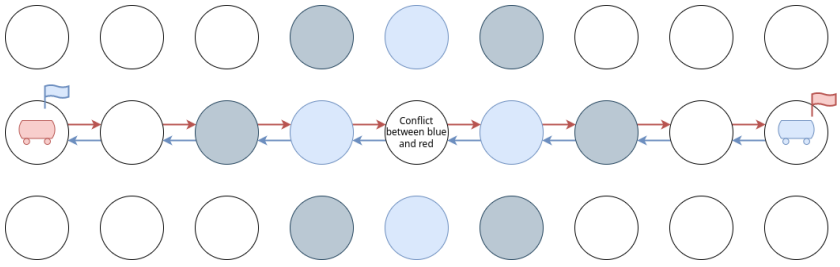
As described for the corridor instance, having a k as diamond size too big would end up covering the whole graph. This approach would probably works fine with a relatively small Conflict window; having all conflicts concentrated in area of the graph would allow small size of the diamond extension, which reduce the space search. The instance represented in figure 9, has a "small" conflict-window. While solving this instance (or this kind of instance, lot of robots, small conflict window) we can easily imagine that solving the conflicts issued by the IPF would create other conflicts but still in the same area. In consequences, diamond extension seems appropriate since the extension is covering also close area around conflicts.

Figure 9: Example of diamond extension interesting case



From the properties described in section 3, we can argue that paths that have few bendings would work better for the diamond extension; a higher proportion of the extension (of size higher than 1) will already be in the original paths. Furthermore contrary to the corridor extension, whenever the proportion of conflict is low compared to the length of the path, diamond extension would probably work better e.g. figure10, based on section 4.1.2 and figure 6 the proportion of the sub-graph that is not used is way higher for the diamond extension.

Figure 10: Example of case where diamond extension would work but not corridor extension

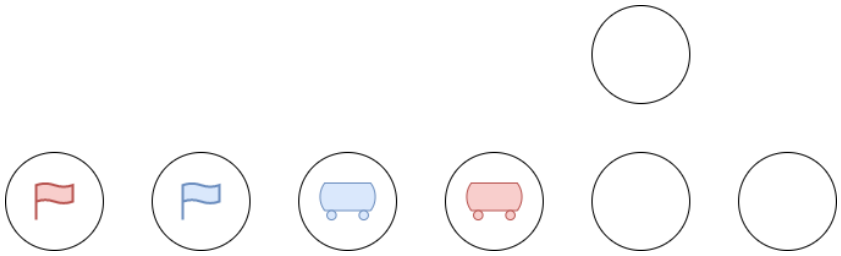


4.1.4 Systematic extention

Approach presented above are not complete and aim to reduce the space search or computation time at the probable cost of optimality. Furthermore we can imagine instances where both approaches would not work, or require too much computation time. In the case represented on figure 11, considering that τ contains for both agent one shortest path, it is easy to tell that both heuristics

on this kind of instances would not work decently because it would require that the extension covers the whole graph.

Figure 11: Example instance that are more complex to solve



We can use solving strategies described in paper [9, 19]. These paper introduce four different strategies; Baseline Strategy, Makespan-add Strategy, Prune-and-cut strategy and the Combined strategy. For short, these strategies are using a $k - restrictedgraph$ which is similar to sub-graph; it takes a shortest path SP to define a subgraph, it also add "vertices that are at most distance k away from some vertex in SP". The main idea is to add vertices and/or raise the makespan and see if a solution exist. According to the benchmarks, the Combined strategy seems to to most successful.

5 Heatmap

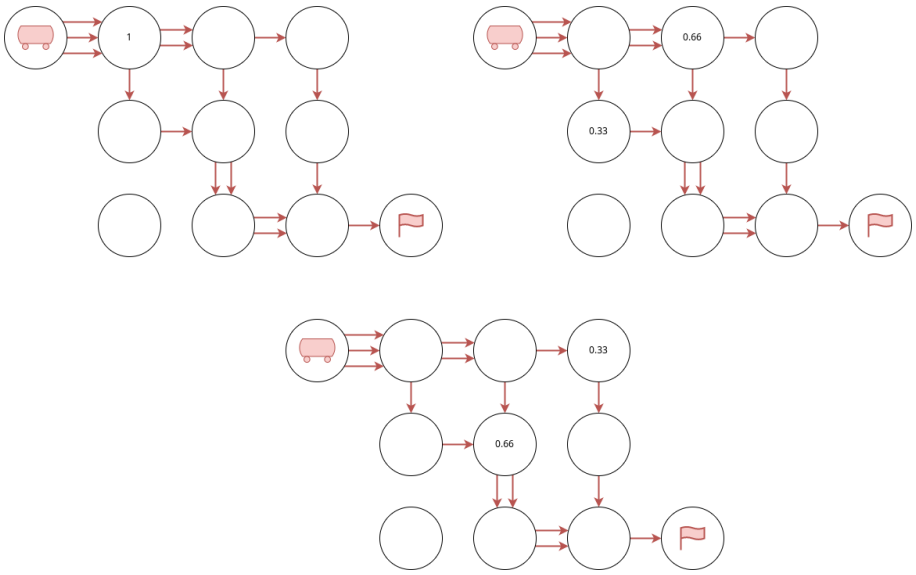
Based on paper [1]

Heatmap is about projecting likelihood of presence of agents on vertices. Likelihood refers to the chance of an agent to be positioned at a specific vertex of the graph at a time step t. We introduce $\phi(a, v, t)$ which compute likelihood of the vertex v at time step t for the agent a. To do so, we introduce a set P that is created as such :

$$P = \bigcup^{a \in A}$$

From this definition, we can compute the first three step of an agent with an associated $|\gamma| = 3$ below (figure 12).

Figure 12: Heatmap example of the first three time step for a $|\gamma| = 3$



This heatmap-per-agent can be used to compute a "global one", to do so, for each vertex and for each time step, we sum each ϕ and then divide it by the number of agent (see appendix 16), we will refer to this as Φ where $\Phi(v)$ refer to the value of a vertex v. Computed this way, global heatmap do not represent likelihood of presence anymore but more of an indicator[1] where both heatmap taken together can be useful on different part of plan merging resolution.

5.1 Heatmap as a property

The heatmap we proposed is from the perspective of agents, but it can also be considered with regards to the γ's scope. To do so we define the following sequence (which is not modifying the computation of heatmap)

$$fun(\pi_a) = \{(v, \phi(a, v, t))|(v, t) \in enum(\pi_a)\}$$

$$funbool(\pi_a, T) = \bigcup_{(v,t)\in enum(\pi_a)} \phi(a, v, t) \leq T$$

We have then have a property that can be used as objective function among other in 3. Affecting a specific threshold to the function $funbool$ can result in having different consequences; for instance selecting T as a value of 0.4 would have as impact that γ size is at least 3. Furthermore it also means that if a specific time step t, two paths π and π' belonging to a γ having $\pi(t) = \pi'(t)$ (we will refer to this as "crossing", it do not refer to a strong formal notion),

[1] I believe that we can find a way to compute global heatmap that could give us directly if the resolution is possible straightforward

this will result in forcing the size of γ of at least five ($2/4 = 0.4$, where "2" is the number of agent on the same vertex at the same time step t and "5" is the number of agent). This example illustrate that considering heatmap (with a path scope) can have strong consequences, and gives strong indication on paths that are composing γ, we used the number of path and the number of possible "crossing". However threshold for heatmap can influence the coverage (of an agent) 3.3.2 of the graph (maximizing/minimizing ϕ would directly change the number of conflict and then, the coverage of the graph), diversity 3.4.5 and distance 3.4.6[2].

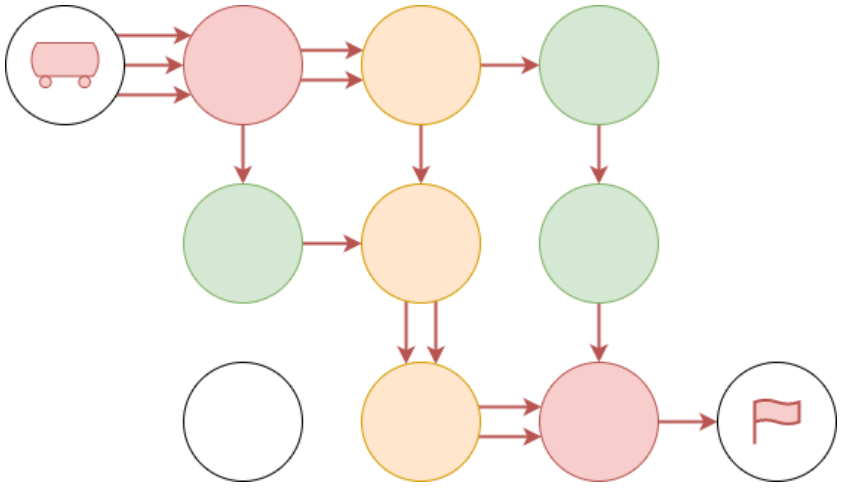
In addition, in the perspective of τ (which is the same principle of γ), depending on the value of T, we interferences with the number of potential conflict and path conflict, it may gives us information considering a specific interval of time step, which can influence conflict windows ans so on.

To conclude, heatmap as a property of a path can have a wide use cases and give us various information given very few requirements.

5.1.1 Heatmap as a comparison tool

Global heatmap as we define it may also be a way to compare τ together, respectively agent heatmap to another possible γ for a same agent. We can use a visual way to represent and compare different γ or τ. Even thought it it might not be the most reliable way, but it may be useful in order to fast compare multiple ones, for instance, the figure 13 shows how the complete figure 12 (or the figure 16) could look like with some coloration.

Figure 13: Proposition of coloration for an agent heatmap, gradient from white (no presence), green (few presence) to red (high presence)



[2]Property may of course influence each other, however, as we mention, we have with heatmap a direct influence on certain other property

Furthermore, we formulate the following assumption *"having a heatmap (global or agent one) that have the lowest sum of ϕ and the lowest maximum ϕ possible is desirable"* do not sound like a costly assumption. Given this assumption, we can then order different γ or τ heatmaps. It of course, necessitate that the assumption is true, which may be (partially) decided with experiments.

5.2 Heatmap for subgraph

5.2.1 Straightforward solving

In merging section 4, we introduce "straightforward solving" which describe the case where we can find a combination of paths in τ where a valid conflict-free set of paths can be found. However the process of picking these paths may be costly due to the exponential numbers of combination. However we can use heatmap as a preprocessing work; from a global heatmap, we can directly tell from the paths if a straightforward solving is not possible if, with $|A| > 1$, for every vertices v in the heatmap, the following equation is true $\Phi(v) > \sum(a|v \in \gamma_a)/|A|$. (Is the reciproque true ?) The following figure 14 shows an example where the straightforward solving is decided from the heatmap; the orange vertex as a Φ value superior to $\sum a|v \in \gamma_a/|A| = 3/4 = 0.75$. On the other side, the figure 15 show the opposite case.

Figure 14: Example of instance that is not possible to use straightforward solving

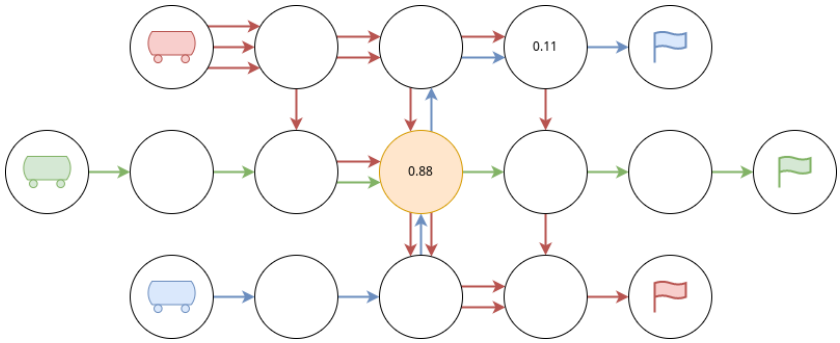
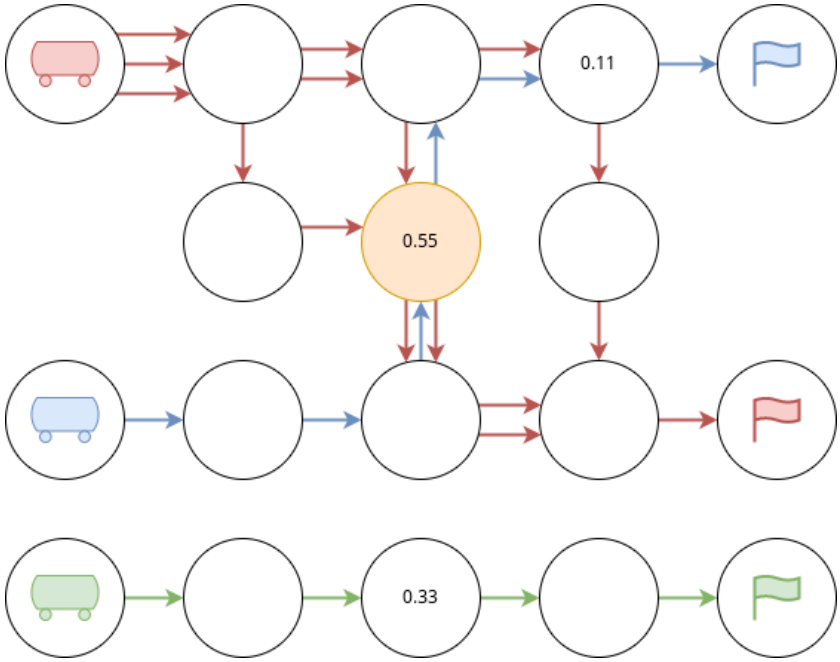


Figure 15: Example of instance that straightforward solving is at least undetermined



5.2.2 Pruning the subgraph

5.3 Heatmap as a solving strategy

5.3.1 Misc

- X Can help for making τ ; The way we define ϕ can be an objective function; trying to minimize the sum of every ϕ
- Comparing τ ; It seems that it is a visual and easy way to compare two different τ, compare to other properties of paths that we defined, this one seems the most appropriate for comparison.
- Can help on the decision if the straight forward resolution is possible.
- Helping for the subgraph subgraph; we can remove part of the subgraph for some agent where the global heat indicator is high and sees if ot relax the problem or not (keeping a connected subgraph). It can also probably provide information on which vertex could be added instead
- Solving by Guiding the agent
- Suit easily for robustness

Straightforward resolution refer to the fact that among all path in τ, we can find a combination of paths (1 for each agent) that is an actual solution for the problem. (explaining what is the process)

Guiding the agent

6 Future work & Conclusion

To conclude, in this report we described and define what could be plan merging, starting by defining and formalize what are the two main part; Individual Path Finding and Plan Merging. We described for IPF various functions that could use as cost or objective functions in order to build interesting τ. Then we described different solving strategies that would take as input any τ. Multiple similar approaches and definition [10, 11] exists but do not obviously separate the two steps that I tried to define, which was my main objective.

The whole report introduce different function, heuristics and strategies that could be implemented in order to be tested and benchmarked in future work(s). Furthermore, in the future, we would like to develop a framework which allows to visualize, test, and benchmark different IPF approaches, different *tau* building and merging strategies. In addition, due to the lack of time, multiple merging or solving strategies have not been explored such as "guiding the agent via heatmap", "guiding the agent with time interval"[12, 14], taking into consideration robustness [1, 2], using highways [4, 5]. And finally explore the selection of strategies and heuristics given a MAPF problem instance. All of these topics are potential future addition to this report.

7 References

[1] D. Atzmon et al. "Probabilistic Robust Multi-Agent Path Finding". In: *Proceedings of the Thirtieth International Conference on Automated Planning and Scheduling (ICAPS'20)*. Ed. by J. Beck et al. AAAI Press, 2020, pp. 29–37.

[2] D. Atzmon et al. "Robust Multi-Agent Path Finding and Executing". In: *Journal of Artificial Intelligence Research* 67 (2020), pp. 549–579.

[3] R. Barták and J. Svancara. "On SAT-Based Approaches for Multi-Agent Path Finding with the Sum-of-Costs Objective". In: *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*. Ed. by P. Surynek and W. Yeoh. AAAI Press, 2019, pp. 10–17.

[4] L. Cohen et al. "Improved Solvers for Bounded-Suboptimal Multi-Agent Path Finding". In: *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. Ed. by R. Kambhampati. IJCAI/AAAI Press, 2016, pp. 3067–3074.

[5] Liron Cohen and Sven Koenig. "Bounded Suboptimal Multi-Agent Path Finding Using Highways." In: *IJCAI*. 2016, pp. 3978–3979.

[6] Daniel Foead et al. "A systematic literature review of A* pathfinding". In: *Procedia Computer Science* 179 (2021), pp. 507–514.

[7] Christian Häcker et al. "Most Diverse Near-Shortest Paths". In: *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*. 2021, pp. 229–239.

[8] Tesshu Hanaka et al. "Computing diverse shortest paths efficiently: A theoretical and experimental study". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 4. 2022, pp. 3758–3766.

[9] M. Husár et al. "Reduction-based Solving of Multi-agent Pathfinding on Large Maps Using Graph Pruning". In: *Proceedings of the Twenty-first International Conference on Autonomous Agents and Multiagent Systems (AAMAS'22)*. Ed. by P. Faliszewski et al. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022, pp. 624–632. DOI: 10.5555/3535850.3535921.

[10] Antonın Komenda et al. "How to repair multi-agent plans: Experimental approach". In: *Proceedings of Distributed and Multi-agent Planning (DMAP) Workshop of 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*. Citeseer. 2013.

[11] Jiaoyang Li et al. "Anytime multi-agent path finding via large neighborhood search". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2021.

[12] Venkatraman Narayanan, Mike Phillips, and Maxim Likhachev. "Anytime safe interval path planning for dynamic environments". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 4708–4715.

[13] Bernhard Nebel. "On the Computational Complexity of Multi-Agent Pathfinding on Directed Graphs". In: (2019). DOI: 10.48550/ARXIV.1911.04871. URL: https://arxiv.org/abs/1911.04871.

[14] Mike Phillips and Maxim Likhachev. "Sipp: Safe interval path planning for dynamic environments". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 5628–5635.

[15] G. Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[16] R. Stern et al. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". In: *Proceedings of the Twelfth International Symposium on Combinatorial Search (SOCS'19)*. Ed. by P. Surynek and W. Yeoh. AAAI Press, 2019, pp. 151–159.

[17] R. Stern et al. "Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks". In: *CoRR* abs/1906.08291 (2019). URL: http://arxiv.org/abs/1906.08291.

[18] Roni Stern. "Multi-agent path finding–an overview". In: *Artificial Intelligence* (2019), pp. 96–115.

[19] J. Švancara et al. "Multi-agent Pathfinding on Large Maps Using Graph Pruning: This Way or That Way?" In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 15. 1. 2022, pp. 320–322.

Appendix

Figure 16: Heatmap global and local