Date of publication xxxx 00, 0000, date of current version January 08, 2020.
Digital Object Identifier 10.1109/ACCESS.2017.DOI

A Discrete Event Formulation for Multi-Robot Collision Avoidance on Pre-planned Trajectories

DIEGO DEPLANO [1], MAURO FRANCESCHELLI [1], (Member, IEEE), SIMON WARE, SU RONG [2], (Senior, IEEE), AND ALESSANDRO GIUA [1], (Fellow, IEEE)
[1]Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy (e-mail: {diego.deplano, mauro.franceschelli, giua}@unica.it)
[2]School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798 (e-mail: {sware, rsu}@ntu.edu.sg)
Corresponding author: M. Franceschelli (e-mail: mauro.franceschelli@unica.it).

This work was supported in part by Region Sardinia (RAS) with project MOSIMA, RASSR05871, FSC 2014-2020, Annualita' 2017, Area Tematica 3, Linea d'Azione 3.1.

ABSTRACT In this paper we consider the problem of collision avoidance among robots that follow pre-planned trajectories in a structured environment while minimizing the maximum traveling time among them. More precisely, we consider a discrete event formulation of this problem. Robots are modeled by automata, the environment is partitioned into a square grid where cells represent free space, obstacles and walls, which are modeled as shared resources among robots. The main contribution of this paper is twofold. First, we propose a problem formulation based on mixed integer linear programming to compute an optimal schedule for the pre-planned trajectories. Second, we propose a heuristic method to compute a sub-optimal schedule: the computational complexity of this approach is shown to be polynomial with the number of robots and the dimension of the environment. Finally, simulations are provided to validate performance and scalability of the proposed approach.

INDEX TERMS Discrete Event Systems; Heuristic Solution; MILP Problem; Multi-Robot Path-Planning; Optimal Solution; Optimization; Scheduling.

I. INTRODUCTION

MULTI-ROBOT Path Planning (MRPP) is a fundamental problem in robotics, whose objective is to move all agents to their respective goal through the same environment while taking into account safety constraints, such as collisions avoidance [1]–[4], and performance constraints, related to the travelling time [5]–[7]. The focus of this work is on centralized MRPP problems where the environment is discretized and the objective is to minimize the maximum traveling time among the robots. As this problem is PSPACE-complete [8] it is generally solved by heuristic approaches giving not optimal solutions. A solution assigns a trajectory to each robot, i.e., a set of movements and a time schedule. Thus, the non optimality of a solution may depend on the robots' movements and on the associated time schedule.

We address the problem of improving a given solution by computing new time schedules for the trajectories, which reduce the maximum traveling time among the robots while not changing the robots' movements and avoiding collisions. Given a solution computed from an external path planner, we provide two approaches to compute an improved time schedule. The first approach is optimal with an high computational complexity; the second approach is heuristic with a polynomial complexity. A preliminary version of such a heuristic method was introduced in [9], while here the method is formalized and a proof is provided. To evaluate the proposed optimal and heuristic approaches, we use as reference path planner the algorithm we proposed in [10].

The **main contributions** of this paper are: (i) a Mixed Integer Linear Programming (MILP) problem formulation and (ii) a heuristic approach to minimize the maximum traveling time among robots; (iii) a characterization of the complexity of the proposed heuristic; (iv) numerical results which show the effectiveness of the proposed approaches.

After a brief review of related works in Section II, we present in Section III all relevant and necessary preliminaries to our discrete event formulation of the MRPP problem [9]. In Section IV the main problem under consideration is stated. In Sections V-VI we propose, respectively, a MILP formulation and a heuristic algorithm, whose complexity is discussed in Section VII. Finally, after the presentation of numerical results in Section VIII, we give concluding remarks in Section IX.

II. RELATED WORKS

In the current literature, a wide variety of MRPP problem formulations are proposed under different working assumptions and by means of different mathematical tools. For background and theory on motion planning we refer the reader to [11] and to [12], [13] for comprehensive reviews.

The focus of this paper is on centralized approaches, which can be divided into three classes: (i) translating the MRPP problem to other problems that are well studied in computer science; (ii) heuristic solvers; (iii) optimal solvers. In the following, we include a brief review of heuristic solvers belonging to the *search-based* and *rule-based* families of solvers having similar formulations as ours, to which the approaches proposed in this work can be possibly applied.

A notable example of search-based solvers is the Hieratchical Cooperative A* (HCA*) [14]. Such solvers reduce the problem complexity by computing, for each robot, independent paths [15] [16] which are then coordinated to avoid all possible collisions. However, they cannot guarantee finding a solution in all cases [17], except under certain conditions [18], and cannot guarantee an optimal solution.

Graph-based MRPP problem formulations and corresponding rule-based MRPP solver [6], [19]–[22] can be traced back to [23]. As highlighted in [24] and more recently in [25], such solvers are guaranteed to return a feasible solution if there is one and have polynomial time complexity. In this formulation, the robots are confined to an arbitrary connected graph, where nodes model the partitioning of the environment and edges the allowed movements, and collisions arise if two robots move to the same vertex or along the same edge. In two of the most recent and relevant works [6], [25], it is shown that the problem of minimizing the makespan is NP-hard and several heuristics are introduced.

The main difference between these works and the one presented here is that they allow *cyclic rotations* of robots along fully occupied cycles since a robot can enter a position while another one is leaving it.

III. PRELIMINARIES

Event sequences and languages are a simple mean to describe the behaviour of a discrete event system. Their basic building blocks are *events* σ, which belong to an *alphabet* Σ.

A *string* is a sequence of events $s = \sigma_1 \cdots \sigma_m$. The set of all strings of events in Σ is denoted Σ^*: this set also includes the *empty string* ε. A *language* is a set of strings $L \subseteq \Sigma^*$. The *concatenation* of two strings $s, t \in \Sigma^*$ is written as st.

A *substring* s' of s is a string contained in s. Further, $s[p:q]$ denotes the substring of $s = \sigma_1 \cdots \sigma_m$ having $\sigma_p \in s$ as the first event and $\sigma_q \in s$ as the last event. A *superstring* s of s' is a string which contains s'. We denote the set of substrings of s as $sub(s)$ and the set of superstrings of s as $sup(s)$. *Prefixes* and *suffixes* are special cases of substring.

A prefix p of a string s is a substring of s that occurs at the beginning of s, while a suffix q of a string s is a substring that occurs at the end of s. We denote the set of prefixes of s as $pre(s)$ and the set of superstrings of s as $suf(s)$. It holds that $pre(s) \subset sub(s)$ and $suf(s) \subset sub(s)$.

System behaviors are modelled using *automata*. A *finite-state automaton* is a 5-tuple $G = (X, \Sigma, \delta, x_0, X_m)$ where X is a finite set of states, Σ is a *finite alphabet* of events, $\delta : X \times \Sigma \to X$ is the (partial) transition function, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of marked (or final) states. The transition function is extended to strings in Σ^* by letting, for all $x \in X$, $\delta(x, \varepsilon) = x$ and, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, $\delta(x, s\sigma) = \delta(\delta(x, s), \sigma)$. Furthermore, given a string $s \in \Sigma^*$, $\delta(x, s)!$ means that there exists a state y such that $\delta(x, s) = y$, i.e., $\delta(x, s)$ is defined.

The *language* generated by an automaton G, or the closed behaviour of G, is $L(G) = \{s \in \Sigma^* | \delta(x_o, s)!\}$. The language accepted by an automaton G, or the marked behaviour of G, is is $L_m(G) = \{s \in \Sigma^* | \delta(x_o, s) \in X_m\}$.

IV. DISCRETE EVENT PROBLEM FORMULATION

We consider a set of n mobile robots positioned in a planar environment. The environment where the robots operate is partitioned into a square grid, each square cell is denoted as a *position*. As shown in Figure 1, *accessible* positions are denoted by white squares while *inaccessible* positions are denoted by black squares. Each accessible position is labeled with p_i with $i = 1, \ldots, N_p$, where N_p denotes the total number of accessible positions in the environment. A *wall* between two adjacent positions is represented by a thick line. Two adjacent positions are *connected* if there is no wall between them and are *disconnected* otherwise. The movement of a robot between two accessible and connected position is said to be a *feasible* transition. A robot can move only between connected and accessible positions.

Each robot is an automaton $G_g = (X_g, \Sigma_g, \delta_g, x_{0,g}, X_{m,g})$, defined as follows.

- X_g is the set of states: a state $x_{g,i}$ belongs to this set if position p_i can be visited by robot r_g.
- Σ_g is the alphabet: an event $\sigma = (g, p_i, p_j)$ belongs to this set if there exists an available transition of robot g from position p_i to position p_j.
- $\delta_g : X_g \times \Sigma_g \to X_g$ is the transition function: for all available transitions $\sigma = (g, p_i, p_j)$ we define $x_{g,j} = \delta(x_{g,i}, \sigma)$.
- $x_{g,0}$ is the initial state of robot g corresponding to a generic initial position.
- $X_{g,m}$ is the set of final states of robot g corresponding to generic final positions.

The overall system can be completely described by the 3-tuple $(G, f, \mathcal{R})$, which we call System of Time-weighted Automata with Resources (STAR). First we introduce the concept of resource.

Definition 1 (Standard Set of Resources) Consider a set of n robots described by automata $G_g =$

$(X_g, \Sigma_g, \delta_g, x_{0,g}, X_{m,g})$ for $g \in \{1, \dots, n\}$ and let $\Sigma = \bigcup_{g=1}^{n} \Sigma_g$ be set of all events across all robots.

A *resource* is a set of events $r \subseteq \Sigma_g$ which can not occur simultaneously.

The *standard set of resources* $\mathcal{R}$ of $G = G_1 \times \cdots \times G_n$ contains:

- A resource R_g for each robot r_g with $g = 1, \dots, n$ such that all events of Σ_g are in that resource;
- A resource P_i for each position p_i with $i \in 1, \dots, N_p$ such that all events (across all robots) which involve the position p_i are in that resource. ■

Definition 2 (System of Time-weighted Automata with Resources) A *System of Time-weighted Automata with Resources (STAR)* is a 3-tuple $\mathcal{G} = (G, f, \mathcal{R})$, in which

- $G = G_1 \times G_2 \times \ldots \times G_n = (X, \Sigma, \delta, x_0, X_m)$ denotes the finite-state automaton obtained by parallel composition of n automata, each one describing a single robot;
- $f : \Sigma \mapsto \mathbb{R}^+$ is a $weight$ function which assigns to each event $\sigma \in \Sigma$ a positive real number $f(\sigma)$ which denotes the time required to execute that event;
- $\mathcal{R}$ denotes the standard set of resources, as in Definition 1. ■

We now give an example of two robots in an environment, their automata and their $STAR$.

Example 1 In Figure 1 it is shown an environment where the initial and final positions of robots r_1 and r_2 are denoted, respectively, as I1, I2 and F1, F2. On the sides, the automata G_1 and G_2 corresponding to robots 1 and 2 are depicted. We now show how to construct the corresponding STAR $\mathcal{G} = (G, f, \mathcal{R})$:

- $G = (X, \Sigma, \delta, x_0, x_m) = G_1 \times G_2$ is the parallel composition of the automata where the single final state $x_m \in X_m$ is the cartesian product to the single final states of automata G_1 and G_2.
- f is a function which assigns a weight to all events $\sigma \in \Sigma$ corresponding to the time required by the robot to execute the movement. In this example robots always require exactly one time unit to move between two squares, thus it holds $f(\sigma) = 1, \forall \sigma \in \Sigma$.
- $\mathcal{R}$ is a set of resources defined as follows. Each robot r_g is modeled as a resource R_g such that for each feasible transition of the robot, its corresponding event is in that resource. Each position p_j is also modeled as a resource P_j such that for each feasible transition (across all robots) which involves position p_j, its corresponding event is in that resource. In this example, the set of resources is $\mathcal{R} = \{R_1, R_2, P_1, P_2, P_3, P_4, P_5\}$ where

$$
\begin{aligned}
R_1 = \ & \{(1,p_3,p_1),(1,p_3,p_2),(1,p_3,p_4),(1,p_3,p_5),\\
& (1,p_1,p_3),(1,p_2,p_3),(1,p_4,p_3),(1,p_5,p_3)\};\\
R_2 = \ & \{(2,p_3,p_1),(2,p_3,p_2),(2,p_3,p_4),(2,p_3,p_5),\\
& (2,p_1,p_3),(2,p_2,p_3),(2,p_4,p_3),(2,p_5,p_3)\};\\
P_1 = \ & \{(1,p_1,p_3),(1,p_3,p_1),(2,p_1,p_3),(2,p_3,p_1)\};\\
P_2 = \ & \{(1,p_2,p_3),(1,p_3,p_2),(2,p_2,p_3),(2,p_3,p_2)\};\\
P_3 = \ & \{(1,p_3,p_1),(1,p_3,p_2),(1,p_3,p_4),(1,p_3,p_5),\\
& (1,p_1,p_3),(1,p_2,p_3),(1,p_4,p_3),(1,p_5,p_3),\\
& (2,p_3,p_1),(2,p_3,p_2),(2,p_3,p_4),(2,p_3,p_5),\\
& (2,p_1,p_3),(2,p_2,p_3),(2,p_4,p_3),(2,p_5,p_3)\};\\
P_4 = \ & \{(1,p_4,p_3),(1,p_3,p_4),(2,p_4,p_3),(2,p_3,p_4)\};\\
P_5 = \ & \{(1,p_5,p_3),(1,p_3,p_5),(2,p_5,p_3),(2,p_3,p_5)\}.
\end{aligned}
$$

■

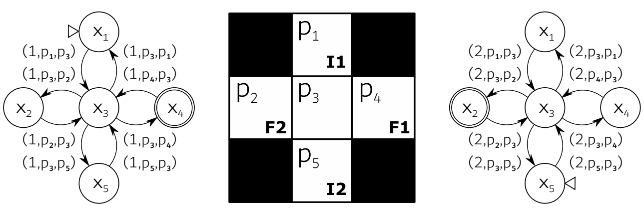


Figure 1: Translation of MRPP problem in a DES context: on the left a sample of environment and on the right an automaton representing a robot with start and goal position.

Given a STAR $\mathcal{G} = (G, f, \mathcal{R})$, a string $s \in L_m(G)$ denotes a sequence of movements which leads the robots from their initial position to their final position.

Definition 3 (Projection) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR and $s = \sigma_1, \dots, \sigma_m \in L_m(G)$ be a string.

The projection $\pi_r(s)$ of string s over $r \in \mathcal{R}$ is a new string where events not belonging to resource r are removed. ■

Thus, $\pi_{R_g}(s)$ denotes the sequence of movements of robot g. Events in a string are executed depending on their order. The main problem addressed by this paper is to determine a *schedule* for the events in s (allowing events to be executed at the same time) such that robots do not collide.

Definition 4 (Schedule) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR and $s = \sigma_1, \dots, \sigma_m \in L_m(G)$ be a string.

A *schedule* of s for $\mathcal{G}$ is an ascending ordered list of nonnegative real numbers $\rho = [t_1, \dots, t_m] \subset \mathbb{R}^m_{\geq 0}$, with t_k establishing the start of the execution of the event σ_k. We denote $\mathcal{P}(s)$ the set of schedules of s.

A schedule is a *sequential schedule* if for all $k = 1, \dots, m$ it holds that $t_1 = 0$ and $t_{k+1} = t_k + f(\sigma_k)$. ■

Thus, when a schedule $\rho = [t_1, \dots, t_m]$ is associated to a string $s = \sigma_1 \cdots \sigma_m$, each event $\sigma_k \in s$, for $k = 1, \dots, m$, has a *starting time* t_k, an *event execution time* $f(\sigma_k)$ and a *completion time* $t_k + f(\sigma_k)$. The events are executed following the order in the string s but their execution can be overlapped in time. The sequential schedule represents the classic way to execute events in a discrete event system, i.e., no two events are executed simultaneously. In this context, the sequential schedule implies that each robot moves while the others are standing. In this work we deal with strings for which the sequential schedule does not give a collision, i.e., no two robots occupy the same poisition a the same time. We call them *valid strings*, as follow.

Definition 5 (Valid String) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR with $G = G_1 \times \cdots G_n$ being the concurrent composition of n robots and $x_0 = (x_{1,0}, \cdots, x_{n,0})$ be the initial states.

The string $s \in L(G)$ is a *valid string* if for all prefixes $\hat{s}$ of s, and for all couples $g, h \in \{1, \dots, n\}$, it holds that

$$\delta_g(x_{g,0}, \pi_{R_g}(\hat{s})) \neq \delta_h(x_{h,0}, \pi_{R_h}(\hat{s})). \quad (1)$$

i.e., no two robots occupy the same position after the firing of any prefix of s. The set of all valid strings of G is denoted $\mathcal{V}(G) \subset L(G)$. ■

Given a valid string, for all other schedules which are not the sequential one, one wants to characterize the ones which do not give rise to collision. We call them *collision-free schedules*.

Definition 6 (Collision-Free Schedule) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR, $s \in \mathcal{V}(\mathcal{G})$ a valid string, $\rho \in \mathcal{P}(s)$ a schedule of s.

The schedule ρ is a *collision-free schedule* if and only if for all $q, v \in \{1, \dots, n\}$, $q < v$ and $\sigma_q, \sigma_v \in r \in \mathcal{R}$ it holds that $t_q + f(\sigma_q) \leq t_v$. The set of collision-free schedules of s is denoted $\mathcal{P}_{cf}(s)$. ■

For any valid string there exist infinitely many collision-free schedules: first, there exists at least one, which is the sequential one; second, delaying the firing time of all events in the sequential schedule by the same quantity results in a new valid schedule. When a collision-free schedule is associated with a string, we can define the makespan of a string, which is the time needed to execute all events in the string with respect to the given schedule.

Definition 7 (Makespan) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR, $s = \sigma_1, \dots, \sigma_m \in \mathcal{V}(G)$ a valid string, $\rho = [t_1, \dots, t_m] \in \mathcal{P}_{cf}(s)$ a collision-free schedule of s.

The *makespan* of a string s given a schedule ρ is the largest completion time among all events in the string

$$\tau_\rho(s) = \max_{k=1,\dots,m} t_k + f(\sigma_k).$$

If $s = \varepsilon$, i.e., the empty string, we let $\tau_\rho(\varepsilon) = 0$. ■

Finally, it is of interest defining the best collision-free schedule with respect to the resulting makespan of a given string. The smallest makespan among all possible collision-free schedules is called *strict makespan*.

Definition 8 (Strict Makespan) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a time-weighted automaton, let $s \in L_m(G)$ be a string.

The *strict makespan* of the string s is

$$\upsilon(s) = \min_{\rho \in \mathcal{P}_{cf}(s)} \tau_\rho(s).$$

If $s = \varepsilon$, i.e., the empty string, we let $\upsilon(\varepsilon) = 0$. ■

Given robots' trajectories $s_g \in \mathcal{V}(G_g)$, there can exists several strings $s \in \mathcal{V}(G)$ that give trajectories s_g, i.e., $\pi_{R_g}(s) = s_g$ for all $g \in \{1, \dots, n\}$. We thus define the trajectory invariant set of a string.

Definition 9 (Trajectory Invariant Set) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR and $s \in \mathcal{V}(\mathcal{G})$ a valid string.

The *trajectory invariant set* of the string s is defined as $\mathcal{S}(s) = \{s' \in \Sigma^* | \pi_{R_g}(s') = \pi_{R_g}(s), \ \forall g = 1, \dots, n\}$. ■

Given a STAR $\mathcal{G} = (G, f, \mathcal{R})$, we point out that two valid strings $s_1, s_2 \in \mathcal{V}(G)$ such that $s_2 \in \mathcal{S}(s_1)$, can have different strict makespans since the events in a string can be fired only following their order in the string (see Example 2). Because of this fact, given a valid string s_1, we address the problem of computing a new valid string s_2 obtained by shuffling events of s_1 while preserving robots' trajectories, i.e., $s_2 \in \mathcal{S}(s_1)$, such that s_2 has the lowest strict makespan among all strings in $\mathcal{S}(s_1)$. In other words, the trajectory of each robot g is fixed as $\pi_{R_g}(s_1)$, but the order of events of different robots are adjustable via shuffling, aiming for a minimum strict makespan while avoiding collisions.

Problem 1 (Optimal String) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR and let $s = \sigma_1 \cdots \sigma_m \in \mathcal{V}(G) \subset L_m(G)$ be a valid string.

Compute among all valid strings in $\mathcal{S}(s)$, the one whose strict makespan is minimum, i.e.,

$$s^* = \arg\min_{s' \in \mathcal{V}(G) \cap \mathcal{S}(s)} \upsilon(s').$$

V. OPTIMAL SOLUTION

To solve Problem 1, given a valid string s, one can directly compute the earliest time each event in s can be fired and then construct the new valid string based on this schedule. This consideration translates the problem into Problem 2.

Problem 2 (Optimal Schedule) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR, let $s = \sigma_1 \cdots \sigma_m \in \mathcal{V}(G) \subset L_m(G)$ be a valid string and let $s_g = \pi_{R_g}(s) = \sigma_g^1 \sigma_g^2 \cdots \sigma_g^{m_g}$, for all $g = 1, \dots, n$.

Compute schedules $\rho_g = [t_g^1, t_g^2 \cdots t_g^{m_g}]$ for all robots $g = 1, \dots, n$ such that, defined

(i) A schedule $\bar{\rho}$ by sorting elements of ρ_g for $g = 1, \dots, n$ in ascending order;
(ii) A string $\bar{s}$ by sorting elements of s_g for $g = 1, \dots, n$ with respect to $\bar{\rho}$,

it holds that

1) The string $\bar{s}$ is a valid string, i.e., $\bar{s} \in \mathcal{V}(G)$;
2) The makespan $\tau_{\bar{\rho}}(\bar{s})$ of $\bar{s}$ given $\bar{\rho}$ is equal to the lowest stric makespan among all the valid shufflings of s, i.e.,

$$\tau_{\bar{\rho}}(\bar{s}) \equiv \min_{s' \in \mathcal{V}(G) \cap \mathcal{S}(s)} \upsilon(s').$$

■

Proposition 1 Solutions to Problem 1 and Problem 2 are equivalent.

Proof. In Problem 1 the solution is a string s^* while in Problem 2 the solution is a set of schedules ρ_g for $g = 1, \dots, n$. They are equivalent in the sense that the string $\bar{s}$, obtained as explained in Problem 2, has the same optimal makespan as the string s^*, i.e., $\upsilon(\bar{s}) \equiv \upsilon(s^*)$. □

An optimal solution to Problem 2 (and consequently to Problem 1 by Proposition 1) can be computed by the MILP Model in Proposition 2. Note that the constraints considered in equation (2) are defined in following, within the proof of the proposition..

Proposition 2 (MILP Model) Let $(G, f, \mathcal{R})$ be a STAR, $s = \sigma_1 \cdots \sigma_m \in \mathcal{V}(G)$ be a valid string and let us denote the projection of s on robot g as $s_g = \pi_{R_g}(s) = \sigma_g^1 \sigma_g^2 \cdots \sigma_g^{m_g}$, for all $g = 1, \ldots, n$. Consistently, for each event σ_g^k in the string s we define a variable t_g^k which refers to its firing time. The following MILP

$$\begin{aligned} \text{minimize} \quad & \max_{\sigma_g^k \in s}\{t_g^k + f(\sigma_g^k)\} \\ \text{subject to} \quad & \textit{Positivity constraints } (3) \\ & \textit{Robot constraints } (4) \\ & \textit{Position constraints } (5.1, 5.2, 5.3) \end{aligned} \tag{2}$$

allows to compute schedules $\rho_g = [t_g^1, t_g^2 \cdots t_g^{m_g}]$ which are solution to Problem 2. ■

Proof. We will go through a detailed explanation of all the constraints and the objective function, with the aim of showing that constraints set (2) defines the set of collision free schedules for the considered problem and that the objective function is optimized by the optimal schedule.

Objective function: The objective function of (2) is the makespan of the string infered from the firings time, consistently to contidion 2) of Problem 2.

Positivity constraints: We consider the first moment a robot starts moving as $t = 0$, then all starting times are in $\mathbb{R}_{\geq 0}$. Thus, we have the following set of constraints $\forall g \in [1, n],\ k \in [1, m_g]$

$$t_g^k \geq 0\,. \tag{3}$$

Robot constraints: Events belonging to the same robot must be executed subsequently with respect to their order in s. Thus, we have the following set of constraints $\forall g \in [1, n],\ k \in [1, m_g - 1]$

$$t_g^k + f(\sigma_g^k) \leq t_g^{k+1}\,, \tag{4}$$

i.e., event σ_g^{k+1} has to start after the completion of event σ_g^k.

Position constraints: Each position occupied by a robot can be the *initial*, *final* or *intermediate*. For each category, we define a set of constraints.

- Initial: Each robot has to leave its initial position before all other robots enter it. This means that the first event of each robot must be executed (and terminated) before all events of other robots which involve that position. Thus, call P_i^g the resource of initial position of robot g, we have the following set of constraints $\forall g \in [1, n]$, $\forall \sigma_j^k \in s$ such that $\sigma \in P_i^g$

$$t_g^1 + f(\sigma_g^1) \leq t_j^k, \tag{5.1}$$

- Final: Each robot has to enter its final position after all other robots leave it. This means that the last event of each robot must be executed before all events of other robots which involve that position. Thus, call P_f^g the resource of final position of robot g, we have the following set of constraints $\forall g \in [1, n]$, $\forall \sigma_j^k \in s$ such that $\sigma \in P_f^g$

$$t_j^k + f(\sigma_j^k) \leq t_g^{m_g}, \tag{5.2}$$

- Intermediate: For each position through which two robots pass, it has to be required either that the first enter it after the second leave it or vice versa. This has to be required for each couple of robots that access the same intermediate position. Witouth loss of generality, let us suppose that robots g, h share position a position and that thei enter it with event σ_g^i, σ_h^i and leave it with event σ_g^o, σ_h^o. Thus, it has to be required that either event σ_g^i starts after the completion of event σ_h^o, i.e., $t_h^o + f(\sigma_h^o) \leq t_g^i$, or event σ_h^i starts after the completion of event σ_g^o, i.e., $t_g^o + f(\sigma_g^o) \leq t_h^i$.
To express the *XOR*-condition − whose operand is $\oplus$ − between our literals $X_1 = [t_h^o + f(\sigma_h^o) - t_g^i \leq 0]$, $X_2 = [t_g^o + f(\sigma_g^o) - t_h^i \leq 0]$ by linear constraints we refer to [26]. Let δ_1, δ_2 be two logical variables associated to literals X_1, X_2, i.e., $X_1 \Leftrightarrow [\delta_1 = 1]$ and $X_2 \Leftrightarrow [\delta_2 = 1]$. Therefore, require $X_1 \oplus X_2$ is equivalent to require

$$\begin{cases} X_1 \Leftrightarrow [\delta_1 = 1] \\ X_2 \Leftrightarrow [\delta_2 = 1] \\ \delta_1 + \delta_2 = 1 \end{cases}$$

Consider the literals are of the type $X = [f(x) \leq 0]$, where $f(x) : \mathbb{R}^n \to \mathbb{R}$ is a linear function, and assume that $x \in \mathcal{X}$ with $\mathcal{X}$ a given bounded set and define $U = \max_{x \in \mathcal{X}} f(x)$, $L = \min_{x \in \mathcal{X}} f(x)$ and ε is a small tolerance, e.g. the machine precision. Thus,

$$[f(x) \leq 0] \Leftrightarrow [\delta = 1] \iff \begin{cases} f(x) \leq U(1 - \delta) \\ f(x) \geq \varepsilon + (L - \varepsilon)\delta \end{cases}$$

In our case we have $f_1 = t_h^o + f(\sigma_h^o) - t_g^i$, $f_2 = t_g^o + f(\sigma_g^o) - t_h^i$, $U = |s| + 1$ and $L = -|s| + 1$. Furthermore, we name $\delta_1 = \delta_{h,g}$ and $\delta_2 = \delta_{h,g}^{l,k}$. Thus, we have the following set of constraints for each position and each couple of robots $g, h \in [1, n]$ passing through that position

$$\begin{cases} t_h^o + f(\sigma_h^o) - t_g^i \leq U(1 - \delta_{h,g}) \\ t_h^o + f(\sigma_h^o) - t_g^i \geq \varepsilon + (L - \varepsilon)\delta_{h,g} \\ t_g^o + f(\sigma_g^o) - t_h^i \leq U(1 - \delta_{h,g}^{l,k}) \\ t_g^o + f(\sigma_g^o) - t_h^i \geq \varepsilon + (L - \varepsilon)\delta_{h,g}^{l,k} \\ \delta_{h,g} + \delta_{h,g}^{l,k} = 1 \end{cases}. \tag{5.3}$$

Constant values: Whenever U, L appear, they take the following values: $U = |s| + 1$ and $L = -|s| + 1$.

Number of variables: There are $m = |s|$ firing variables, one for each event σ in the string s and also a certain amount of logical variables. An upper bound can be computed by supposing that all robots share the same path but not initial and final position, for a total of $\frac{n(n-1)}{2}$ couples.. Suppose that all robots substrings has m/n events, i.e., $m/n - 1$

Algorithm 1: *HeuristicShuffle*$(s, f, \mathcal{R}, N)$

Input : A valid string $s = \sigma_1 \ldots \sigma_m \in V(G)$,
a transition weight function $f : \Sigma \mapsto \mathbb{R}$,
a set of resources $\mathcal{R} \subseteq 2^{\Sigma}$
an integer number $N \in \mathbb{N}$.
Output: A valid string s'
Set : $s' \leftarrow CompressTrace\,(s, \mathcal{R}, f)$
$H \leftarrow ResMap_{\mathcal{R}}(s)$
$S \leftarrow \emptyset$

1 **for** $q \leftarrow 2$ **to** m **do**
2 $\quad \bar{s} \leftarrow \pi_{R_g}(s'[1:q])$ // $\sigma_q \in R_g$
3 $\quad$ **if** $|\bar{s}| \geq N$ **then**
4 $\quad\quad \hat{s} \leftarrow$ last N events in $\bar{s}$
5 $\quad\quad \hat{q} \leftarrow \max\{First(H, \hat{s}),\ Last(H, \hat{s})\}$
6 $\quad\quad$ **if** $Middle_{\mathcal{R}}(H, s, \hat{s}, \hat{q})$ **then**
7 $\quad\quad\quad S \leftarrow S \cup \{(\hat{s}, \hat{q})\}$
8 **for** $(\hat{s}, \hat{q}) \in S$ **do**
9 $\quad s^* \leftarrow$ string s where $\hat{s}$ is shifted after $\sigma_{\hat{q}}$
10 $\quad s^* \leftarrow CompressTrace\,(s^*, \mathcal{R}, f)$
11 $\quad$ **if** $\upsilon(s^*) < \upsilon(s')$ **then**
12 $\quad\quad s' \leftarrow s^*$
13 **return** s'

intermediate positions, amounting to $\frac{n(n-1)}{2} \cdot (m/n - 2) = \frac{m \cdot n}{2} - \frac{m}{2} - n^2 + n \approx m \cdot n$ logical variables. Thus, the number of unknowns in the MILP 2 is $\mathcal{O}(m \cdot n)$.

Number of constraints: In the same scenario of above, position constraints clearly outnumbers the others and so that intermediate positions. Thus, the number of constraints is $\mathcal{O}(m \cdot n)$. $\square$

VI. HEURISTIC SOLUTION

On the other hand, to solve Problem 1 one can compute all the strings which can be obtained by shuffling events in s, compute their minimal schedule and finally select the one with the smallest strict makespan. This approach would be much more complex because of its combinatorial nature. The main idea underlying the proposed heuristic approach, given in Algorithm 1, is to shuffle events in the string s in a smart way, selecting only shuffles which result in a new valid string.

A previous approach called $CompressTrace$ [27] - used in our algorithm - operates an optimal rearrangement without swapping two events involving the same resource, i.e., if $s' = CompressTrace(s)$ for each resource $r \in \mathcal{R}$ it holds that $\pi_r(s) = \pi_r(s')$. Our algorithm relaxes this constraint for resources associated to positions. Thus, it allows $\pi_{P_j}(s) \neq \pi_{P_j}(s')$ for a generic position p_j with $j = 1, \ldots, N_p$ and associated resource P_j. This relaxation allows robots having a common path to change their priority of access to it, which is the intrinsic idea of the proposed approach.

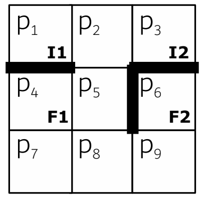


Figure 2: Swap example: I1, I2 and F1, F2 are respectively the initial and final positions of robots 1 and 2.

A. GENERAL IDEA

Given a valid string s, Algorithm 1 allows to identify all subsequences $\hat{s}$ containing N events which can be shifted backwards in position $\hat{q}$ while resulting in a new valid string. This is done at lines $1 - 7$. A subsequence which can be possibly shifted backward in the string is called a *dense subsequence*.

Definition 10 (Dense subsequence) Let $(G, f, \mathcal{R})$ be a STAR, $s = \sigma_1 \cdots \sigma_m \in \mathcal{V}(G)$ and $s \in V(G)$ be a valid string.

A *dense subsequence* of s is any substring $\hat{s}$ of $\pi_{R_g}(s)$, where R_g is the resource of robot r_g. ■

After identifying all subsequence-index couples, Algorithm 1 performs all the shifts and, if there is, returns the one with the smallest strict makespan. This is done at lines $8 - 12$. In order to better understand which kind of couples subsequence-index the algorithm is able to find and how a shuffle can effectively improve a string, an example is given.

Example 2 Referring to Figure 2 a valid string can be $s = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_7\sigma_8 = (1,1,2)(1,2,5)(1,5,4)(2,3,2)(2,2,5)(2,5,8)(2,8,9)(2,9,6)$. The strict makespan of the string s is $\upsilon(s) = 7$ considering all event weights equal to 1. In this example robots r_1 and r_2 pass through positions 2 and 5. In the string s robot 1 has priority of access to it because its events precede those of robots r_2. Chosen $N = 3$, Algorithm 1 is able to find that the subsequence $\hat{s} = \sigma_4\sigma_5\sigma_6 = (2,3,2)(2,2,5)(2,5,8)$ can be moved in position $\hat{q} = 1$. The resulting string is $s' = (2,3,2)(2,2,5)(2,5,8)(1,1,2)(1,2,5)(1,5,4)(2,8,9)(2,9,6)$ and is a valid string. This shift means that now robot 2 has priority of access to the intersection. Thus, $CompressTrace$ computes a new reordering at line 10 and the final string is $s' = (2,3,2)(2,2,5)(1,1,2)(2,5,8)(1,2,5)(2,8,9)(1,5,4)(2,9,6)$ with strict makespan $\upsilon(s') = 5$ which is less than $\upsilon(s) = 7$. ■

B. CORRECTNESS OF THE APPROACH

In the following we first state and prove conditions for a dense subsequence to be shifted in a specific position and then prove that Algorithm 1 outputs a valid string based on these conditions.

Proposition 3 (Shift of a dense subsequence within a string) Let $\mathcal{G} = (G, f, \mathcal{R})$ be a STAR, let $x_0 = (x_{1,0}, \cdots, x_{n,0})$ be the initial state of all robots, let $s = \sigma_1 \dots \sigma_m \in \mathcal{V}(G)$ be a valid string, let $\hat{s} = \sigma_p \dots \sigma_q$ be a dense subsequence of s of robot g, let $\hat{q}$ be an index such that $\hat{q} < p$. Define

- $s_i = s[1 : \hat{q} - 1]$,
- $\bar{s} = s[\hat{q} : m] \setminus \hat{s}$,

such that the string obtained by the shift is $s' = s_i \hat{s} \bar{s}$. Three conditions are defined:

(1) there exists an event $\sigma \in \bar{s}$ such that $\sigma \in R_g$.

(2a) there is a robot h such that $\delta_h(x_{h,0}, \pi_{R_h}(s_i)) = \delta_g(x_{g,0}, \pi_{R_g}(s_i \hat{s}[1 : l]))$ for one $l \in [1, |\pi_{R_g}(\hat{s})|]$.

(2b) there is a robot h such that $\delta_h(x_{h,0}, \pi_{R_h}(s_i \bar{s}[1, l])) = \delta_g(x_{g,0}, \pi_{R_g}(s_i \hat{s}))$ for one $l \in [1, |\pi_{R_h}(\bar{s})|]$

Let be s' the string obtained by the shift, then we have the following statement

(i) The dense subsequence $\hat{s}$ can be shifted in position $\hat{q}$ within string s, getting $s' \in V(G)$, if and only if (1), (2a), (2b) are false. □

Proof. Condition (1) is equivalent to a shift which implies a shuffle of events belonging to the same robot. In this case, the resulting string does not belong to $L(G)$. Thus $s' \in L(G)$ if and only if (1) is false. By definition, s_i is a valid string. String $s_i \hat{s}$ is a valid string if and only if (2a) is false. String $\hat{s}\bar{s}$ is a valid string if and only if (2b) is true. Thus, $s_i \hat{s} \bar{s} = s' \in V(G)$ if and only if $s' \in L(G)$ and (2a) and (2b) are false. Therefore, $s' \in V(G)$ if and only if (1), (2a) and (2b) are false.

□

Algorithm 1 computes subsequence-index couples by means of functions $First$, $Middle_{\mathcal{R}}$ and $Last$ which are defined next.

Let $s \in V(G)$ be a valid string, $\mathcal{R}$ a standard set of resources, and $\hat{s} = \sigma_p \dots \sigma_q$ a dense subsequence of s. For the sake of simplicity, the next functions are defined with respect to an event with an arbitrary index $\sigma_k = (g_k, p_k^i, p_k^f)$.

$$ResMap_{\mathcal{R}}(s) = H_{m \times |\mathcal{R}|},$$

where each element $h_{ab} = a$ if $\sigma_a \in \{R_b \cup P_{b-n}\}$ and $h_{ab} = 0$ otherwise.

$$First(H, \hat{s}) = \max\{h_{ab} | a < p, b = g_p\} \cup \{0\}.$$

$$Last(H, \hat{s}) = \max\{h_{ab} | a \notin I_g \vee a < q, b = p_q^f + n\} \cup \{0\},$$

where $I_g = \{p, \dots, q\}$.

$Middle_{\mathcal{R}}(H, s, \hat{s}, \hat{q}) = true$ if $q^* = 0$ and $false$ otherwise,

where $q^* = \max\{h_{ab} | \sigma_a \in s^*, b - n \in I_i\}$ and $I_i = \{p_p^i, \dots, p_q^i\}$ and $s^* = \{t[1 : 1] | t = \pi_{R_g}(s[\hat{q} + 1 : q] \setminus \hat{s}), \forall R_g \in \mathcal{R}\}$.

Theroem 1 (Algorithm 1 outputs a valid string) Let $(G, f, \mathcal{R})$ be a STAR, $s = \sigma_1 \cdots \sigma_m \in \mathcal{V}(G)$ a valid string and let $s' = HeuristicShuffle(s, f, \mathcal{R}, N)$.

Then $s' \in V(G)$.

Proof. Given a dense subsequence $\hat{s} = \sigma_p \cdots \sigma_q$ of the string s, Algorithm 1 selects an index $\hat{q}$ by using functions $First$ and $Last$ at line 5 and then validates it by function $Middle_{\mathcal{R}}$. It is necessary to ensure that such indexes are feasible, in the sense that the new string obtained by shifting back $\hat{s}$ after $\sigma_{\hat{q}}$ results in a valid string. By Proposition 3 we know that if a couple $(\hat{s}, \hat{q})$ satisfies conditions (1), $(2a)$ and $(2b)$, then the new string it is a valid string. We point out that:

- Given $\hat{q} = First(H, \hat{s})$ and $\bar{s} = s[\hat{q} : q] \setminus \hat{s}$, then condition (1) of Proposition 3 is false.
- Given $\hat{q} = Last(H, \hat{s})$ and $\bar{s} = s[\hat{q} : q] \setminus \hat{s}$, then condition $(2b)$ of Proposition 3 is false.
- Given $\hat{q} = 1, \dots, m$, $v = Middle_{\mathcal{R}}(H, s, \hat{s}, \hat{q})$ and $\bar{s} = s[\hat{q} : q] \setminus \hat{s}$, then condition $(2b)$ of Proposition 3 is false if and only if $v = true$.

Therefore, if a couple $(\hat{s}, \hat{q})$ is added at line 7, it is sure that satisfies conditions (1), $(2a)$ and $(2b)$ and then $s' = HeuristicShuffle(s, f, \mathcal{R}, N)$ is a valid string, i.e. shifting $\hat{s}$ in position $\hat{q} + 1$ leads to a valid string. □

Example 3 Given the valid string $s = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_7\sigma_8 = (2, 2, 3)(1, 8, 9)(1, 9, 6)(1, 6, 5)(1, 5, 4)(2, 3, 6)(1, 4, 1)(2, 6, 9)$, let $\hat{s} = \sigma_6\sigma_8 = (2, 3, 6)(2, 6, 9)$ be a subsequence of s. First we compute matrix H with function $ResMap_{\mathcal{R}}(s)$: "

$$H = \begin{array}{c|ccccccccccc|}
 & R_1 & R_2 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 & P_7 & P_8 & P_9 \\
\hline
\sigma_1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\sigma_2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 \\
\sigma_3 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 3 \\
\sigma_4 & 4 & 0 & 0 & 0 & 0 & 0 & 4 & 4 & 0 & 0 & 0 \\
\sigma_5 & 5 & 0 & 0 & 0 & 0 & 5 & 5 & 0 & 0 & 0 & 0 \\
\sigma_6 & 0 & 6 & 0 & 0 & 6 & 0 & 0 & 6 & 0 & 0 & 0 \\
\sigma_7 & 7 & 0 & 7 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 \\
\sigma_8 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 8
\end{array}$$

We now compute function $First(H, \hat{s})$ which selects all entries h_{ab} of H such that $a < p = 6$ and $b = g_p = 2$ (because $\hat{s} = \sigma_p \cdots \sigma_q = \sigma_6\sigma_8$), giving the set $\{0, 1\}$, then it returns the maximum value 1. This means that subsequence $\hat{s}$ can not be moved before event σ_1. We now compute function $Last(H, \hat{s})$ which selects all entries h_{ab} of H such that $a \notin I_g = \{6, 8\}$, $a < 8$ and $b = f_q + n = 9 + 2 = 11$ giving the set $\{0, 2, 3\}$, then it returns the maximum value 3. This means that subsequence $\hat{s}$ can not be moved before event σ_3. We select the maximum value $\hat{q} = 3$ between the two values returned by $First$ and $Last$. Now we must evaluate $\hat{q}$ with function $Middle_{\mathcal{R}}(H, s, \hat{s}, \hat{q})$. First it computes $s^* = \{\sigma_4\}$ and $I_i = \{3, 6\}$, then S takes all entries h_{ab} of H such that $\sigma_a \in s^*$ (in this case $a = 4$) and $b - n \in I_i$ giving $q^* = \max\{0, 4\} = 4$, then it returns $false$ because $q^* \neq 0$. This means that $\hat{s}$ can not be shifted. ■

VII. COMPLEXITY ANALYSIS

In this section the complexity of Algorithm 1 is analyzed.

The first for-loop is executed m times, where m is total number of robots' movements. We approximate each robots' journey with the maximum shortest distance between any two positions in the environment, which, considering a square grid without obstacles with side length $\sqrt{N_p}$, leads to $m \approx n\sqrt{2N_p}$. The projection operated at line 2 requires q iterations, while the slicing operated at line 4 requires N iterations. At line 5 the maximum between two values is taken, which has constant time complexity, while the two called functions $First$ and $Last$ have both $\mathcal{O}(m)$ complexity. Then, at line 6, the if-condition calls function $Middle$ whose complexity is $\mathcal{O}(m \cdot n)$. Since all these operations are executed serially, the overall loop complexity is equal to $\mathcal{O}(m \cdot n)$ since $q \leq m$ and $N \leq m$. Second for-loop is executed m times because in the worst case tests at lines 6 and 9 succeed and exactly m tuples are added in S. The most complex operation is $CompressTrace$ whose complexity is $\mathcal{O}(m \cdot (n + N_p))$, see [27]. The overall loop complexity is equal to $\mathcal{O}(m^2 \cdot (n + N_p)) \approx \mathcal{O}(m^2 \cdot N_p)$. The slowest for-loop is the second because $N_p \gg n$, i.e., $\mathcal{O}(m^2 \cdot N_p) \approx \mathcal{O}(n^2 N_p)$. Finally, considering the number of robots n be upper-bounded by the number of positions N_p, the time complexity of Algorithm 1 is $\mathcal{O}(N_p^3)$.

As mentioned in Section II, most of rule-based algorithms to solve MRPP problems do not have optimality guarantees but their solution can be provided in $\mathcal{O}(N_p^3)$ time, bound given by [23], equal to the time complexity of the proposed approach.

VIII. SIMULATION RESULTS

In order to test the effectiveness of the presented algorithm we generated random MRPP problems in three different environment: Cyclic Corridor, Closed Corridor, and Extensible Grid depicted in Figures 3, 4 and 5.

To generate a random problem instance with n robots we selected a random start and goal position amongst all the possible free positions in the environment, such that no two robot share the same start position, nor do they share the same goal position.

It is possible for a robot to have its start position as its goal position and another robot's start position as its goal position. Velocity of robots is constant but two robots may have different velocities. In particular, time required two move between two cell may vary between 1 second and 2 seconds, i.e., one robot can move at most twice as fast as another robot. For each setup (number of robots and environment) 50 different problems were generated and a solution to them is computed by [10]. For each problem to which a solution was found:

1) We solve Problem 2 with a MILP problem formulation in Proposition 2. When the complexity of the problem becomes computationally intractable, we exploit a lower bound for the optimal solution by choosing the maximum traveling time among all robots.
2) We applied Algorithm 1 to find a sub-optimal solution to Problem 1 and evaluated the distance to the optimal solution or to the lower bound for large problems.

Average results of these simulations are depicted in Figures 6-7, where the x-axis represents the number of robots being coordinated and y-axis represents the average ratio of the execution time compared to our lower bound (or optimal solution when available).

Furthermore, we analyzed the time required by the optimal and sub-optimal approach (see Figure 8) by fixing the ratio

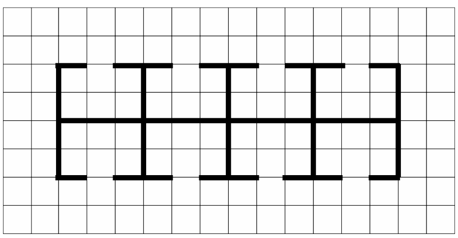


Figure 3: Open Corridor

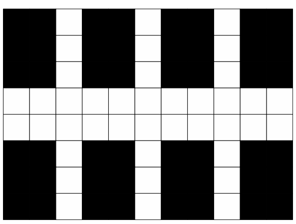


Figure 4: Closed Corridor

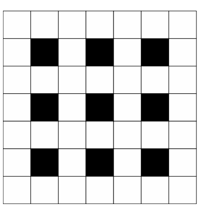


Figure 5: Extensible Grid Environment

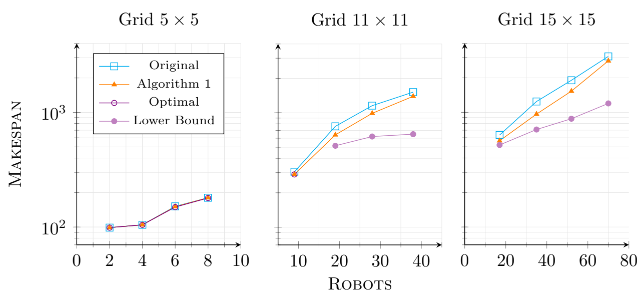
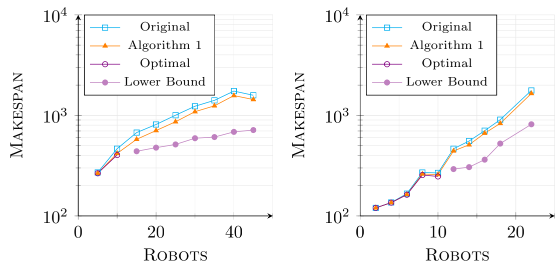


Figure 6: Simulation results for the exensible grid environment



(a) Open Corridor

(b) Closed Corridor

Figure 7: Simulation results for the corridor environments

between number of positions and robots at 20% and enlarging the size of the grid environment up to 11×11.

IX. CONCLUSIONS

Optimal and heuristic approaches to minimize the makespan of pre-planned robot's trajectories are proposed and proved to be guaranteed to return a feasible solution if there is one. The improvement is achieved through the computation of new time schedules by addressing the collision avoidance problem in a discrete event formulation [10] [9] of the MRPP problem, which makes use of time-weighted automata.

The heuristic algorithm was tested on a variety of problems and it was shown, both theoretically and by simulations, that its use always leads to a lower makespan, when it is possible, getting very close to the optimal value. When the optimal solution was not available, due to the complexity of the problem, we compared heuristic solutions to a specific lower bound. Such an heuristic approach does not require a significant amount of time with respect to the time required to compute an optimal or sub-otpimal solution.

The speed of the proposed heuristic strategy leads us to regard potential its use in a dynamic context, which would be a future prosecution of this work.

References

[1] J. van den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, ser. Springer Tracts in Advanced Robotics, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Springer Berlin Heidelberg, 2011, vol. 70, pp. 3–19.

[2] H. Yu, P. Shi, C.-C. Lim, and D. Wang, "Formation control for multi-robot systems with collision avoidance," *International Journal of Control*, vol. 92, no. 10, pp. 2223–2234, 2019.

[3] C. Wang, W. Liu, and M. Q.-H. Meng, "Obstacle avoidance for quadrotor using improved method based on optical flow," in *IEEE International Conference on Information and Automation*, 2015, pp. 1674–1679.

[4] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *arXiv preprint arXiv:1910.11689*, 2019.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

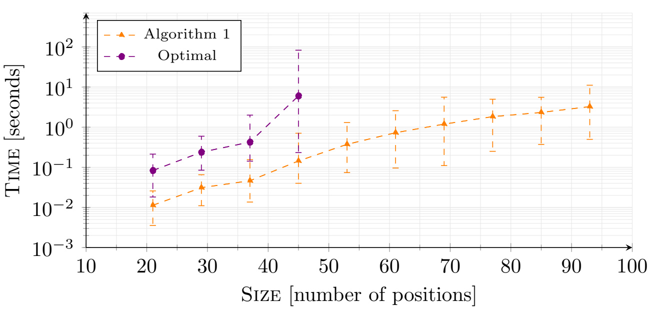


Figure 8: Chart showing average time required to compute an optimal solution through MILP in Proposition 2 and a sub-optimal through Algorithm 1 vs size of the environment whit a congestion equal to to 20%. We considered a failure when comptuing the optimal solution took more than 5 minutes: it happened when we tried to coordinate more than 10 robots.

[6] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[7] P. MacAlpine, E. Price, and P. Stone, "Scram: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[8] J. Hopcroft, J. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem"," *The International Journal of Robotics Research*, vol. 3, pp. 76–88, 1984.

[9] D. Deplano, S. Ware, R. Su, and A. Giua, "A heuristic algorithm to optimize execution time of multi-robot path," in *13th IEEE International Conference on Control Automation*, July 2017, pp. 909–914.

[10] S. Ware and R. Su, "Incremental scheduling of discrete event systems," in *2016 13th International Workshop on Discrete Event Systems*, May 2016, pp. 147–152.

[11] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[12] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination," *International Journal of Advanced Robotic Systems*, vol. 10, no. 12, p. 399, 2013.

[13] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[14] D. Silver, "Cooperative pathfinding." *AIIDE*, vol. 1, pp. 117–122, 2005.

[15] J. H. Oh, J. H. Park, and J. T. Lim, "Centralized decoupled path planning algorithm for multiple robots using the temporary goal configurations," in *2011 Third International Conference on Computational Intelligence, Modelling Simulation*, 2011, pp. 206–209.

[16] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 5954–5961.

[17] G. Sanchez and J. C. Latombe, "Using a prm planner to compare centralized and decoupled planning for multi-robot systems," in *IEEE International Conference on Robotics and Automation*, vol. 2, 2002, pp. 2112–2119.

[18] M. Cap, P. Novak, A. Kleiner, and M. Selecky, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Transactions on Automation Science and Engineering*, vol. 12, pp. 835–849, 2015.

[19] M. R. K. Ryan, "Exploiting subgraph structure in multi-robot path planning," *Journal of Artificial Intelligence Research*, vol. 31, pp. 497–542, 2008.

[20] P. Surynek, "A novel approach to path planning for multiple robots in bi-connected graphs," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 3613–3619.

[21] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[22] B. De Wilde, A. W. Ter Mors, and C. Witteveen, "Push and rotate: a complete multi-agent pathfinding algorithm," *Journal of Artificial Intelligence Research*, vol. 51, pp. 443–492, 2014.

[23] D. Kornhauser, G. Miller, and P. Spirakis, "Coordinating pebble motion on graphs, the diameter of permutation groups, and applications," in *25th Annual Symposium on Foundations of Computer Science, 1984.*, 1984.

[24] G. Röger and M. Helmert, "Non-optimal multi-agent pathfinding is solved (since 1984)," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[25] J. Yu and D. Rus, "Pebble motion on graphs with rotations: Efficient feasibility tests and planning algorithms," in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 729–746.

[26] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407 – 427, 1999.

[27] S. Ware and R. Su, "An application of incremental scheduling to a cluster photolithography tool," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 1114 – 1120, 2017, 20th IFAC World Congress.

DIEGO DEPLANO received the B.S. and M.S. degrees in Electronic Engineering "cum laude" from the University of Cagliari, Italy, respectively in 2015 and 2017. He spent visiting periods at the Nanyang Technological University (NTU), Singapore, at the Centre National de la Recherche Scientifique (CNRS), Grenoble, France, and at the University of Toronto (UofT), Toronto, Canada. He is currently pursuing a Ph.D. degree in Electronic Engineering and Computer Science at the Department of Electrical and Electronic Engineering, University of Cagliari, Italy. His research interests include nonlinear multi-agent systems, positive systems, consensus problems and mobile robotics.

MAURO FRANCESCHELLI (M'11) is Senior Assistant Professor (RTD-B) at the Department of Electrical and Electronic Engineering, University of Cagliari, Italy, since 2019. He received the Laurea degree in Electronic Engineering "cum laude" in 2007 and the PhD degree in 2011 from the University of Cagliari. He spent visiting periods at the Georgia Institute of Technology (GaTech), and the University of California at Santa Barbara (UCSB), USA. In 2013 he received a fellowship from the National Natural Science Foundation of China (NSFC), Grant No. 61450110086, at Xidian University, Xi'an, China. In 2015 he was awarded a position of Assistant Professor (RTD-A) funded by the Italian Ministry of Education, University and Research (MIUR) under the 2014 call "Scientific Independence of Young Researchers" (SIR) with project "CoNetDomeSys", code RBSI14OF6H. He is member of the Conference Editorial Board (CEB) for the IEEE Control Systems Society (CSS) since 2019. He serves as Associate Editor for the IEEE Conference on Automation Science and Engineering (CASE) since 2015, the IEEE American Control Conference (ACC) since 2019 and IEEE Conference on Decision and Control since 2020. His research interests include consensus problems, gossip algorithms, multi-agent systems, multi-robot systems, distributed optimization and electric demand side management.

SIMON WARE received his bachelors of computing and mathematical sciences with honours, as well as his PhD at the University of Waikato in 2008 and 2014 respectively. The majority of his research work has been focused on model verification, supervisory control, and optimization. This was primarily pursued as a research fellow at Nanyang Technological University from 2014 to 2018. He is currently working on allocation problems as a data scientist at Grab Holdings Inc.

RONG SU (M'11-SM'14) obtained the Bachelor of Engineering degree from University of Science and Technology of China in 1997, and the Master of Applied Science degree and PhD degree from University of Toronto in 2000 and 2004, respectively. He was affiliated with University of Waterloo in Canada and Eindhoven University of Technology in the Netherlands before he joined Nanyang Technological University in 2010. Currently, he is an associate professor in School of Electrical and Electronic Engineering. Dr Su's research interests cover areas of discrete-event system theory, including (networked) supervisory control, cyber security analysis and model-based fault diagnosis, consensus control of multi-agent systems, and real-time optimization in complex networked systems with applications in smart manufacturing, intelligent transportation systems, and green buildings. In the aforementioned areas he has 75 journal publications and more than 112 conference publications, and 2 granted USA/Singapore patents. Dr Su is a senior member of IEEE, and an associate editor for Automatica (IFAC), Journal of Discrete Event Dynamic Systems: Theory and Applications, and Journal of Control and Decision. He was the Chair of the Technical Committee on Smart Cities in the IEEE Control Systems Society in 2016 - 2019, and is currently the Chair of IEEE Control Systems Chapter, Singapore.

ALESSANDRO GIUA (F'17) is professor of Automatic Control at the Department of Electrical and Electronic Engineering (DIEE) of the University of Cagliari, Italy. He received a laurea degree from the University of Cagliari in 1988 and a Master and Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute (Troy, NY, USA) in 1990 and 1992. He has also held faculty or visiting positions in several institutions worldwide, including Aix-Marseille University, France and Xidian University, Xi'an, China. His research interests include discrete event systems, hybrid systems, networked control systems, Petri nets and failure diagnosis.

He serves the Institute of Electrical and Electronic Engineers (IEEE) as Vice President for Conference Activities of the Control Systems Society (CSS), having previously held the roles of General Chair of the 55th Conference on Decision of Control (2016) and member of the CSS Board of Governors (2013-15). He also serves the International Federation of Automatic Control (IFAC) as a member of the Publications Committee and has also served as chair of the IFAC Technical Committee 1.3 on Discrete Event and Hybrid Systems (2008-14).

He is currently editor in chief of the IFAC journal Nonlinear Analysis: Hybrid Systems, a senior editor of the IEEE Trans. on Automatic Control, and a department editor of the Springer journal Discrete Event Dynamic Systems.

He is a Fellow of both the IEEE and IFAC for contributions to Discrete Event and Hybrid Systems, a recipient of the IFAC Outstanding Service award and a laureate of the People's Republic of China Friendship Award.