A Heuristic Algorithm to Optimize Execution Time of Multi-Robot Path

Diego Deplano, Simon Ware, Rong Su, Alessandro Giua

***Abstract*—A common problem in the field of robotics is to coordinate motions of multiple robots to ensure the shortest possible execution time. The problem is known PSPACE-complete, thus, it is impossible to find the best solution in a reasonable time for large scale problems. For this reason, in this work we look for sub-optimal solutions by systematically improving a given one. We present a heuristic algorithm to reduce execution time of a path by changing robots' priorities in case of path overlap. The algorithm is applied to a solution computed by a decoupled method in a discrete event system context. It is shown that the proposed approach is effective in finding a solution with shorter execution time. Tests show that the proposed algorithm can achieve improvement up to 45%.**

I. Introduction

The multi-robot path-planning (MRPP) problem is a fundamental task in robotics. In MRPP the aim is to plan a path for multiple robots attempting to move through the same environment. In such problems, each robot has a speci fic start position and has to reach its goal position so that all robots reach their goal position in the minimum time [1], not necessarily in the minimum number of moves. The difficulty of finding this solution lies in the fact that when multiple robots move through a shared environment to perform independent tasks each one will become a mobile obstacle for the others, therefore the motion planning of each individual robot has to take into account motion of the others.

A MRPP problem can be solved by coupled or decoupled methods. The former often try to provide a complete solution [2] [3] by searching the path through a combined state space of all robot, but have high requirements for time computation because of the PSPACE-complexity of the problem [4]. The latter reduce the problem complexity by calculating paths independently for each robot [5] [6] which are then coordinated in a unique path avoiding all possible conflicts [7]. However they cannot guarantee to find a solution in all cases [8], except under certain conditions [9], and cannot guarantee optimal solution.

Of particular interest is to understand how and in which cases a given not optimal path can be improved in terms of time execution. In this paper we give a formal description of the problem in a discrete event system (DES) context and analyze in which cases a path can be improved by rearranging movements within a path.

This paper improves upon [10] where some of us introduced a method for further optimizing a path by identifying single movement which can be swapped with another one, whereas here we propose a new method which is based on prioritizing multiple movements of the same robot with respect to other movements. This allows us to find new solutions because the proposed method is able to switch robots' priorities in case of path overlap. The new path is guaranteed to be valid, assuming the original path was valid, but results in a better execution time.

This paper is divided into six sections. In Section II we first provide all relevant and necessary concepts about languages and time-weighted automata. Section III gives a thorough description of the problem we are solving and how it is represented in DES. Section IV gives the algorithms used to improved the planned path. Section V shows experimental resuls for our algorithm. Section VI gives some concluding remarks.

II. Preliminaries

Event sequences and languages are a simple means to describe discrete event system behaviours. Their basic building blocks are *events* σ, which are taken from an *alphabet* Σ.

A *string* is a sequence of events $s = \sigma_1 \dots \sigma_n$. The set of all strings of events from Σ is denoted Σ^* which includes the *empty string* ε. A *language* is a set of string $L \subseteq \Sigma^*$. The *concatenation* of two strings $s, t \in \Sigma^*$ is written as st. A *substring* of s is denoted $s[p:q]$ where p, q are the first and last event indexes of s. System behaviours 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*, $\delta : X \times \Sigma \mapsto X$ is the (partial) transition function, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of marked states. The transition relation is extended to strings in Σ^* by letting $\delta(x, \epsilon) = x$ for all $x \in X$, and $\delta(x, s\sigma) = \delta(\delta(x, s), \sigma)$. Furthermore $!\delta(x, \sigma)$ means that exists y such that $\delta(x, \sigma) = y$, i.e. $\delta(x, \sigma)$ is defined.

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

A *resource* is a set of events which cannot be executed simultaneously $r \subset \Sigma$. For every pair of events $\sigma, \sigma' \in \Sigma$ there exists a resource $r \in R$ such that $\sigma, \sigma' \in r$ if and only

D. Deplano is with School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798 and also with DIEE, University of Cagliari, 09123 Cagliari, Italy. Email: di.deplano1@studenti.unica.it

S. Ware and R. Su are affiliated with Division of Control and Instrumentation, School of Electrical and Electronic Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798. Email: {sware,rsu}@ntu.edu.sg

A. Giua is with Aix Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille, France (email: alessandro.giua@lsis.org) and and DIEE, University of Cagliari, 09123 Cagliari, Italy (email: giua@diee.unica.it).

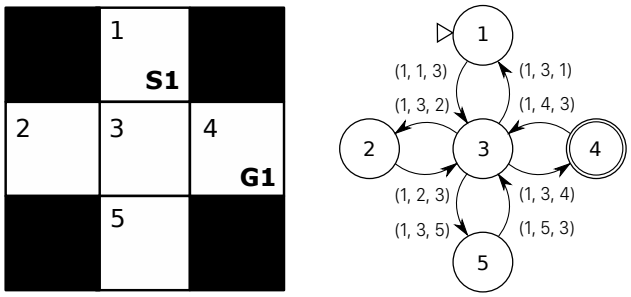

Figure 1: Translation of MRPP problem in a DES context: on the left a sample of environment and on the right an automaton representing a robot with start and goal position.

if the firings of σ and σ' are mutually exclusive, i.e. if one event is under execution the other cannot be fired.

The *projection* of a string s over a resource r returns a new string denoted $proj_r(s)$ where events not requiring that resource are removed. The *co−projection* of a string s over a resource r returns a new string denoted *co-*$proj_r(s)$ where events requiring that resource are removed.

A *finite-state time-weighted automaton* is a 3-tuple $\mathcal{G} = (G, f, R)$, where G is a finite-state automaton, $f : \Sigma \mapsto \mathbb{R}^+$ is a *weight* function which assigns to each event in Σ a positive real number called weight which denotes the time required to execute the corresponding event, and $R \subseteq 2^\Sigma$ is a set of resources.

When a string s is executed the events are fired sequentially based on the order in the string. Therefore, each event $\sigma_k \in s$ has a *starting time* t_k^s of firing and an *execution time* $f(\sigma_k)$, so that its *completion time* is $t_k^s + f(\sigma_k)$.

A *time-stamp* of the string s with respect to G which establishes the starting time of each event is a nondecreasing list of nonnegative real numbers $\rho_s = [t_1^s, \ldots, t_n^s] \subset (\mathbb{R}_0^+)^*$ where $t_k^s \leq t_{k+1}^s$ for $k \in \{1, \ldots, n\}$. Several time-stamps may correspond to a string, we denote $\mathcal{P}_G(s)$ the set of all time-stamps of s with respect to G.

A *legal time-stamp* of the string s with respect to $\mathcal{G} = (G, f, R)$ is any time-stamp $\vartheta \in \mathcal{P}_G(s)$ such that for all $q, v \in \{1, \cdots, n\}$ and $q < v$ if $\sigma_q, \sigma_v \in r$ and $t_k^s + f(\sigma_k) \leq t_v^s$. We denote $\Theta_{G,f,R}(s)$ the set of all legal time-stamps of s.

The execution time of a string s is measured from the time the first robot starts moving to the time at which the last robot reaches its destination. From now on we suppose $t_1^s = 0$, i.e., we start from the time when the first robot moves.

Definition 1. Let $\mathcal{G} = (G, f, R)$ be a time-weighted automaton and let $s = [\sigma_1, \ldots, \sigma_n] \in L(G)$ be a string.

The minimum execution time of the string s is $\upsilon_{G,f,R}(s) = \min_{\vartheta \in \Theta_{G,f,R}(s)} \{\max\{t_1^s + f(\sigma_1), \cdots, t_n^s + f(\sigma_n)\}\}$.

As a convention, $\upsilon_{G,f,R}(\epsilon) := 0$. □

The minimum execution time of the string s can be computed as described in [10].

III. Problem Formulation

The environment where a robot operates can be partitioned into a grid, where white squares denote accessible positions and black squares denote inaccessible positions. A robot can move from one square to another one if the two squares are adjacent, i.e., have a common side, if the transition is avalaible, i.e., the side is not emboldened, and if the arrival square is empty. We construct a timed automata model $\mathbb{G}$ representing multiple robots navigating this envirenment by creating a timed automaton $\mathcal{G} = (G, f, R)$ for each robot.

We first define an automaton G which represents the possible movements of the g-th robot, where each state x_j represents the discrete j-th position in the environment which the robot can move to and each event $\sigma = (g, i, f) \in \Sigma$ represents the act of the g-th robot moving from the i-th position to the f-th position. The initial state of the automaton is set to be the start position of g-th robot and the marked state is set to be the goal position. In Figure 1 it is shown an example of an environment and a robot (here $g = 1$) moving in it.

A path which potentially can direct a set of robots from their start position to their goal position is a particular string termed *trace*, i.e. a sequence of movements which when executed directs all robots to reach their destinations without collisions.

Definition 2. Let $\mathbb{G} = \{G_1, \ldots, G_n\}$ be a set of automata, let $G = G_1 \times G_2 \times \ldots \times G_n$ let be the composite automaton of the set $\mathbb{G}$, let $x_0 = (x_{0,1}, \cdots, x_{0,n})$ be the state tuple of initial states of all automata and let $s \in L(G)$ be a string generated by the composite automaton G.

The string s is a trace if $s \in L_m(G)$, i.e. if the string s is marked by the composite automaton G. □

Secondly, we define the weight function f as a function which assigns a weight to all events $\sigma \in \Sigma$ equal to the time required by the robot to execute the movement.

Finally we define the set of all resources R of the system. For each robot automaton $G_g \in \mathbb{G}$, where $\mathbb{G}$ is the set of all robot automata, there exist a robot-resource r_{G_g} such that each movement event of the g-th robot is in that resource. For each position $P_j \in \mathbb{P}$, where $\mathbb{P}$ is the set of all positions, there exist a position-resource r_{P_j} such that each movement event (across all robots) which involves the j-th position is in that resource. An event $\sigma = (g, i, f)$ describes motion of the g-th robot, modelled by the G_g automaton, from the position P_i to the position P_f, therefore this event requires three resources, one robot-resource and two position-resources.

The set of resources R ensures that a robot can not enter a position while another robot is vacating it, but we also wish to ensure that a robot cannot enter a position if another robot is still in it. For this reason we define the function $collisionfree$, which takes as argument a state tuple $(x_1, \ldots, x_n)$, where each x_g is the current state of automata g and evaluates to true if and only if for every distinct pair of indices i, j it holds that $x_i \neq x_j$. This is equivalent to saying that two robots cannot be in the same position at the same time.

Algorithm 1 SwapTrace

Require: A valid trace $t = \sigma_1 \dots \sigma_{|s|} \in \Sigma^*$, a set of resources $R \subseteq 2^\Sigma$, a transition weight function $f : \Sigma \mapsto \mathbb{R}$, an integer number N.

1: $s, v \leftarrow CompressTrace\,(t, R, f)$
2: $S \leftarrow \emptyset$
3: **for** $q \in \{1, \dots, |s|\}$ **do**
4: $(g, i, f) \leftarrow \sigma_q$
5: $\bar{s} \leftarrow proj_{r_{G_g}}(s[1:q])$
6: **if** $|\bar{s}| \geq N$ **then**
7: $\hat{s} \leftarrow \bar{s}[|\bar{s}| - N + 1 : \bar{s}]$ ▷ last N events
8: Let be σ_p and σ_q the first and last event in $\hat{s}$
9: Determine, if there is, the lowest $\hat{q} < p$ such that $\hat{s}$ and $\sigma_{\hat{q}}$ are independent. ▷ see Definition 5
10: **if** $\hat{q}$ exists **then**
11: $S \leftarrow S \cup \{(\hat{s}, \hat{q})\}$
12: **end if**
13: **end if**
14: **end for**
15: **for** $(\hat{s}, \hat{q}) \in S$ **do**
16: Events in $\hat{s}$ belong to the g-th robot
17: $s^* \leftarrow co\text{-}proj_{r_{G_g}}(s)$
18: $s' \leftarrow s^*[1 : \hat{q} - 1] \cdot \hat{s} \cdot s^*[\hat{q} : |s^*|]$
19: $s', v' \leftarrow CompressTrace\,(s', R, f)$
20: **if** $v' < v$ **then**
21: $s, v \leftarrow t', v'$
22: **end if**
23: **end for**
24: **return** s

Definition 3. Let $\mathbb{G} = \{G_1, \dots, G_n\}$ be a set of automata, let $G = G_1 \times G_2 \times \dots \times G_n$ let be the composite automaton of the set $\mathbb{G}$, let $x_0 = (x_{0,1}, \cdots, x_{0,n})$ be the state tuple of initial states of all automata and let $s \in L_m(G)$ be a trace.

The trace s is valid if for every string $p \in \Sigma^*$ such that $p \leq s$ it holds $collisionfree(\delta_{\mathbb{G}}(x_0, p))$. We denote $V(\mathbb{G})$ the set of all valid traces of $\mathbb{G}$. □

Given the above setup, our problem is to improve a valid trace by reducing its minimum execution time.

Problem 1. Let $\mathbb{G} = \{G_1, \dots, G_n\}$ be a set of automata, let $\mho = \{f_1, \dots, f_n\}$ be a set of weight functions, let $R \in 2^\Sigma$ be a set of resources and let $s \in V(\mathbb{G})$ be a valid trace.

How can we find a valid trace $s' \in V(\mathbb{G})$ such that $\upsilon_{\mathbb{G},\mho,R}(s')$ is less than $\upsilon_{\mathbb{G},\mho,R}(s)$? □

IV. Algorithm

The basic concept of Algorithm 1 is to change robots' priorities in case of path overlap between at least two robots by moving backward a subsequence of events within a trace. Given a valid trace s, Algorithm 1 allows to identify all subsequences $\hat{s}$ containing N events which can be moved backward in the trace and the first event $\sigma_{\hat{q}}$ before which they can be moved. This is done at lines $3 - 14$. A subsequence which can be potentially moved backward in the trace is called a *dense* subsequence.

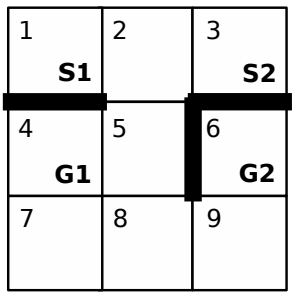

Figure 2: Swap example: S1, S2 and G1, G2 are respectively start and goal positions of robots 1 and 2.

Definition 4. Let $\mathbb{G} = \{G_1, \dots, G_n\}$ be a set of automata, let $R \subseteq 2^\Sigma$ be a dependance satisfying resource set with respect to $\mathbb{G}$, let $t = \sigma_1 \dots \sigma_{|s|} \in V(\mathbb{G})$ be a valid trace, let Σ_g be the alphabet of the g-th robot.

A dense subsequence of t is any subsequence $\hat{s} = [\sigma_i | \sigma_i \in s]$ such that $\forall \sigma \in \hat{s}$ then $\sigma \in \Sigma_g$ and given two subsequent events $\sigma_x, \sigma_z \in \hat{s}$ there no exists an event $\sigma_y \in s$ such that $x < y < z$ and $\sigma_y \in \Sigma_k$. □

If a dense subsequence $\hat{s}$ can be moved before an event $\sigma_{\hat{q}}$ getting a new valid trace, we call them *independent*.

Definition 5. Let $\mathbb{G} = \{G_1, \dots, G_n\}$ be a set of automata, let be $t = \sigma_1 \dots \sigma_{|s|} \in V(\mathbb{G})$ be a valid trace, let $\hat{s} = \sigma_p \dots \sigma_q$ be a dense subsequence of t, let be $\sigma_{\hat{q}}$ an event such that $\hat{q} < p$

The dense subsequence $\hat{s}$ and the event $\sigma_{\hat{q}}$ are independent if the string s resulting from $\hat{s}$ being moved before $\sigma_{\hat{q}}$ in t is a valid trace, i.e. $s \in V((G))$. □

After identifying all couples subsequence-index, Algorithm 1 perform all the swaps and, if there is, return the one which gives the shortest execution time. This is done at lines $15 - 25$.

Algorithm 1 makes use of another algorithm called $CompressTrace$ [11] which optimizes a valid trace, in terms of minimum execution time, by rearranging events within it without changing robots' priorities. This rearrangement is done by identifying single events which can be moved backward in the trace. After the rearrangement, it also computes the minimum execution time v of the resulting string s.

In order to better understand which kind of couples subsequence-index the algorithm is able to find, an example is given.

Example 1. Referring to Figure 2 a valid trace can be $t = (1,1,2)(1,2,5)(1,5,4)(2,3,2)(2,2,5)(2,5,8)(2,8,9)(2,9,6)$. The minimum execution time of the string s is $v = 8$ considering all event weights equal to 1. In this example robots 1 and 2 share a path (i.e. states 2-5) and robot 1 has priority of access to this. Chosen $N = 3$, Algorithm 1 is able to find that the subsequence $\hat{s} = (2,3,2)(2,2,5)(2,5,8)$ can be moved before event $\sigma_{\hat{q}} = (1,1,2)$ without giving rise to collisions. This swap means that now robot 2 has priority

Table I: Results for Cyclic Corridor Environment

	Unit	Results							
robots		5	10	15	20	25	30	35	40
congestion	%	3	7	11	15	19	23	27	31
$S_c \equiv S_{sc}$	%	100	100	100	100	100	100	100	96
S_{sw}	%	100	100	100	100	100	100	100	96
S_{bs}	%	100	100	100	100	100	100	100	98
T_c	s	0.26	1.42	4.23	9.44	17.01	27.78	39.28	51.24
T_{sc}	s	0.36	1.72	4.77	10.25	18.02	29.03	40.51	52.33
T_{sw}	s	0.5	2.67	7.37	15.55	27.04	42.53	58.68	75.01
T_{bs}	s	0.86	4.39	12.14	25.8	45.06	71.56	99.2	127.34
μ_c		18.03	20.3	22.48	25.2	28.27	33.14	37.98	48.07
μ_{sc}		17.95	20.08	22.08	24.51	27.49	31.99	36.4	44.93
μ_{sw}		17.95	20.02	21.91	24.05	26.57	31.13	35.42	43.83
μ_{bs}		17.95	20.02	21.89	24.02	26.45	30.78	34.81	42.95
μ_c/LBE		1.01	1.04	1.12	1.24	1.36	1.59	1.8	2.22
μ_{sc}/LBE		1.01	1.03	1.1	1.21	1.32	1.53	1.73	2.08
μ_{sw}/LBE		1.01	1.03	1.09	1.18	1.28	1.49	1.68	2.03
μ_{bs}/LBE		1.01	1.03	1.09	1.18	1.27	1.47	1.65	1.99
I_{sc}	%	34.01	25.26	16.91	14.3	10.2	9.39	9.32	11.94
I_{sw}	%	34.01	32.02	24.71	23.67	22.41	16.4	14.98	15.83
I_{bs}	%	34.01	32.02	25.49	24.27	23.94	19.32	18.67	19.25

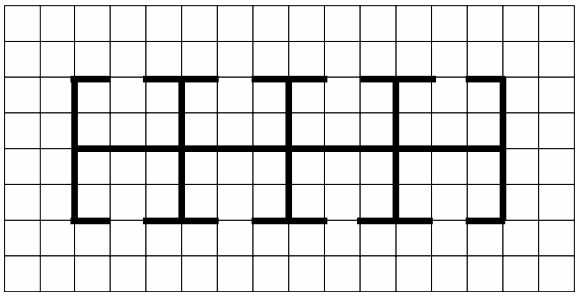

Figure 3: Cyclic Corridor Environment

Table II: Results for Grid Environment

	Unit	Grid 7x7				Grid 11x11				Grid 15x15			
robots		4	8	12	16	9	19	28	38	17	35	52	70
congestion	%	10	20	30	40	10	20	30	40	10	20	30	40
$S_c \equiv S_{sc}$	%	100	100	100	91	100	100	100	73	100	100	98	38
S_{sw}	%	100	100	100	91	100	100	100	74	100	100	98	40
S_{bs}	%	100	100	100	94	100	100	100	85	100	100	98	51
T_c	s	0.05	0.26	0.71	1.26	0.5	3.46	10.28	17.77	2.88	23.51	69.65	103.81
T_{sc}	s	0.06	0.29	0.75	1.29	0.61	3.76	10.71	17.91	3.6	25.49	71.74	104.25
T_{sw}	s	0.07	0.36	0.94	1.61	0.9	5.8	15.54	26.03	7.62	51.23	125.23	179.5
T_{bs}	s	0.13	0.65	1.69	2.89	1.5	9.56	26.25	43.94	11.22	76.72	196.97	283.75
μ_c		7.89	9.57	12.82	20.8	13.55	16.94	23.84	42.49	20.06	25.04	35.66	63.03
μ_{sc}		7.89	9.44	12.12	19.32	13.52	16.51	22.27	37.9	19.86	24.32	33.43	55.92
μ_{sw}		7.89	9.44	12.07	18.98	13.52	16.41	21.89	36.34	19.82	23.91	31.91	51.85
μ_{bs}		7.89	9.44	12.04	18.94	13.52	16.4	21.66	35.99	19.82	23.79	31.41	52.55
μ_c/LBE		1.04	1.13	1.44	2.24	1.03	1.13	1.53	2.67	1.02	1.19	1.62	2.77
μ_{sc}/LBE		1.04	1.11	1.36	2.08	1.03	1.1	1.43	2.39	1.01	1.16	1.52	2.45
μ_{sw}/LBE		1.04	1.11	1.36	2.07	1.03	1.1	1.41	2.28	1.01	1.14	1.45	2.32
μ_{bs}/LBE		1.04	1.11	1.35	2.05	1.03	1.1	1.39	2.26	1.01	1.13	1.43	2.33
I_{sc}	%	0	12.98	17.83	12.56	8.95	22.13	19.08	17.25	45.01	18.15	16.42	17.71
I_{sw}	%	0	12.98	19.01	13.92	8.95	27.71	23.86	23.75	54.46	29.11	27.63	25.09
I_{bs}	%	0	12.98	19.91	15.23	8.95	28.18	26.72	24.9	54.46	31.91	31.29	24.58

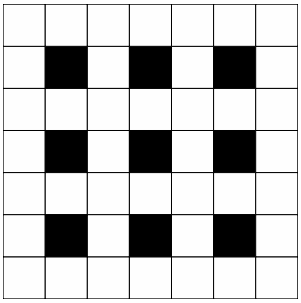

Figure 4: Extensible Grid Environment

of access to the shared path. The resulting string will be $s_{new} = (2,3,2)(2,2,5)(2,5,8)(1,1,2)(1,2,5)(1,5,4)(2,8,9)(2,9,6)$ and after the reordering made by $CompressTrace$ at line 20 the final trace will be $t_{new} = (2,3,2)(2,2,5)(1,1,2)(2,5,8)(1,2,5)(2,8,9)(1,5,4)(2,9,6)$ with execution time $v_{new} = 5$. □

There are three potential reasons why one subsequence $\hat{s} = \sigma_p \dots \sigma_q$ of events which belong to the same robot and one event σ_m, which occurs before first event of $\hat{s}$, should not be swapped, i.e. they are not indipendent. When we talk about swapping we mean moving $\hat{s}$ before $\sigma_{\hat{q}}$, further we need to refer to a valid trace $t \in V(\mathbb{G})$ because the dependance properties between σ_m and $\hat{s}$ depend not only on them but also on events between them. First if there exists $\sigma_i \notin \hat{s}$ such that $\hat{q} < i < q$ and it shares an alphabet with events in $\hat{s}$, then swapping $\sigma_{\hat{q}}$ and $\hat{s}$ leads to a string $s' \notin L(G)$. Second if there exists $\sigma_i \notin \hat{s}$ such that $\hat{q} < i < q$ and it moves in or out of the state which σ_q moves into, then swapping $\sigma_{\hat{q}}$ and $\hat{s}$ leads to collision. Third if there exists $\sigma_i \notin \hat{s}$ such that $\hat{q} < i < q$ and it is the first event after $\sigma_{\hat{q}}$ which belong to a language Σ_j, then if σ_i leaves a state which belongs to the path of $\hat{s}$, then swapping $\sigma_{\hat{q}}$ and $\hat{s}$ leads to collision.

Example 2. Given the valid trace $t = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5\sigma_6\sigma_7\sigma_8 = (2,2,3)(1,8,9)(1,9,6)(1,6,5)(1,5,4)(2,3,6)(1,4,1)(2,6,9)$, let $\hat{s} = \sigma_6\sigma_8 = (2,3,6)(2,6,9)$ be a subsequence of t, we analyze swapping $\hat{s}$ and events $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$. Swapping $\hat{s}$ with σ_1 will results in a string not generated by $L(G)$ because they share the same alphabet (first case). Swapping $\hat{s}$ with σ_2 or with σ_3 will result in an invalid trace because both move in or out the state 9 and the last event of $\hat{s}$ move in the same state 9 (second case). Swapping $\hat{s}$ with σ_4 will result in an invalid trace because σ_4 is the first event between σ_4 and last event of $\hat{s}$ which belong to the language Σ_1 and it move out the state 6 and there is one event of $\hat{s}$, i.e., $(2,3,6)$, which move in the same state 6 (third case). Swapping $\hat{s}$ with σ_5 will result in an valid trace because all the events between σ_5 and last event of $\hat{s}$ which do not belong to $\hat{s}$ (i.e. $(1,5,4)$ and $(1,4,1)$) are not in one of these 3 cases. □

V. Experimental Results

A. Description

In order to test the effectiveness of the presented algorithm we first computed a solution [10] for the MRPP problem in two different environment: Cyclic Corridor and Extensible Grid depicted in Figures 3 and 4. To generate a random problem instance with n robots we selected a random start and goal position from amongst all the possible free location 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 another robot's start position as its goal position. It is also possible for a robot to have its current start position as its goal position, in which case the robot will not have to move other than to get out of the way of other robots. The algorithms were run on a Intel(R) Core(TM) i7-4770 CPU @ 3.40GHz.

For each experiment we calculate a lower bound (LB) for the minimum execution time of the solution in the following manner. We first calculate the shortest path for each individual robot assuming that there are no other robots to get in its way. Then we find the robot with the longest journey and select it's duration as our LB.

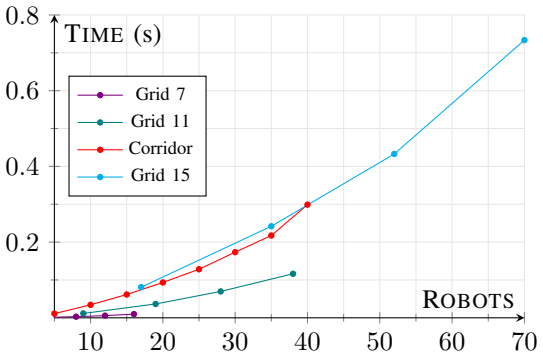

Figure 5: Chart showing average additional time required to improving a trace using Algorithm 1 vs number of robots

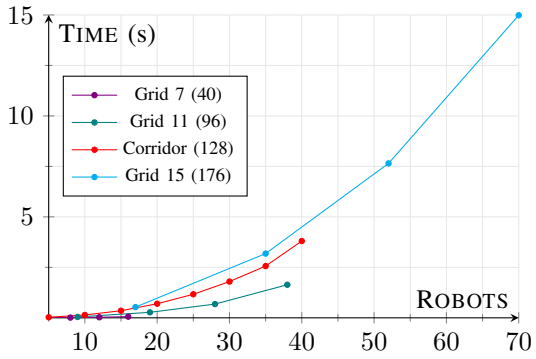

Figure 7: Chart showing average additional time required to compute a trace applying Algorithm 1 vs number of robots

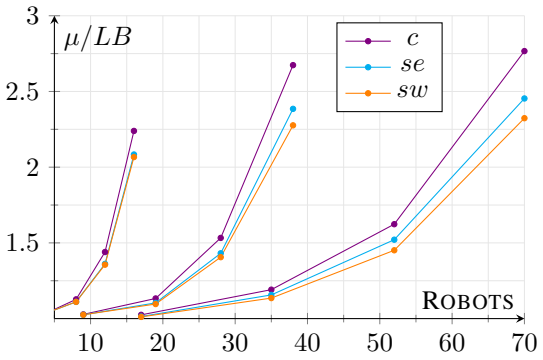

Figure 6: Chart comparing μ/LB between tests c, se and sw in Table II vs number of robots.

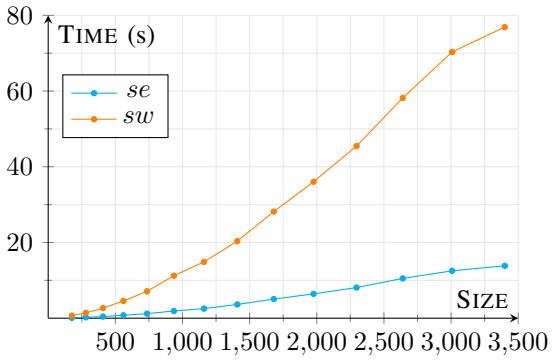

Figure 8: Chart comparing additional time between tests se and sw over c vs size of environment.

In each environment, four types of test were carried out and Tables I and II show their average results over 100 experiments. For purposes of presentation we will refer to these cases by an acronym. First we found a solution to the problem; we refer to this case by acronym "c" which stands for "Compare". Second we applied the Algorithm 1 to the result of test c for $N \in \{1, \ldots, LB\}$; we refer to this case by acronym "se" which stands for "Swap at the End". Third we found a new solution applying Algorithm 1 while computing the path; we refer to this case by acronym "sw" which stands for "Swap While". Forth we considered when at least one of tests se and sw succeeded and we took the best result; we refer to this case by acronym "bs" which stands for "Best Swap".

Table I uses Figure 3 as environment and Table II uses Figure 4 as environment, the grid is also extended to an 11 by 11 and a 15 by 15 grid. In these tables, ROBOTS represent the number of robots being coordinated, CONGESTION represents congestion percentage, i.e., the number of robots coordinated as a percentage of the number of free slots in the grid, $S_i\%$ represents the number of experiments which found a path for all robots, $T_i(s)$ is the average time required to run the expirement, μ_i is the average minimum execution time of the solution, μ_i/LB is the average ratio of the minimum execution time compared to our lower bound, I_i is the improvement obtained applying test i over test c, where "i" denotes the acronym of the test. In Equation 1 we define the improvement obtained with our algorithm for each test $i \in \{se, sw, bs\}$.

$$I_i = \begin{cases} 1 - \frac{\mu_i - LB}{\mu_c - LB} & \mu_c \neq 0 \\ 0 & otherwise \end{cases} \quad (1)$$

B. Analysis of results

We start analyzing tests se which are the most important to evaluate the direct effectiveness of Algorithm 1 introduced in this paper. Comparing results related to tests se we find that Algorithm 1 does not appear to add a significant amount of time to the time required for computing the solution. On the other hand there does appear to be a considerable increase in the quality of the returned trace, moreover it can be seen that as the congestion increases the use of Algorithm 1 becomes more important in absolute terms but not in terms of percentage. When congestion reach values around 30% an improvement between 11% and 19% is reached in both environments. Comparing improvements reached with all different congestion percentages in open corridor (up to 34%) and grid (up to 45%) environments we can say that improvement depends not only on the congestion but also on the structure of the environment. This is not further discussed in this paper however.

We now analyze tests sw which can give an hint on how much Algorithm 1 can be useful if applied at each phase of a generic decoupled approach, in which paths are calculated independently for each robot: intuitively, the improvement that can be obtained by this application of the algorithm depends on the decoupled technique used. Comparing results related to tests sw we find that applying the addition of Algorithm 1 in the corridor environment it appears to cause a noticeable slow down in finding a solution, the time increases around 50% in the corridor and around 100% in the grid. Nevertheless, on the whole, the quality of solutions has improved considerably compared to tests se, between 10% and 50%. Percentage of founded solutions does not appear to change significantly.

We are now interested in understanding which relation exists between solutions founded by tests sw and tests se, so we analyzed all cases in which at least one of this two tests found a solution. We can see that average quality of solutions is not changed significantly but we are able to find an higher number of solutions. The best environment to see this is the Grid 15 by 15, in which we found 13% more solutions (and only 2% more in tests sw). This suggests that when a decoupled approach is applied and when a new path is computed, chances to find a solution depend not only on the path of each robot but also on the temporal sequence in which various states are occupied.

Figure 5 gives a chart comparing the time required to improve a trace using Algorithm 1 vs the number of robots which need to be coordinated in different environments and Figure 7 gives a chart comparing the extra time required to compute a trace applying Algorithm 1 at each step vs the number of robots which need to be coordinated in different evnironments. These charts show different data from Tables I-II. If the latter shows the total time needed to compute each path in all cases (success and fail), here we consider only cases in which tests c succeeded and then calculated in Figure 5 the additional time required by Algorithm 1 to improve the resulting trace and in Figure 7 the additional time required to compute again the trace by applying Algorithm 1 each step. They show that time complexity appears to grow polynomially with respect to the number of robots which must be coordinated which is in line with our expectations, and each curve seems to differ from the other by a constant factor which is proportional to the size of the environment.

Figure 6 shows the degree to which Algorithm 1 improves trace quality as the number of robots increases for the Extensible Grid Environment in the cases in which it is applied only at the end or during each step of a decoupled approach. We can distinguish three groups of curves, each one relates to a different dimension of the grid, respectively 7 by 7, 11 by 11 and 15 by 15. It shows that as the number of robots which need to be coordinated increases the use of Algorithm 1 becomes more important in absolute terms, where at 70 robots the original algorithm has an average μ/LB of 2.77 compared to the improved trace μ/LB of 2.45 and the new trace μ/LB of 2.32.

Figure 8 shows the same timings described for Figures 5 and 7 vs the number of positions in the environment. This data was obtained by running Algorithm 1 on extensible grid environment with dimension between 11 by 11 and 67 by 67. Under all circumstances exactly 20 robots were coordinated and each data point represents the average of 100 executions. The execution time only seems to grow roughly polynomially with the size of the environment for both cases se and sw.

VI. Conclusion

In this paper we proposed a heuristic algorithm to improve solutions to the multi-robot path-planning problem implemented in a DES context [10] using timed discrete event automata [12]. The algorithm was tested on a variety of problems and it was shown that its use in any centralized planning strategy is useful in terms of execution time, reducing it up to 45% in our tests, and number of solutions founded for the multi-robot path-planning problem, up to 13% more. Future works will aim at a smart choosing of the parameter N, planning strategies to avoid cases in which applying the swap strategy a solution is not found and testing of more complicated environments, such as the presence of places in which a vehicle cannot stop or different movements timing.

References

[1] 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, no. 2, pp. 100–107, July 1968.

[2] T. S. Standley, "Finding optimal solutions to cooperative pathfinding problems." in *24th AAAI Conf. Artif. Intell. (AAAI)*, vol. 1, 2010, pp. 173–178.

[3] T. Standley and R. Korf, "Complete algorithms for cooperative pathfinding problems," in *22nd Int. Joint Conf. Artifi. Intell. (IJCAI/AAAI)*, 2011, pp. 668–673.

[4] 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, no. 4, pp. 76–88, 1984.

[5] 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*, Sept 2011, pp. 206–209.

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

[7] 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.

[8] G. Sanchez and J. C. Latombe, "Using a prm planner to compare centralized and decoupled planning for multi-robot systems," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, 2002, pp. 2112–2119 vol.2.

[9] M. Cap, P. Novak, A. Kleiner, and M. Selecky, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 3, pp. 835–849, July 2015.

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

[11] ——, "An application of incremental scheduling to a cluster photolithography tool," in *20th IFAC World Congress, (IFAC WC 2017)*, May 2017, p. Submitted to.

[12] R. Su, J. H. van Schuppen, and J. E. Rooda, "The synthesis of time optmial supervisors by using heaps-of-pieces," *IEEE Transactions on Automatic Control*, vol. 57, pp. 105–118, 2012.