2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Local Optimization of MAPF Solutions on Directed Graphs

S. Ardizzoni[1], I. Saccani[1], L. Consolini[1], M. Locatelli[1]
[1]Dipartimento di Ingegneria e Architettura, Università di Parma, Parco Area delle Scienze, 181/A, Parma, Italy

***Abstract*—Among sub-optimal Multi-Agent Path Finding (MAPF) solvers, rule-based algorithms are particularly appealing since they are complete. Even in crowded scenarios, they allow finding a feasible solution that brings each agent to its target, preventing deadlock situations. However, generally, rule-based algorithms provide solutions that are much longer than the optimal one. The main contribution of this paper is the introduction of an iterative local search procedure in MAPF. We start from a feasible suboptimal solution and we perform a local search in a neighborhood of this solution, to find a shorter one. Iteratively, we repeat this procedure until the solution cannot be shortened any longer. At the end, we obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one. We use dynamic programming for the local search procedure. Under this respect, the fact that our search is *local* is fundamental to reduce the time complexity of the algorithm. Indeed, if we apply a standard dynamic programming the number of explored states grows exponentially with the number of agents. As we will see, the introduction of a locality constraint allows solving the (local) dynamic programming problem in a time that grows only polynomially with respect to the number of agents.**

I. Introduction

We focus on the Multi-Agent Path Finding (MAPF) problem [1, 2]. We consider a directed graph and a set of agents. Each agent occupies a different node and can move to free nodes, i.e., nodes not occupied by other agents. The MAPF problem consists in computing a sequence of movements that repositions all agents to assigned target nodes, avoiding collisions. The main motivation comes from the management of fleets of automated guided vehicles (AGVs). AGVs move items between different locations in a warehouse. Each AGV follows predefined paths, that connect the locations in which items are stored or processed. We associate the layout of the paths to a directed graph. The nodes represent positions in which items are picked up and delivered, together with additional locations used for routing. The directed arcs represent the precomputed paths that connect these locations. If various AGVs move in a small scenario, each AGV represents an obstacle for the other ones. In some cases, the fleet can reach a deadlock situation, in which every vehicle is unable to reach its target. Hence, it is important to find a feasible solution to MAPF, even in crowded configurations.

Literature review. Various works address the problem of finding the optimal solution of MAPF (i.e., the solution with the minimum number of moves). For instance, CBS is a two-level algorithm which uses a search tree, based on conflicts between individual agents (see [3]). However, finding the optimal solution of MAPF is NP-hard (see [4]), and computational time grows exponentially with the number of agents. Therefore, optimal solvers are usually applied when the number of agents is relatively small. Conversely, sub-optimal solvers are usually employed when the number of agents is large. In such cases, the aim is to quickly find a path for the different agents, and it is often intractable to guarantee that a given solution is optimal. Among these, search-based solvers aim to provide a high quality solution, but are not complete (i.e., they are not always able to return a feasible solution). A prominent example is HCA* [5], in which agents are planned one at a time according to some predefined order. Instead, rule-based approaches include specific movement rules for different scenarios. They favor completeness at low computational cost over solution quality. One of the first important results in this field is from Kornhauser's thesis [6], which presents a rule-based procedure to solve MAPF (or to establish that MAPF has no feasible solution). Two relevant recent rule-based algorithms are TASS [7] and *Push and Rotate* [8] [9]. TASS is a tree-based agent swapping strategy which is complete on trees, while *Push and Rotate* solves every MAPF instance on graphs that contains at least two holes (i.e., unoccupied vertices). Reference [10] presents a method that converts the graph into a tree (as in [11]), and solves the resulting problem with TASS. Rule-based algorithms are also used for directed graph with at least two holes. In particular, reference [12] presents diSC algorithm, which solve any MAPF instance on strongly connected digraphs, i.e., directed graphs in which it is possible to reach any node starting from any other node. Another relevant reference is [13], which proposes diBOX, an algorithm that solves MAPF on the specific class of biconnected digraphs, i.e., strongly connected digraphs where the undirected graphs obtained by ignoring the edge orientations have no cutting vertices.

Motivations and statement of contribution. Among sub-optimal MAPF solvers, rule-based algorithms are particularly appealing since they are complete. Even in crowded scenarios, they allow finding a feasible solution that brings each agent to its target, preventing deadlock situations [12, 13]. However, generally, rule-based algorithms provide solutions that are much longer than the optimal one. This is a crucial limitation in industrial applications. The main contribution of this paper is the introduction of an iterative local search procedure in MAPF. We start from a feasible suboptimal solution, for instance the one provided by a rule-based algorithm. We perform a local search in a neighborhood of this solution, to find a shorter one. Iteratively, we repeat this procedure until the solution cannot be shortened any longer. At the end, we obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one. We use dynamic programming for the local search procedure. Under this respect, the fact that our search is *local* is fundamental to reduce the time complexity of the algorithm. Indeed, in principle, it is possible to solve the general MAPF problem by dynamic programming. However, the number of explored states grows exponentially with the number of agents, so that we cannot apply standard dynamic programming to problems involving many agents. As we will see, the introduction of a locality constraint allows solving the (local) dynamic programming problem in a time that grows only polynomially with respect to the number of agents (see Theorem IV.1).

979-8-3503-0123-6/23/$31.00 ©2023 IEEE

II. Problem Definition

A. MAPF problems

Let $G = (V, E)$ be a directed graph, with vertex set V and edge set E. We assign a unique label to each agent, and set P contains these labels. A *configuration* is a function $\mathcal{A} : P \to V$ that assigns the occupied vertex to each agent. A configuration is *valid* if it is injective (i.e., each vertex is occupied by at most one agent). Set $\mathcal{C} \subset \{P \to V\}$ represents all valid configurations.

Time is assumed to be discretized. At every time step, each agent occupies one vertex and executes a single action. There are two types of actions: *wait* and *move*. We denote the wait action by ι. An agent that executes this action remains in its current vertex for another time step. We denote a move action by $u \to v$. In this case, the agent moves from its current vertex u to an adjacent vertex v (i.e., $(u, v) \in E$). Therefore, the set of all possible actions for a single agent is $\bar{E} = E \cup \{\iota\}$.

Function $\rho : \mathcal{C} \times \bar{E} \to \mathcal{C}$ is a partially defined transition function such that $\mathcal{A}' = \rho(\mathcal{A}, u \to v)$ is the configuration obtained by moving an agent from u to v:

$$\mathcal{A}'(q) := \begin{cases} v, & \text{if } \mathcal{A}(q) = u; \\ \mathcal{A}(q), & \text{otherwise .} \end{cases} \tag{1}$$

Notation $\rho(\mathcal{A}, u \to v)!$ means that the function is well-defined. In other words $\rho(\mathcal{A}, u \to v)!$ if and only if $(u, v) \in E$ and $\mathcal{A}' \in \mathcal{C}$. Moreover, $(\forall \mathcal{A} \in \mathcal{C})\ \rho(\mathcal{A}, \iota)!$ and $\rho(\mathcal{A}, \iota) = \mathcal{A}$.

Since the movements of the agents can be synchronous, at each time step an action is an element of $\mathcal{E} = \bar{E}^{|P|}$, $a = (a_1, \ldots, a_{|P|})$ where a_i is the single move of agent i. We can extend function $\rho : \mathcal{C} \times \bar{E} \to \mathcal{C}$ to $\rho : \mathcal{C} \times \mathcal{E} \to \mathcal{C}$, by setting $\mathcal{A}' = \rho(\mathcal{A}, a)$ equal to the configuration obtained by moving agent i along edge a_i (or by not moving the agent if $a_i = \iota$). In this case, $(\forall a \in \mathcal{E}, \mathcal{A} \in \mathcal{C})\ \rho(\mathcal{A}, a)!$ if and only if the following conditions hold:

1) $\mathcal{A}' \in \mathcal{C}$: two or more agents cannot occupy the same vertex at the same time step;
2) $\forall i = 1, \ldots, |P|$, if $a_i = (u, v)$, then $\nexists j \in \{1, \ldots, |P|\}$ such that $a_j = (v, u)$: two agents cannot swap locations in a single time step.

We represent plans as ordered sequences of actions. It is convenient to view the elements of $\mathcal{E}$ as the symbols of a language. We denote by $\mathcal{E}^*$ the Kleene star of $\mathcal{E}$, that is the set of ordered sequences of elements of $\mathcal{E}$ with arbitrary length, together with the empty string ϵ: $\mathcal{E}^* = \bigcup_{i=1}^{\infty} \mathcal{E}^i \cup \{\epsilon\}$.

We extend function $\rho : \mathcal{C} \times \mathcal{E} \to \mathcal{C}$ to $\rho : \mathcal{C} \times \mathcal{E}^* \to \mathcal{C}$. $(\forall s \in \mathcal{E}^*, e \in \mathcal{E}, \mathcal{A} \in \mathcal{C})\ \rho(\mathcal{A}, se)!$ if and only if $\rho(\mathcal{A}, s)!$ and $\rho(\rho(\mathcal{A}, s), e)!$ and, if $\rho(\mathcal{A}, se)!$, then $\rho(\mathcal{A}, se) = \rho(\rho(\mathcal{A}, s), e)$.

Note that ϵ is the trivial plan that keeps all agents and holes at their positions.

We denote by $\mathcal{E}^*_{\mathcal{A}} = \{f \in \mathcal{E}^* : \rho(\mathcal{A}, f)!\}$ the set of plans such that $\rho(\mathcal{A}, f)$ is well defined. The problem of detecting a feasible solution is the following:

Problem 1: (**Feasibility MAPF problem**). Given a digraph $G = (V, E)$, an agent set P, an initial valid configuration $\mathcal{A}^s$, and a final valid configuration $\mathcal{A}^t$, find a plan f such that $\mathcal{A}^t = \rho(\mathcal{A}^s, f)$.

Now, for a feasible plan f, we define $|f|$ as the length of plan f, i.e., the number of time steps needed to let all agents reach the final configuration through plan f. Furthermore, given $k \in \mathbb{N}$, we denote by f_k the k-th prefix of f (that is, the prefix of f of length k, made up of the first k actions of f). Note that $|f_k| = k$.

We aim to solve a given MAPF instance while minimizing a global cumulative cost function. We employ the cost function called *Makespan*, equal to the time when the last agent reaches its destination (i.e., the maximum of the individual costs).

Problem 2: (**Optimization MAPF problem**). Given $\mathcal{A}^s$ and $\mathcal{A}^t$ initial and final valid configurations on a digraph G, the optimization MAPF problem with *Makespan* is defined as

$$\begin{aligned} &\min |f| \\ &\text{s.t. } \mathcal{A}^t = \rho(\mathcal{A}^s, f) \\ &f \in \mathcal{E}^*_{\mathcal{A}^s}. \end{aligned} \tag{2}$$

Let f_1 and f_2 be two solutions of the feasibility MAPF problem. We say that f_1 has better quality than f_2 (or, equivalently, f_2 is longer then f_1) if $|f_1| < |f_2|$. Other cost functions have also been used in the literature. *Sum-of-costs*, for example, is the summation, over all agents, of the number of time steps that an agent employs to reach its target without leaving it again. Unfortunately, finding the optimal solution, i.e., the minimal *Makespan* or *sum-of-costs*, has been shown to be NP-hard [4]. Therefore, in this paper we propose an approach to detect a good quality sub-optimal solution in polynomial time.

B. Distances

As said, we propose a solution approach based on the exploration of a *neighborhood* of a reference plan. To define a neighborhood, we introduce distances between vertices, configurations, and plans. Let $G = (V, E)$ be a digraph and P be a set of agents. We define the distance of vertex u from vertex v as the length of the shortest path on G from v to u: $d(u, v) = \ell(\pi_{vu})$, where π_{vu} is the shortest path in G from v to u and $\ell(\pi_{vu})$ is the length of that path, defined as the number of edges of π_{vu}. Note that d is not symmetrical, since π_{uv} and π_{vu} can be different. Next, we define the distance of configuration $\mathcal{A}^1$ from configuration $\mathcal{A}^2$ as the sum of the distances between the vertices that each agent occupies in the two configurations:

$$d : \mathcal{C} \times \mathcal{C} \to \mathbb{N} \qquad d(\mathcal{A}^1, \mathcal{A}^2) = \sum_{p \in P} d(\mathcal{A}^1(p), \mathcal{A}^2(p)).$$

Finally, we define the asymmetrical distance between two plans in $\mathcal{E}^*$. To do that, we associate to each plan a function in $\mathcal{L} = \{\psi : \mathbb{N} \to \mathcal{C}\}$ using the following

$$\Phi_{\mathcal{A}} : \mathcal{E}^*_{\mathcal{A}} \to \mathcal{L}, \qquad f \to \psi_f(k) := \begin{cases} \rho(\mathcal{A}, f_k), & k < |f|, \\ \rho(\mathcal{A}, f), & k \geq |f|, \end{cases}$$

where ψ_f is the function which associates to each $k \in \mathbb{N}$ the configuration at step k, that depends on the k-th prefix of f. We define the distance of plan f from plan g as the distance between the associated functions $\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)$:

$$d : \mathcal{E}^*_{\mathcal{A}} \times \mathcal{E}^*_{\mathcal{A}} \to \mathbb{N} \qquad d(f, g) := \bar{d}(\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)).$$

We can define $\bar{d}$ in various ways, leading to different definitions of the distance between plans f and g:

1) ∞-distance:

$$\bar{d}_{\infty}(\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)) := \max_{1 \leq k \leq \min\{|f|, |g|\}} d(\psi_f(k), \psi_g(k));$$

2) 1-distance:

$$\bar{d}_1(\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)) := \sum_{k=1}^{\min\{|f|,|g|\}} d(\psi_f(k), \psi_g(k));$$

3) max-min distance:

$$\bar{d}^*_\infty(\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)) := \max_{k\in\mathbb{N}} \min_{h\in\mathbb{N}} d(\psi_f(k), \psi_g(h));$$

4) sum-min distance:

$$\bar{d}^*_1(\Phi_{\mathcal{A}}(f), \Phi_{\mathcal{A}}(g)) := \sum_{k=1}^{\min\{|f|,|g|\}} \min_{h\in\mathbb{N}} d(\psi_f(k), \psi_g(h)).$$

Namely, with the ∞-distance, the distance of plans f and g corresponds to the maximum, with respect to time-step k, of the distance between the corresponding configurations at k. With the 1-distance, this distance corresponds to the sum, with respect to time-step k, of the distances between the corresponding configurations at k. With the max-min distance (respectively, the sum-min distance), this distance corresponds to the maximum (respectively, the sum) with respect to k, of the distance of the configuration that plan f reaches at step k with respect to the set of all configurations encountered by plan g. It is easy to see that, for each couple of plans f, g, the distance obtained from the 1-distance is the largest of the four, while the distance obtained from the max-min distance is the smallest. After having defined these distances, we can introduce an interesting variant of the optimization MAPF problem (2), namely, the *optimization MAPF problem constrained to a given plan*. This problem is faced when we have a sub-optimal solution f_0 of a MAPF instance, and we want to find another solution of the same problem which is not too far from f_0 and has better quality, i.e., shorter length. Given $\mathcal{A}^s$ and $\mathcal{A}^t$, initial and final valid configurations on a digraph G, given $f_0 \in \mathcal{E}^*_{\mathcal{A}^s}$ such that $\mathcal{A}^t = \rho(\mathcal{A}^s, f_0)$ (i.e., f_0 is a feasible solution of the MAPF instance), and given $r \in \mathbb{N}$ and a distance d between plans, the optimization MAPF problem with *Makespan* constrained to f_0 is defined as

$$\begin{aligned} &\min |f| \\ &\text{s.t. } \mathcal{A}^t = \rho(\mathcal{A}^s, f) \\ &\qquad f \in \mathcal{E}^*_{\mathcal{A}^s},\ d(f, f_0) \le r. \end{aligned} \tag{3}$$

C. Domain reduction of Problems (2) and (3)

In Problems (2) and (3), variable f belongs to the set of well-defined plans $\mathcal{E}^*_{\mathcal{A}^s}$. In order to reduce the cardinality of the feasible set of these two problems, we leverage some invariance properties. Namely, we define two equivalence relations on the set of plans $\mathcal{E}^*_{\mathcal{A}^s}$ such that the objective function of Problems (2) and (3) has the same value for all plans on the same equivalence class. Further, a plan is feasible if and only if all plans of the same equivalence class are feasible. In this way, we can convert Problems (2) and (3) into equivalent problems that have the set of equivalence classes as the optimization domain. Note that the set of equivalence classes corresponds to the states set that we will use in the dynamic programming solution algorithm. We will consider the following two equivalence relations on $\mathcal{E}^*_{\mathcal{A}}$.

Definition 2.1: Let $f_0 \in \mathcal{E}^*_{\mathcal{A}}$ be a reference plan. Given $f, g \in \mathcal{E}^*_{\mathcal{A}}$, then

1) $f \sim_1 g$ if and only if
   a) $|f| = |g|$;
   b) $\rho(\mathcal{A}, f) = \rho(\mathcal{A}, g)$.
2) $f \sim_2 g$ if and only if
   a) $f \sim_1 g$;
   b) $d(f, f_0) = d(g, f_0)$.

We denote by $\tilde{\mathcal{E}}^i_{\mathcal{A}}$ the set of all equivalence classes of $\sim_i$ on $\mathcal{E}^*_{\mathcal{A}}$. Let $\hat{f} \in \tilde{\mathcal{E}}^i_{\mathcal{A}}$ and $f \in \mathcal{E}^*_{\mathcal{A}}$ be a representative of the equivalence class $\hat{f}$. We define:

- the length of $\hat{f}$, $|\hat{f}| := |f|$;
- a new transition function,
$$\rho^* : \mathcal{C} \times \tilde{\mathcal{E}}^i_{\mathcal{A}} \to \mathcal{C}, \qquad \rho^*(\mathcal{A}, \hat{f}) := \rho(\mathcal{A}, f).$$
- the distance from $\hat{f}_0$ (the equivalence class to which f_0 belongs), $d(\hat{f}, f_0) := d(f, f_0)$ (only if $i = 2$).

Note that $|\hat{f}|$ is well-defined, since, by definition, all elements of equivalence class $\hat{f}$ have the same length. Similarly, ρ^* and d are well-defined since, for all elements f_1, f_2 of equivalence class $\hat{f}$, $\rho(\mathcal{A}, f_1) = \rho(\mathcal{A}, f_2)$ and, for $\sim_2$, $d(f_1, f_0) = d(f_2, f_0)$.

Let $\alpha_1 : \tilde{\mathcal{E}}^1_{\mathcal{A}} \to \mathbb{N} \times \mathcal{C}$ be such that

$$\alpha_1(\hat{f}) = (|\hat{f}|, \rho^*(\mathcal{A}, \hat{f})). \tag{4}$$

This function is well-defined because if $f_1 \sim_1 f_2$ then $|f_1| = |f_2|$ and $\rho(\mathcal{A}, f_1) = \rho(\mathcal{A}, f_2)$. Moreover, α_1 is injective because, if $\hat{f}_1$ and $\hat{f}_2$ are such that $\alpha_1(\hat{f}_1) = \alpha_1(\hat{f}_2)$, then $|\hat{f}_1| = |\hat{f}_2|$ and $\rho^*(\mathcal{A}, \hat{f}_1) = \rho^*(\mathcal{A}, \hat{f}_2)$, and, therefore, $\hat{f}_1 = \hat{f}_2$.

Let $\alpha_2 : \tilde{\mathcal{E}}^2_{\mathcal{A}} \to \mathbb{N} \times \mathcal{C} \times \mathbb{N}$ be defined as follows:

$$\alpha_2(\hat{f}) = (|\hat{f}|, \rho^*(\mathcal{A}, \hat{f}), d(\hat{f}, f_0)). \tag{5}$$

This function is well defined because if $f_1 \sim_2 f_2$, then $|f_1| = |f_2|$, $\rho(\mathcal{A}, f_1) = \rho(\mathcal{A}, f_2)$ and $d(f_1, f_0) = d(f_2, f_0)$. Moreover, α_2 is injective because, if $\hat{f}_1$ and $\hat{f}_2$ are such that $\alpha_1(\hat{f}_1) = \alpha_2(\hat{f}_2)$, then $|\hat{f}_1| = |\hat{f}_2|$, $\rho^*(\mathcal{A}, \hat{f}_1) = \rho^*(\mathcal{A}, \hat{f}_2)$, $d(\hat{f}_1, f_0) = d(\hat{f}_2, f_0)$, and, therefore, $\hat{f}_1 = \hat{f}_2$.

Since $f \sim_i g$, $i = 1, 2$, implies that $|f| = |g|$, $\rho(A^s, f) = \rho(A^s, g)$ and $d(f, f_0) \le r \leftrightarrow d(g, f_0) \le r$, it turns out that problems (2) and (3) are invariant under the equivalence relations $\sim_i$. Therefore, given $\hat{f}_0 \in \tilde{\mathcal{E}}^i_{\mathcal{A}^s}$, problem (3) (similar for problem (2)) can be defined as follows over the set of equivalence classes:

$$\begin{aligned} &\min |\hat{f}| \\ &\text{s.t. } \mathcal{A}^t = \rho^*(\mathcal{A}^s, \hat{f}) \\ &\qquad \hat{f} \in \tilde{\mathcal{E}}^i_{\mathcal{A}^s},\ d(\hat{f}, f_0) \le r. \end{aligned} \tag{6}$$

D. Neighborhoods

Given the distances defined in Section II-B and the definition of the equivalence classes in Section II-C, we can define the neighborhood of an equivalence class and estimate its cardinality. Such estimate is needed to evaluate the time needed to explore the neighborhoods, an operation that is central in the approach proposed in this paper.

Given a radius $r \in \mathbb{N}$, we define the following **neighborhood** of $f_0 \in \mathcal{E}^*_{\mathcal{A}}$:

$$\mathcal{N}_r(f_0) := \{\hat{g} \in \tilde{\mathcal{E}}^i_{\mathcal{A}} :\ |\hat{g}| \le |f_0|,\ d(\hat{g}, f_0) \le r\}.$$

Here we consider the distance based on the max-min distance. However, each of the distances defined in Section II-B can be

used to define the neighborhood. As we will see, different neighborhoods lead to different exploration policies of the MAPF solutions. The following proposition proved in [14] provides an upper bound on the cardinality of $\mathcal{N}_r(f_0)$.

Proposition II.1: The neighborhood of f_0 of radius r has a polynomial cardinality with respect to the number of nodes. In particular, $\exists C = C(r) \in \mathbb{R}$ such that

$$|\mathcal{N}_r(f_0)| \leq |f_0|^2 \left(1 + C(r+k)^r \phi^r\right),$$

where $k = |P|$ and $\phi = outdeg(G)$.
Note that, since the max-min distance is the smallest among the considered distances, the upper bound provided in the proposition is also valid for the distances based on the other distances previously discussed.

III. Iterative Neighborhood Search

In this section we describe the proposed iterative approach to detect a sub-optimal solution of Problem (2), or the equivalent counterpart of this problem defined over the set of equivalence classes. The returned solution is locally optimal with respect to the neighborhood of a reference solution. At each iteration, we solve an instance of Problem (6). More in detail, the algorithm takes as input a feasible solution f_0, that may be of poor quality. For instance, we can obtain f_0 from a rule-based algorithm, such as diSC [12] . We aim at improving f_0, obtaining a shorter solution. To this end, we solve Problem (6) with a dynamic programming algorithm. Namely, in neighborhood $\mathcal{N}_r(f_0)$ we search for plans shorter than f_0 through algorithm *DynProg*, that we will describe below. If we cannot obtain a solution shorter than f_0 (that is, f_0 is locally optimal) we stop the algorithm. Otherwise, if we obtain an improved solution f^*, we redefine the reference solution as $f_0 = f^*$ and solve again Problem (6). We iterate this procedure until we cannot shorten the current solution any further.

This algorithm can be classified as a **Neighborhood Search** algorithm (see [15]).

To define the neighborhood $N_r(f_0)$, we can use any distance function among those presented in Section II-B. In our numerical experiments, we used the sum-min distance. Algorithm 1 presents the steps of the procedure.

Algorithm 1 Neighborhood Search

```
Input: f_0, r, A^s, A^t
Output: f*
f* ← f_0
do
|  f_0 ← f*
|  f* ← DynProg(f_0, r, A^s, A^t)
while |f*| < |f_0|
return f*
```

In Algorithm 1, r is the local search radius, $\mathcal{A}^s$ is the initial configuration, and A^t is the final one.

IV. Dynamic Programming Algorithm

To search for the optimal solution of Problem (6), we employ a *Dynamic Programming* (DP) algorithm. In generic DP problems we are given a state space S where $A \subset S$ are the target states, an expansion function $g : S \to P(S)$, where P is the power set of S, and an objective function $c : S \to R$. Starting from an initial state $s_0 \in S$, we iteratively expand states with function g to explore the state space and compute $s_t = g^n(s_0) \in A$ with the minimal objective function. In our case, the states represent the equivalence classes of relation $\sim_2$, defined in Section II-C. Namely, we use injection $\alpha_2 : \tilde{\mathcal{E}}^2_{\mathcal{A}} \to \mathbb{N} \times \mathcal{C} \times \mathbb{N}$ to associate to each equivalence class $\hat{f}$ a triple $(\beta, \gamma, \sigma) = \alpha_2(\hat{f})$, where β is the length of $\hat{f}$, γ the configuration obtained by applying a plan representative of $\hat{f}$ to the initial state $\mathcal{A}^s$, and σ is the distance of a representative of $\hat{f}$ from reference plan f_0. Namely, the state space is

$$\mathcal{S} := \alpha_2(\tilde{\mathcal{E}}^2_{\mathcal{A}}) \subset \mathbb{N} \times \mathcal{C} \times \mathbb{N},$$

where α_2 is defined in (5). Since α_2 is injective, $\mathcal{S}$ and $\tilde{\mathcal{E}}^2_{\mathcal{A}}$ are in one-to-one correspondence. Each state $s = (\beta, \gamma, \sigma) \in \mathcal{S}$, represents the equivalence class:

$$\alpha_2^{-1}(s) = \{f \in \mathcal{E}_{\mathcal{A}} \,:\, \beta = |f|,\, \gamma = \rho(\mathcal{A}^s, f),\, \sigma = d(f, f_0)\}.$$

The initial state is $s_0 = \alpha_2(\epsilon) = (0, \mathcal{A}^s, 0)$. We use a priority queue Q to store the states that have not been visited yet. At the beginning, $Q = \{s_0\}$. We define a partial ordering on $\mathcal{S}$ based on length. Namely, if $s_1 = (\beta_1, \gamma_1, \sigma_1), s_2 = (\beta_2, \gamma_2, \sigma_2)$, $s_1 < s_2$ if $\beta_1 < \beta_2$. We order the elements of Q according to this partial ordering.

A state $s_1 = (\beta_1, \gamma_1, \sigma_1)$ *dominates* $s_2 = (\beta_2, \gamma_2, \sigma_2)$ if

- $\beta_1 \leq \beta_2$,
- $\gamma_1 = \gamma_2$,
- $\sigma_1 \leq \sigma_2$.

In other words, s_1 dominates s_2 if the plans f_1, f_2, corresponding to s_1 and s_2, satisfy the following properties. Plan f_1 is not longer than f_2, f_1 and f_2 lead to the same final configuration, and the distance of f_1 from the reference solution f_0 is not larger than the one of f_2. If s_1 dominates s_2, we can discard s_2. In general, we remove from Q all dominated states. We also define the following *transition function*, which allows to (possibly) add new states to the priority queue:

$$\tilde{\rho} : \mathcal{S} \times \mathcal{E} \to \mathcal{S}$$

$$\tilde{\rho}((\beta, \gamma, \sigma), e) := (\beta + 1, \rho(\gamma, e), \sigma + \min_{k \in \mathbb{N}} d(\rho(\gamma, e), \psi_{f_0}(k))).$$

Applying this function on a state $s = (\beta, \gamma, \sigma)$:

- adds 1 to the length of the class $\alpha_2^{-1}(s)$;
- updates the final configuration of the equivalence class, applying the function $\rho(\gamma, e)$ to the final configuration of $\alpha_2^{-1}(s)$ with e being the chosen set of edges;
- updates σ, adding the computed minimum distance between the updated final configuration $\rho(\gamma, e)$ and the reference plan f_0.

We define $\Sigma := \{\tilde{\rho}((\beta, \gamma, \sigma), e) \,:\, e \in \mathcal{E} \text{ and } \rho(\gamma, e) \in \mathcal{B}_r(\gamma)\} \subset \mathcal{S}$, the set of new states which can be generated through the transition function $\tilde{\rho}$ applied to the current state (β, γ, σ) and all possible actions in $\mathcal{E}$ leading to configurations in $\mathcal{B}_r(\gamma)$. Moreover, we denote with $\Gamma := \alpha_2(\mathcal{N}_r(f_0)) \subset \mathcal{S}$, the set of states that can be visited during a neighborhood search.

A. Algorithm

The Dynamic Programming algorithm is described in Algorithm 2. The priority queue Q maintained inside the algorithm is a set of states, ordered by the length β of their representatives. Function $\texttt{insert}(Q, x)$ inserts a state x maintaining the partial order of Q, in the sense that, after the insertion of x, all the elements of the queue still respect the partial ordering previously defined. Function $\texttt{remove}(Q, x)$ removes x from Q. The head of the queue, that is the state with minimal β, is denoted by $Q[0]$. The algorithm explores the state space

starting from the initial state s_0. At each iteration, the state with minimum β is extracted from the queue. If the state extracted is the target state (that is, if $\gamma = \mathcal{A}^t$) the algorithm stops and we return a representative of the optimal solution of Problem (6) (for a given equivalence class $\hat{f}$, the function $\texttt{repr}(\hat{f})$ returns a representative of the class). Otherwise, the algorithm employs function $\texttt{expand}(s, f_0, r)$, based on the transition function previously defined, to find new states. If a new state is not dominated, then it is added to the queue Q. Moreover, all states in Q dominated by the newly added state are removed from Q. Note that for more complicated solutions, the algorithm is far more effective.

Algorithm 2 Dynamic Programming with Dominance

```
Input: f_0, r, A^s, A^t
Output: f
s_0 ← (0, A^s, 0)
insert(Q, s_0)
while Q ≠ ∅ do:
    s = (β, γ, σ) ← Q[0]
    if γ = A_t then
        f ← repr(α_2^{-1}(s))
        Q ← ∅
    else
        Σ ← expand(s, f_0, r)
        for s_k ∈ Σ do
            if s_k is not dominated in Q then
                insert(Q, s_k)
                for s_i ∈ Q do
                    if s_k dominates s_i then
                        remove(Q, s_i)
return f
```

Theorem IV.1: Algorithm 1 and 2 have polynomial time complexity with respect to the number of nodes of the graph.
Proof. In Algorithm 2, the time complexity is $O(|Q|^2 \cdot |\Sigma|)$. Sets Q and Σ vary with each iteration, but always remain subsets of Γ. So at each iteration the following upper bound for their cardinality always holds: $|Q| \leq |\Gamma|$, $|\Sigma| \leq |\Gamma|$. Reminding that α_2 is injective and using the result of Proposition II.1, $|\Gamma| = |\mathcal{N}_r(f_0)| \leq |f_0|^2 \left(1 + C(1+k)^r \phi^r\right)$, where $k = |P|$ and $\phi = outdeg(G)$. Therefore, the time complexity of Algorithm 2 is $O(|f_0|^6 \left(1 + C(r+k)^r \phi^r\right)^3)$. Algorithm 1 recalls at most $|f_0|$ times Algorithm 2, and so it has time complexity $O(|f_0|^7 \left(1 + C(r+k)^r \phi^r\right)^3)$.

V. Experimental results

We performed two sets of experiments on different graphs and with initial solutions generated in two distinct ways. In both experiments, we used the Neighborhood Search of Algorithm 1, with the *Dynamic Programming* of Algorithm 2, to improve the given initial solutions. The algorithms have been coded with the *C++* programming language, and has been run on a *11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz* processor with a 16 GB RAM.

A. Random Graphs with sequentially generated initial solution

In the first set of experiments we generated random directed graphs with a number of nodes $|V|$ ranging from 20 to 100 by 10, and a number of edges $|E|$ equal to $4|V|$. The graphs are generated by creating $|E|$ random ordered pair of nodes and using them to build a directed graph. Only strongly connected graphs are selected. The number of agents $|P|$ ranges from 2 to 10, while $\mathcal{A}^s$ and $\mathcal{A}^t$ are randomly generated. To generate the initial solutions, each agent is brought to its target node one at a time, following the shortest path, in terms of crossed edges, from its source to its target in the graph obtained by removing the nodes currently occupied by all other agents (either their target or their source, depending on whether they have been already moved or not). Note that such procedure is not complete, i.e., it does not guarantee to find a feasible solution when it exists or to establish that no such feasible solution exists. We generated 100 random graphs, for which the described procedure was able to return a feasible solution, for every combination of number of nodes and number of agents. After some tuning, we set the radius r of the neighborhood equal to 5, which turned out to be a good compromise between the quality of the solutions found in the neighborhood and the time needed to explore the neighborhood (note that such time increases exponentially with r). Given the initial solution f_0 and the final one f^* returned by the proposed approach, the percentage decrease of the final solution w.r.t. the initial solution is equal to $100\frac{f_0 - f^*}{f_0}\%$. In Figures 1 and 2 we report the median of the average percentage decrease and of the running time (in seconds), respectively, for every combination of $|P|$ and $|V|$. It is worthwhile to remark that the percentage decrease tends to be lower as the number of agents increases. A tentative explanation is that for cases with a greater number of agents, when the sequential procedure to generate an initial solution is able to return a feasible solution, such solution is already a good one which cannot be largely improved. This phenomenon is not observed in the second set of experiments, where a different procedure to generate an initial feasible solution is employed. Note that the average percentage decrease is lower when the number of agents becomes higher. This can be explained by the nature of the algorithm employed in finding the initial solution. Increasing the number of agents without increasing the number of nodes gives us way less feasible instances and the solutions found are more difficult to improve.

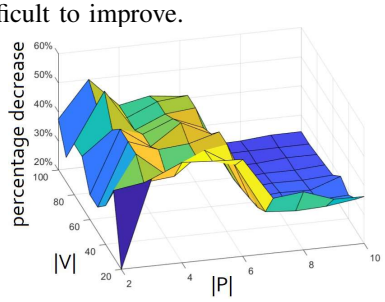

Figure 1: Average Percentage Decrease per $|P|$ and $|V|$.

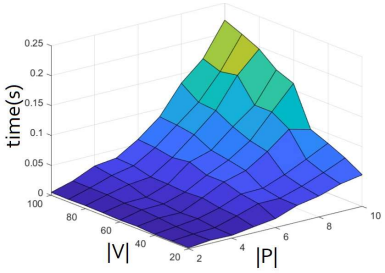

Figure 2: Average Running Time per $|P|$ and $|V|$.

B. Strongly connected with multiple biconnected components graphs and rule-based generated initial solution

In the second set of experiments, we generated strongly connected graphs with multiple biconnected components. The procedure used to generate such graphs can be found in

[12]. The number of agents $|P|$ ranges from 2 to 6, and the number of nodes vary from 20 to 50 with an increment of 10. The initial and final configuration $\mathcal{A}^s$ and $\mathcal{A}^t$ are randomly generated. In this case the initial solutions (if they exist) are generated through the diSC algorithm [12] which is complete, i.e., it always returns a feasible solution in case one exists. Such initial solutions usually have lower quality (i.e., the initial plans are usually longer) with respect to the ones used in the first set of experiments. This might be also the explanation why the percentage decrease in these experiments (see Figure 3) appears to be larger w.r.t the first set of experiments, and also tends to increase with the number of agents (differently from the case of random graphs). Again after some tuning, we set the radius r of the neighborhood equal to 3. Figures 3 and 4 report the median of the average percentage decrease and of the running time (in seconds), respectively, for every combination of $|P|$ and $|V|$. Note that the graphs employed in the second set of experiments appear to be more challenging with respect to the random ones. Indeed, computing times are larger and increase rapidly with the number of agents.

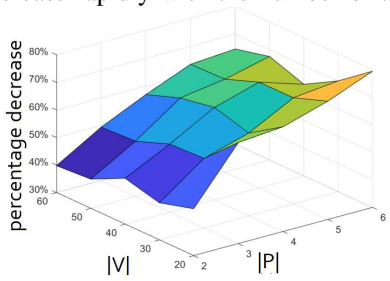

Figure 3: Average Percentage Decrease per $|P|$ and $|V|$.

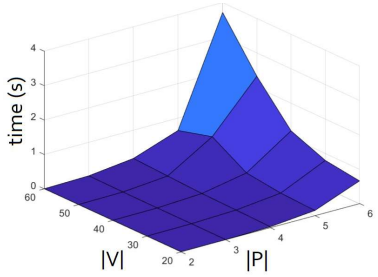

Figure 4: Average Running Time per $|P|$ and $|V|$.

VI. Conclusion and Future Works

We proposed an iterative local search procedure for MAPF, in order to shorten a known feasible solution. We obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one. The proposed algorithm has polynomial time complexity with respect to the number of agents (see Theorem IV.1) and has computational times compatible with industrial applications. We can extend the results in various respects, that will be the focus of future research:

- We can define locality constraints different from the ones considered in Section II-B. For instance, we can set a maximum on the number of agents that modify their path with respect to the reference solution. Alternatively, we can set an upper bound on the number of time intervals in which the solution departs from the reference one.
- We can test the proposed approach on real industrial scenarios.
- We can improve the complexity bound presented in Theorem IV.1, currently based on a quite rough bound.

References

[1] Roni Stern et al. "Multi-agent pathfinding: Definitions, variants, and benchmarks". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 10. 1. 2019, pp. 151–158.

[2] Bernhard Nebel. "On the computational complexity of multi-agent pathfinding on directed graphs". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 212–216.

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

[4] Jingjin Yu and Steven LaValle. "Structure and intractability of optimal multi-robot path planning on graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 1. 2013, pp. 1443–1449.

[5] David Silver. "Cooperative Pathfinding". In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 1.1 (Sept. 2021), pp. 117–122.

[6] D. Kornhauser, G. Miller, and P. Spirakis. "Coordinating Pebble Motion On Graphs, The Diameter Of Permutation Groups, And Applications". In: *25th Annual Symposium onFoundations of Computer Science, 1984.* Singer Island, FL: IEEE, 1984, pp. 241–250.

[7] Mokhtar Khorshid, Robert Holte, and Nathan Sturtevant. "A Polynomial-Time Algorithm for Non-Optimal Multi-Agent Pathfinding". In: *Proceedings of the International Symposium on Combinatorial Search* 2.1 (Aug. 2021), pp. 76–83.

[8] B. de Wilde, A. W. Ter Mors, and C. Witteveen. "Push and Rotate: a Complete Multi-agent Pathfinding Algorithm". In: *Journal of Artificial Intelligence Research* 51 (Oct. 2014), pp. 443–492.

[9] Ebtehal Turki Saho Alotaibi and Hisham Al-Rawi. "Push and spin: A complete multi-robot path planning algorithm". In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. Phuket, Thailand: IEEE, Nov. 2016, pp. 1–8.

[10] Athanasios Krontiris, Ryan Luna, and Kostas Bekris. "From Feasibility Tests to Path Planners for Multi-Agent Pathfinding". In: *Proceedings of the International Symposium on Combinatorial Search* 4.1 (Aug. 2021), pp. 114–122.

[11] Gilad Goraly and Refael Hassin. "Multi-Color Pebble Motion on Graphs". en. In: *Algorithmica* 58.3 (Nov. 2010), pp. 610–636.

[12] S. Ardizzoni et al. "Multi-Agent Path Finding on Strongly Connected Digraphs". In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. Cancun, Mexico: IEEE, Dec. 2022, pp. 7194–7199.

[13] Adi Botea, Davide Bonusi, and Pavel Surynek. "Solving Multi-agent Path Finding on Strongly Biconnected Digraphs". In: *Journal of Artificial Intelligence Research* 62 (June 2018), pp. 273–314.

[14] S Ardizzoni et al. "Local Optimization of MAPF solutions on Directed Graphs". In: *arXiv preprint arXiv:2304.01765* (2023).

[15] David Pisinger and Stefan Ropke. "Large neighborhood search". In: *Handbook of metaheuristics* (2019), pp. 99–127.