Intra-Logistics with Integrated Automatic Deployment:
Safe and Scalable Fleets in Shared Spaces

H2020-ICT-2016-2017
Grant agreement no: 732737

DELIVERABLE 5.5

Second prototype of integrated reasoning for on-line fleet management

Due date: month 42 (June 2020)
Deliverable type: R
Lead beneficiary: ORU

Dissemination Level: PUBLIC

Main author: Federico Pecora (ORU)

1 Introduction

In this deliverable, we provide a final overview the work performed in ILIAD on multi-robot fleet management and task optimization. The deliverable aims to be self-contained, and subsumes deliverable D5.4 (which concerned the first prototype of integrated reasoning for on-line fleet management). We start with an updated overview of the formal algorithmic framework (Section 2). We then summarize the main theoretical achievements of the past two years, specifically focusing on safety (Section 3) and liveness (Section 4). The integration of the resulting approach with task optimization is then discussed (Section 5). The resulting solution achieves an integrated task allocation, motion planning, coordination and control framework, the implementation of which is discussed in Section 6. Finally, Section 7 provides concluding remarks on how results of our work have been integrated into the ILIAD system as a whole, and on how they have contributed to achieving the overall objectives of WP5 and of the project.

2 Loosely-coupled online coordination

An important challenge in industrial transport automation is to effectively coordinate heterogeneous fleets of robots in dynamic environments while ensuring safety, liveness, and good overall fleet performance. While methods exist for managing fleets of hundreds of robots in static, dedicated environments (e. g., [18]), such methods cease to work if key assumptions are dropped. Specifically, there are six crucial requirements of real-world applications that need to be accounted for: (R1) robots are subject to complex kinodynamic constraints, (R2) goals and missions become known only at run time, (R3) discretizing the environment and/or robot paths is too costly or curtails flexibility too much, (R4) robot priorities may change over time, (R5) robot motions (R6) and communications may be subject to disturbances. Reactive techniques can be used to deal with some of these requirements, however, they lack predictability and may lead to deadlocks. Deliberative methods, where collisions and deadlocks are accounted for in motion planning, suffer from severe computational overhead. In ILIAD[1] we have developed a centralized supervisory coordinator for heterogeneous robotic platforms. The framework was first proposed in [15] and summarized in D5.4. As initially proposed, the framework accounted for requirements (R1–R5). We subsequently extended the framework to ensure safety under communication disturbances (R6). This development is summarized in Section 3, while the complete paper [10] is included as Appendix A.

Despite fulfilling all requirements R1–R6, the approach obtained thus far could not guarantee liveness — that is, there were conditions under which one or more robots in the fleet could not reach their intended targets. A further extension was therefore realized in the last year of ILIAD to overcome this limitation. The resulting methods are summarized in Section 4; a submitted journal paper describing all the details of this last development [11] is included as Appendix B.

Our approach is designed for applications in which robots are loosely-coupled [9], that is, they share the same workspace and are subject to non-cooperative tasks. Furthermore, the approach assumes decoupled motion planning and control (which holds, e. g., when robots are driven by car-like or differential-drive kinodynamics). Precedence constraints are computed online and revised periodically. These regulate access to, and

[1] This work was carried out in collaboration also with industrial partners outside of the ILIAD consortium, namely, Volvo, Scania, and Epiroc/Atlas-Copco. These collaborations were supported by Swedish funding agencies KKS, Vinnova, and SSF.

progress through, pairwise contiguous overlapping configurations of robot paths. As we show below, the algorithm that computes precedences accounts for kinodynamic feasibility, thus ensuring safety of the fleet.

2.1 The coordination problem

We describe our multi-robot system with a set $\mathscr{R} = \{1, \dots, n\}$ of (possibly heterogeneous) robots sharing an environment $\mathscr{W} \subset \mathbb{R}^3$ with obstacles $\mathscr{O} \subset \mathscr{W}$. Each robot i is identified by a tuple $r_i = \langle \mathscr{Q}_i, R_i, f_i, g_i, s_i \rangle$[2], where:

- $\mathscr{Q}_i$ is the robot's configuration space;
- $R_i(q_i)$ is a geometry describing the space occupied by the robot when placed in configuration $q_i \in \mathscr{Q}_i$;
- $f_i(q_i, \dot{q}_i) \leq 0$ is a set of kinematic constraints on the robot's motion;
- $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i^{\text{acc}}, u_i^{\text{dec}}, t)$ is a model of the robot's dynamics, with maximum acceleration/deceleration $u_i^{\text{acc/dec}}$;
- s_i is the robot's status, containing information about its current mission.

We assume R_i to be independent from $(\dot{q}_i, \ddot{q}_i)$. Also, let $\mathscr{Q}_i^{\text{free}} = \{q_i \in \mathscr{Q}_i : R_i(q_i) \cap \mathscr{O} = \emptyset\}$ be the set of obstacle-free configurations of robot i.

When idle, a robot i may be assigned to a non-cooperative, asynchronously posted task which involves moving from its starting configuration $q_i^{\text{s}} \in \mathscr{Q}_i^{\text{free}}$ to a goal configuration $q_i^{\text{g}} \in \mathscr{Q}_i^{\text{free}}$ and stay there. The definition is general and may be easily extended to account for non-cooperative operations (e. g., pick-and-place) or interim configurations to be reached. The connection between task allocation and coordination is further explored in Section 5.

Given a pair of configurations $(q_i^{\text{s}}, q_i^{\text{g}}) \in \mathscr{Q}_i^{\text{free}} \times \mathscr{Q}_i^{\text{free}}$, a path $\boldsymbol{p}_i : [0,1] \to \mathscr{Q}_i^{\text{free}}$ (parametrized using the arc length $\sigma \in [0,1]$) is a sequence of $q_i \in \mathscr{Q}_i^{\text{free}}$ so that $\boldsymbol{p}_i(0) = q_i^{\text{s}}$, $\boldsymbol{p}_i(1) = q_i^{\text{g}}$, satisfying the set of kinematic constraints $f_i(q_i, \dot{q}_i) \leq 0$ (see Figure 1.a). Idle robots are associated with a path of length one corresponding to their current configuration. For each $\boldsymbol{p}_i$, the trajectory planning problem is the problem of synthesizing an executable temporal profile $\sigma_i(t)$ typically considering the robot's kinodynamic constraints [8]. Given two or more robots, the coordination problem is defined as follows:

Problem 1 (Coordination Problem)**.** Given a workspace $\mathscr{W}$ with obstacles $\mathscr{O}$, a fleet of robots $\{r_i\}_{i=1}^n$, and non-cooperative, asynchronously posted tasks, the coordination problem is the problem of synthesizing and revising during time a set of spatio-temporal constraints on robot trajectories $\bigcup_{i \in \mathscr{R}} \{\boldsymbol{p}_i, \sigma_i(t)\}$ so that both of the following two properties are satisfied:

(P1) Safety. Robots never collide:

$$\forall (i, j \neq i) \in \mathscr{R}^2, \forall t \; R_i(\boldsymbol{p}_i(\sigma_i(t))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) = \emptyset.$$

(P2) Liveness. All robots eventually reach their destination:

$$\forall i \in \mathscr{R}, \exists t < \infty \text{ such that } \sigma_i(t) = 1.$$

As we will see, our work has focused mainly on ensuring that properties (P1–P2) hold under as general boundary conditions as possible.

[2] The notation $(\cdot)_i$ is used in the following to indicate that the variable $(\cdot)$ is related to the robot i.

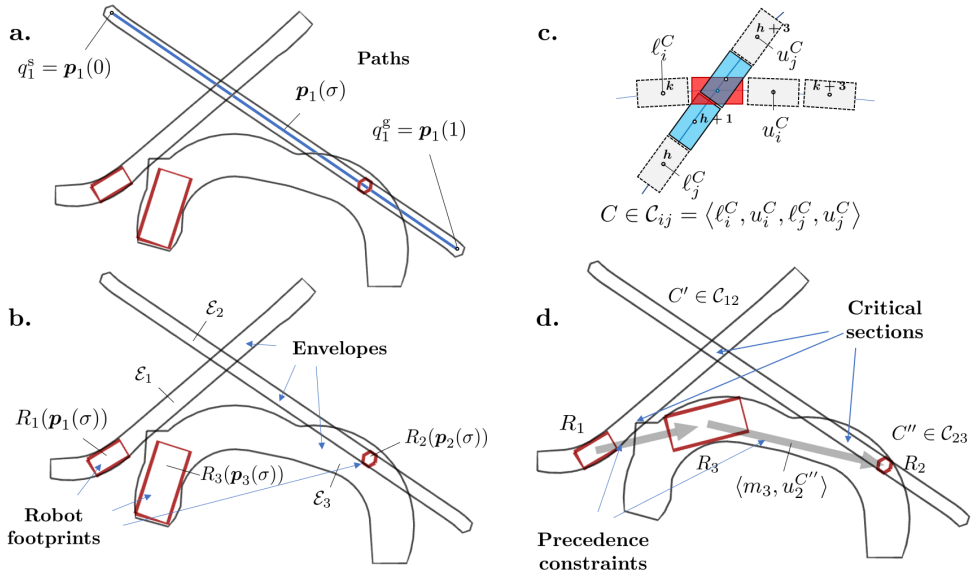


Figure 1: Main concepts of the approach in [15, 10].

2.2 A heuristic, priority-based coordinator

To solve Problem 1 while accounting for constraints that are posted online (either due to contingencies or new posted tasks), our approach relies on a centralized decoupled priority-based supervisory coordinator [15, 10] whose main body, running at each discrete time $k \in \mathbb{N}$, is listed in Algorithm 1. From now on, we use discrete time k to indicate any time $t \in [kT_c, (k+1)T_c)$, where T_c is the period of the coordination loop. We remark also that paths (line 3) may be asynchronously computed by distributed and/or robot-specific planners running in parallel. This speeds up the computation, allows to better explore the heterogeneity of the fleet, and makes it possible to have planning parameters that are private to individual robots (e.g., kinematic constraints, gains, type of search).

Algorithm 1: Coordination at time k.

1 get last received status messages $\bigcup_{i \in \mathcal{R}} s_i(t_i)$;
2 **if** new goals have been posted **then**
3 update the set of paths $\mathcal{P}$ (using appropriate planners);
4 update the set $\mathcal{C}$ of critical sections;
5 revise the set $\mathcal{T}(k)$ of precedence constraints;
6 compute the set of critical points $\bar{\Sigma}(k)$;
7 communicate changed critical points;
8 sleep until control period T_c has elapsed;

The approach uses precedence constraints to regulate access to, and progress through, pairwise overlapping portions of spatial envelopes, called critical sections. We define these concepts as follows.

Definition 1 (see [1])**.** For each path p_i, a spatial envelope $\mathcal{E}_i$ is a set of constraints such that $\bigcup_{\sigma \in [0,1]} R_i(p_i(\sigma)) \subseteq \mathcal{E}_i$.

If the equality holds (which we assume for simplicity from now on), a spatial envelope is the sweep of the robot's geometry along its path (Figure 1.b). A portion of a spatial envelope is identified by the arc length $S \subseteq [0,1]$ it spans, namely, $\mathscr{E}_i^S = \bigcup_{\sigma \in S} R_i(\boldsymbol{p}_i(\sigma))$.

In order to formally characterize blocking and deadlock situations, we need a formal means to specify whether spatial envelopes "avoid" certain sets of configurations:

Definition 2. We say that $\boldsymbol{p}_i$ is Q-avoiding if $Q \subset \mathscr{W} \implies \mathscr{E}_i \cap Q = \emptyset$ and $Q \subset \bigcup_{i \in \mathscr{R}} \mathscr{Q}_i \implies \mathscr{E}_i \cap \bigcup_{q_i \in Q} R_i(q_i) = \emptyset$.

Note that each path $\boldsymbol{p}_i$ is $\mathscr{O}$-avoiding by definition.

Definition 3. A critical section C is a tuple $\langle \ell_i^C, u_i^C, \ell_j^C, u_j^C \rangle$ of continuous intervals of the arc lengths σ_i and σ_j such that for every $\sigma_i \in (\ell_i^C, u_i^C)$, there exists $\sigma_j \in (\ell_j^C, u_j^C)$ such that $R_i(\boldsymbol{p}_i(\sigma_i)) \cap R_j(\boldsymbol{p}_j(\sigma_j)) \neq \emptyset$, and vice versa.

In other words, ℓ_i^C is the highest value of σ_i before robot i enters C, and u_i^C is the lowest value of σ_i after robot i exits C (analogously for j) — see Figure 1.c for examples. Let $\mathscr{C}_{ij}$ be the set of all critical sections pertaining to the two robots i and j. We say that $C \in \mathscr{C}_{ij}$ is active while $\sigma_i(t) < u_i^C \wedge \sigma_j(t) < u_j^C$, that is, when neither robot has exited C. Given the set of robot paths $\mathscr{P} = \bigcup_{i \in \mathscr{R}} \boldsymbol{p}_i$, let $\mathscr{C} \subseteq \bigcup_{(i,j>i) \in \mathscr{R}^2} \mathscr{C}_{ij}$ be the set of all active critical sections. Henceforth, let $\mathscr{P}(\mathrm{k})$ and $\mathscr{C}(\mathrm{k})$ be the values of these sets after all paths have been computed, that is, after executing line 4 of Algorithm 1.

Definition 4. A precedence constraint $\langle m_i, u_j^C \rangle$ is a constraint on the temporal evolution of $\sigma_i(t)$ such that $\ell_i^C \leq m_i < u_i^C$ and $\sigma_j(t) < u_j^C \implies \sigma_i(t) \leq m_i$.

Thus, imposing a precedence constraint $\langle m_i, u_j^C \rangle$ means imposing that robot i does not navigate beyond $\boldsymbol{p}_i(m_i)$ along its path until robot j has passed the critical section C (see Figure 1.d). In other words, a precedence constraint defines which robot should yield, where, and until when. Let $\mathscr{T}(\mathrm{k})$ be the set of precedence constraints regulating access to the set of critical section $\mathscr{C}(\mathrm{k})$.

Definition 5. If $\forall C \in \mathscr{C}(\mathrm{k})$, either $\langle m_i, u_j^C \rangle \in \mathscr{T}(\mathrm{k})$ or $\langle m_j, u_i^C \rangle \in \mathscr{T}(\mathrm{k})$, then $\mathscr{T}(\mathrm{k})$ is a complete ordering of robots through $\mathscr{C}(\mathrm{k})$.

Note that if $\mathscr{T}(\mathrm{k})$ is a complete ordering, then it is guaranteed that (P1) holds. This is because, given the set of paths $\mathscr{P}(\mathrm{k})$, a complete ordering $\mathscr{T}(\mathrm{k})$ defines the selected homotopic class of collision-free trajectories [5, 6]. In particular, for each $C \in \mathscr{C}(\mathrm{k})$, the precedence constraint $\langle m_i, u_j^C \rangle \in \mathscr{T}(\mathrm{k})$ is computed as

$$m_i(\mathrm{k}) = \begin{cases} \max\left\{\ell_i^C, r_{ij}(\mathrm{k})\right\} & \text{if } \sigma_j \leq u_j^C \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

$$r_{ij}(\mathrm{k}) = \sup_{\sigma \in [\sigma_i(t_i), u_i^C]} \left\{ \mathscr{E}_i^{[\sigma_i(t_i), \sigma]} \cap \mathscr{E}_j^{[\sigma_j(t_j), u_j^C]} = \emptyset \right\},$$

where $\sigma_i(t_i)$ and $\sigma_j(t_j)$ are the last known positions of robot i and robot j, received by the coordinator at time t_i, $t_j \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$, respectively. Note that m_i is updated at each control period (line 5 in Algorithm 1), allowing robots to "follow each other" through critical sections.

Let $\Psi_i = \{m_i \mid \exists j : \langle m_i, u_j^C \rangle \in \mathscr{T}(\mathrm{k})\}$ be the set of all the arc lengths at which robot i may be required to yield.

Definition 6. The critical point $\bar{\sigma}_i(\mathrm{k})$ of robot i at discrete time k is the value of σ corresponding to the last reachable configuration along $\boldsymbol{p}_i$ which adheres to the set of constraints $\mathscr{T}(\mathrm{k})$, i.e.,

$$\bar{\sigma}_i(\mathrm{k}) = \begin{cases} \underset{m_i \in \Psi_i(t)}{\arg\min}\, m_i & \text{if } \Psi_i \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

Hence, Problem 1 is equivalent to that of finding an appropriate $\mathscr{P}(\mathrm{k})$, $\mathscr{C}(\mathrm{k})$, $\mathscr{T}(\mathrm{k})$ and $\bar{\Sigma}(\mathrm{k}) = \bigcup_{i \in \mathscr{R}} \bar{\sigma}_i(\mathrm{k})$, and revising these sets appropriately at each control period.

A key feature of our method is that the selected collision-free homotopic class of trajectories may change online. In other words, precedence orders may be dynamically updated according to any user-defined heuristic-based ordering function $\mathrm{h}(t)$ [15] (let $i \prec_{\mathrm{h}(t)} j$ indicate that i yields for j according to $\mathrm{h}(t)$ at a given $C \in \mathscr{C}_{ij}$), while guaranteeing that safety is preserved. This is done in Algorithm 2, which illustrates the functioning of line 5 of Algorithm 1. The algorithm is used to filter changes of precedence orders that may results in a collision. For this purpose, we define the lookahead Δ_i^{stop} as the interval of time such that a command to yield sent by the coordinator at time t will make robot i stop at most at time $t + \Delta_i^{\mathrm{stop}}$. A conservative estimate of this value allows to ensure safety in the presence of bounded uncertainties in the robot's dynamics [15], and in the communication network [10]. Then, at discrete time $\mathrm{k} \geq 1$ a constraint $\langle m_i, u_j^C \rangle \in \mathscr{T}(\mathrm{k}-1)$ can be replaced with $\langle m_j, u_i^C \rangle \in \mathscr{T}(\mathrm{k})$ (reversed) only if $\sigma_j(t_j + \Delta_j^{\mathrm{stop}}) \leq \ell_j^C$, $t_j \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$ (i. e., the new yielding robot has not already entered the critical section and can stop before entering it if asked to). This feasibility check is implemented via a conservative forward propagation of the two robots' dynamics (see the canStop function of [10] for details).

Algorithm 2: The revise function at time k (implements line 5 of Alg. 1).

Input: $\mathscr{P}$ current set of paths; $\mathscr{C}$ (possibly empty) set of pairwise critical sections; $\mathscr{T}(\mathrm{k}-1)$ (possibly empty) previous set of precedence constraints; $\bar{\Sigma}(\mathrm{k}-1)$ previous set of critical points; $s_i(t_i)$ last received robot status message, including $\sigma_i(t_i)$; h heuristic-based ordering function.

Output: $\mathscr{T}^{\mathrm{rev}}$ set of revised precedence constraints.

```
1  T_rev ← ∅;
2  for C_ij ∈ C, C ∈ C_ij do
3      if σ_i(t_i) < u_i^C ∧ σ_j(t_j) < u_j^C then
4          if σ_i(t_i + Δ_i^stop) ≤ ℓ_i^C ∧ σ_j(t_j + Δ_j^stop) ≤ ℓ_j^C then
5              (h, k) ← get ordering according to h;
6          else if σ_i(t_i + Δ_i^stop) > ℓ_i^C ∧ σ_j(t_j + Δ_j^stop) ≤ ℓ_j^C then (h, k) ← (j, i);
7          else if σ_i(t_i + Δ_i^stop) ≤ ℓ_i^C ∧ σ_j(t_j + Δ_j^stop) > ℓ_j^C then (h, k) ← (i, j);
8          else (h, k) ← get previous ordering from T;
9      ⟨m_h, u_k^C⟩ ← compute as in (1);
10     T_rev ← T_rev ∪ {⟨m_h, u_k^C⟩};
11 return T^rev;
```

3 Safety

Our approach builds on a centralized coordinator which communicates via duplex point-to-point communication (through a dedicated wireless network, subject to bounded delays and/or message loss) with each robot in the fleet. We require each robot to send

an update on its status s_i sampled within its control period T_i (clock-driven system) every T_i seconds (robots may have different control periods). The status message contains the tuple $(q_i(t_i), \dot{q}_i(t_i), \ddot{q}_i(t_i))$, as well as the last critical point $\bar{\sigma}_i$ received by the robot. Also, we assume the coordinator receives at least one update from each agent every T_c seconds, and that $T_c \geq \max_{i\in 1,\ldots,n} T_i$. Robots are not required to be synchronized on a common clock (so communication may be asynchronous).

In order to characterize the ability of our approach to guarantee safety, we must first qualify the conditions under which we assume the fleet to be operating:

(A1) Paths do not start or end in active critical sections.

(A2) Robots are idle at time k = 0, placed in a safe starting configuration. Also, robots are not in motion when idle.

(A3) Robots always stay within their envelope (that is $q_i(t) \in \bigcup_{\sigma\in[0,1]} \boldsymbol{p}_i(\sigma)$).

(A4) Robots do not back up along their paths.

(A5) Each lookahead Δ_i^{stop} provides a conservative estimate of the time required by robot i to yield if required to. This entails that: (i) $\Delta_i^{\text{stop}} \geq T_c$, as communication occurs at the end of the coordination period, while sampling is at the beginning (lines 1 and 7 in Algorithm 1); (ii) $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i^{\text{acc}}, u_i^{\text{dec}}, t)$ is a conservative model of robot i's dynamic; (iii) a conservative model of the communication network is known, e.g., finite upper bounds of transmission delay and the packet loss probability $(\tau_{\max}^{\text{ch}}, \eta)$.

Also, as in [2], we assume

(A6) Prohibitive disturbances, i.e., uncontrollable events requiring human intervention in order to recover from them[3], not to happen.

Note that (A1) and (A6) are standard assumptions in the literature, since they ensure that a solution of the coordination problem exists.

Algorithm 1 is by construction robust to some communication failures, namely, if the failure is such that all the critical points in the most last $\bar{\Sigma}$(k) before the failure are either successfully delivered or lost. In order to preserve safety also in case of asymmetric disturbances (delays, or some messages are lost and not others), we have designed in [10] a UDP-like protocol which ensures safety by relating number of re-transmissions to the properties of the communication channel. We have shown that under assumptions under assumptions (A1–A6), there exists a relation between communication infrastructure requirements, number of robots, controller parameters, and the probability $\bar{p}_u$ of a precedence constraint being violated, which in turn may lead to violating (P1). This relation is characterized formally in [10]. We show, in particular, that if $\eta = 0$ (no messages are lost) and the communication delay is bounded, the use of Algorithms 1 and 2, combined with a UDP-like protocol, is sufficient for ensuring that the probability of constraint violation is zero:

Corollary A1. For any robot i in the fleet, any realization of $\sigma_i(t)$ that adheres to $\bar{\Sigma}(t)$ is safe (guarantees P1). This includes unforeseen stops or changes in velocity due to low-level control and/or safety mechanisms.

[3] e.g., an unpredictable obstruction along the path making the current goal unreachable (there does not exist an executable path leading to it), a failure of one or more robots, of the overall communication network or the coordinator, or a malicious dynamic obstacle.

If $\eta \geq 0$, we have shown that the fleet controller can be synthesized to guarantee any user-defined $\bar{p}_u$. Similarly, the relationship between channel model and controller parameters can be used to dimension the channel, thereby providing a means to co-design the fleet controller, the robot controllers, and the communication channel. Please see [10] or Appendix A for details.

Note that this sufficient condition above holds no matter which heuristic h is used, hence the choice of heuristic does not affect safety. Conversely, as we illustrate below, the choice of h can affect liveness.

4 Liveness

We now turn our attention to ensuring liveness (P2). Towards this aim, we have proceeded in two steps. First, we have formally defined the conditions necessary to ensure liveness by design, that is, the absence of blocking, deadlocks and livelocks. This was achieved by generalizing the notion of well-formed infrastructure introduced in [3] to heterogeneous fleets, and introducing new formal properties of the heuristics used to decide precedences among robots.

Second, we have devised several methods for actively imposing these conditions for systems in which they may otherwise occur. Each of the proposed methods is suitable under a different set of boundary conditions. Specifically, we propose (a) two variants of an online feasibility check that can be used to discard goals and paths that lead to blocking; and (b) three extensions of the original algorithm that prevent and/or recover from deadlocks.

All findings are reported in [11], which is included here as Appendix B. Each method has been validated both formally and empirically, and the trade-off between computational complexity and boundary conditions under which the method is applicable are discussed in detail in Appendix B. In the remainder of this section, we summarize the principal findings of our work.

4.1 Factors affecting liveness

As reported in the literature, several factors may prevent robots from reaching their destinations:

Blocking: a robot should stop its mission for an unbounded time because another robot has parked along its path [3];

Deadlocks: there is a subset of robots such that each waits for another one to proceed along its path [6];

Livelocks: similar to deadlocks, but where robot configurations constantly change, none progressing [14].

However, Algorithms 1 and 2 fail to consider these factors, and hence liveness (P2) is not guaranteed. To overcome this, we alter lines 2–5 of Algorithm 1. In order to ensure that (P2) can be verified formally, we map this to properties of the spatial and the temporal components of the problem, i.e., the sets of trajectory envelopes $\mathscr{E}(\mathrm{k}) = \bigcup_{j \in \mathscr{R}} \mathscr{E}_j(\mathrm{k})$ and of precedence constraints $\mathscr{T}(\mathrm{k})$ at time $t \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$. Specifically, we require that the set of trajectory envelopes is at all times admissible:

Definition 7 (Admissibility of spatial envelopes)**.** $\mathscr{E}(\mathrm{k})$ is admissible iff there exists $\mathscr{T}(\mathrm{k})$ s.t. both (P1) and (P2) hold (see Figure 2).

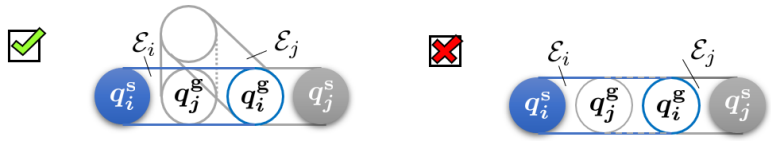


Figure 2: Admissible (left) and not admissible (right) envelopes.

Examples of admissible and non-admissible enveloped are shown in Figure 2. Note that while $\mathscr{E}(\mathrm{k})$ does not change, any temporal evolution of σ_i, $i \in \mathscr{R}$ satisfying $\mathscr{T}(\mathrm{k})$ and computed according to (1) belongs to the same homotopic class, hence it maintains the same properties. Therefore, starting from $\mathscr{E}(0)$, which is admissible thanks to (A1)–(A3), our aim is to define how to ensure that $\mathscr{E}(\mathrm{k})$ remains admissible for all k.

Let $\mathbb{G}_i \subseteq \mathcal{Q}_i^{\text{free}}$ be the set of all the possible end-points of robot i, $\mathbb{G} = \bigcup_{i\in\mathscr{R}} \mathbb{G}_i$, and $\mathbb{G}^{j\neq i}(q_i^g) = \{q_j^g \in \mathbb{G}_j : R_i(q_i^g) \cap R_j(q_j^g) = \emptyset\}$. The absence of blocking can be ensured by design, requiring the environment $\mathscr{W}$ and the set $\mathbb{G}$ to form a well-formed infrastructure, which we define[4] as follows:

Definition 8 (Well-formed infrastructure)**.** The pair $(\mathscr{W}, \mathbb{G})$ is a well-formed infrastructure if $\forall\, i \in \mathscr{R}$, $\forall (q_{i_j}, q_{i_h}) \in \mathbb{G}_i \times \mathbb{G}_i$ there exists a $\mathbb{G}^{j\neq i}(q_{i_h})$-avoiding path from q_{i_j} to q_{i_h} that lies in $\mathcal{Q}_i^{\text{free}}$ and adheres to $f_i(q_i, \dot{q}_i) \leq 0$.

As we summarize below, a well-formed infrastructure can be exploited to guarantee that new goals are rejected if they can possibly lead to blocking.

4.2 Guide to results

All methods for ensuring liveness are achieved by altering lines 2–3 and 5 of Algorithm 1. Table 1 summarizes these methods, with references to the relevant theorems and sub-routines detailed in Appendix B.

	Blocking (lines 2–3, 5)	**Deadlocks (line 5)**	**Livelocks**
Avoidance	Generalized well-formed infrastructure (**Definition 8**) + **Theorem B2**	Totally-ordering heuristics + synchronous goal posting, or FCFS heuristic (**Theorem B7**)	No re-planning and no backtracking along paths (**Theorem B11**)
Global prevention	Online goal checking + rejection/delay (**Theorem B3 or B4**)	Global re-ordering (**Algorithm B3**) + **Theorem B8**	Re-plan only if no backtracking and the new path is shorter (**Theorem B12**)
Local prevention & repair	Re-planning (**Algorithms B6–B7**) + **Theorem B9**	Partial re-ordering (**Algorithm B5**) + re-planning (**Algorithms B6–B7**)	—

Table 1: A guide to results in Appendix B.

In particular, we can we can state which boundary conditions need to be upheld in order to completely avoid the possibility of blocking, deadlock, or livelock:

- In in well-formed infrastructures, the absence of blocking can be guaranteed by imposing that new goals are reached via $\mathbb{G}^{j\neq i}$-avoiding paths (see Theorem B2).

[4] The definition given in [3] is here extended to tackle kinematic constraints and generic robot footprints, while considering, for simplicity, $\mathscr{E}_i = \bigcup_{\sigma\in[0,1]} R_i(\boldsymbol{p}_i(\sigma))$.

- In order to avoid deadlocks, conditions need to be imposed on the heuristic used for regulating access to critical sections and on the way goals are posted — unless, that is, a First-Come-First-Served (FCFS) heuristic is imposed, in which case goals can be posted asynchronously (see Theorem B7).
- Livelocks can be avoided if robots are not allowed to backtrack along their trajectories and plans are not changed once planned (see Theorem B11).

When such conditions cannot be guaranteed, we resort to prevention strategies, that is, algorithms that actively prevent blocking, deadlock and livelock from happening. Prevention strategies may require knowledge of, and affect, the motions of all robots in the fleet. In this case, we can state the following:

- Blocking can be prevented by ensuring that there exists a feasible ordering of robots through critical sections that contain goals (see Theorems B3 and B4); or that posted goals are reached via paths that do not interfere with current goals or the envelopes traversed to reach them (see Theorem B4).
- A global re-ordering method (see Algorithm B3) ensures that $\mathscr{T}(t)$ is live for all t by incremental loop checking and appropriate precedence constraint re-ordering, at the cost of exponential computational overhead (see Theorem B8).
- When re-planning is necessary, livelocks can be avoided if we guarantee that re-planned paths are shorter than the original paths (see Theorem B12).

Finally, if we cannot assume knowledge of the motions of all robots in the fleet, then local repair actions may be warranted to re-establish admissibility. In particular, we have found that:

- Re-planning can be done (see Algorithms B6 and B7) in a safe way and without provoking blocking situations (see Theorem B9).
- A best-effort strategy to avoid deadlocks via a combination of re-ordering (of a subset of the constraints in $\mathscr{T}$, see Algorithm B5) and re-planning (Algorithms B6 and B7) is also shown. While not incurring exponential computation, this strategy cannot guarantee liveness.

All of the prevention/repair strategies listed above preserve the key features of the online setting, that is, goals become known only at run-time, and priorities can be changed online. An empirical study of the various strategies was performed (see Appendix B) using artificial and realistic environments. Simulations with up to 40 robots confirm that complexity is exponential with the size of the largest subset of interfering envelopes. Using the theoretical results summarized above, we provides practical examples of how to co-design the control period, the path planning solution, and the mission assignment mechanism to ensure safety. The two local algorithms for deadlock prevention and recovery (based on re-ordering precedences and re-planning paths, respectively) are also tested. The analysis shows that these can effectively be used to reduce the computational overhead, and proved to be most effective in realistic situations (as opposed to artificially difficult benchmarks).

5 Integrating task optimization

The problem of deciding the goals $\mathbb{G}$ for the fleet of robots $\mathscr{R} = \{r_1, \ldots, r_n\}$ can be seen as a Multi-Robot Task Assignment (MRTA) problem [7]. An ample range of solutions, ranging from constrained optimization to particle swarm optimization methods, have

been studied in the literature — see Chapter 2 in Appendix C for a brief overview of relevant literature. In principle, the loosely-coupled approach to integrated motion planning, coordination and control presented in the previous sections is well suited to any existing MRTA method. This is because goals (or the paths to reach them) can be posted online, imposing no particular restriction on the method used to compute such allocations. Appendix C proposes several methods for computing task assignments based on heuristic search.

5.1 The Multi-Robot Task Allocation problem

The Multi-Robot Task Allocation (MRTA) problem is stated as follows. A possibly different number of robots and tasks are considered, and the robots should be allocated to tasks in order to optimize the overall performance of the fleet. Specifically, let $\{\pi_1,\dots,\pi_m\}$ be the set of tasks posted at a given time (with either $m \leq n$ or $m \geq n$). Assigning task π_j to robot r_i involves the robot moving from its starting configuration $q_i \in \mathcal{Q}^{\text{free}}$ to the sequence $\{\mathrm{x}_j^{\mathrm{s}},\dots,\mathrm{x}_j^{\mathrm{f}}\}$ of poses while performing some non-cooperative operations in each of them (e.g., picking or placing an object). We assume the nominal time to complete each operation to be known. We assume to have a roadmap, defined as follows. Let $\boldsymbol{p}_{ijs}: [0,1] \to \mathcal{Q}_i^{\text{free}}$ be a path associated to robot i and task j (as before, parameterized using arch length σ) such that: (a) $\boldsymbol{p}_{ijs}(0) = q_i$; (b) for each $\mathrm{x} \in \{\mathrm{x}_j^{\mathrm{s}},\dots,\mathrm{x}_j^{\mathrm{f}}\}$ there exists a monotone sequence of σ such that $R_i(\boldsymbol{p}_{ijs}(\sigma)) = \mathrm{x}$; and (c) $R_i(\boldsymbol{p}_{ijs}(1)) = \mathrm{x}_j^{\mathrm{f}}$, $f_i(\boldsymbol{p}_{ijs}(\sigma), \boldsymbol{p}_{ijs}(\dot{\sigma})) \leq 0$. Let $\mathscr{P}_{ij} = \bigcup_{s=1}^{p_{ij}} \boldsymbol{p}_{ijs}$ be the (possibly empty) set of p_{ij} alternative paths to reach the poses in task π_j for a robot r_i — note that p_{ij} may differ for different pairs (i,j). Also, let $\mathscr{P}^{\text{all}} = \bigcup_{i=1}^{n}\bigcup_{j=1}^{m} \mathscr{P}_{ij}$ be the set of all paths at each instance of the MRTA problem. We further constrain the problem so that each given task is assigned to only one robot (so cooperative tasks are not considered), and robots can execute only one task at a time. The task allocation problem can be formulated as an Optimal Assignment Problem $\mathrm{OAP}(n,m,\alpha)$ for n robots, m tasks[5], and objective function weight α, where the task is to find an assignment $z \in \{0,1\}^{\bar{p}}$, $\bar{p} = \sum_{(i,j)\in\{1,\dots,n\}\times\{1,\dots,m\}} p_{ij}$, of robots to tasks:

$$\underset{z}{\text{minimize}} \quad \alpha\mathscr{B} + (1-\alpha)\mathscr{F} \tag{3a}$$

$$\text{subject to} \quad z_{ijs} \in \{0,1\}, \tag{3b}$$

$$\sum_{i=1}^{n}\sum_{s=1}^{p_{ij}} z_{ijs} = 1 \quad \forall j \in \{1,\dots,m\}, \tag{3c}$$

$$\sum_{j=1}^{m}\sum_{s=1}^{p_{ij}} z_{ijs} = 1 \quad \forall i \in \{1,\dots,n\}, \tag{3d}$$

$$z_{ijs} = 0 \quad \text{if } (i,j,s) \in \mathscr{S} \tag{3e}$$

where:

- Decision variable $z_{ijs} = 1$ iff robot r_i is assigned to execute task π_j via path $\boldsymbol{p}_{ijs}$.
- $\mathscr{B} = \sum_{k=1}^{\eta} \beta_k \sum_{i=1}^{n}\sum_{j=1}^{m} \mathscr{B}_{kij}\left(\mathscr{P}^{\text{all}}, \cup_s z_{ijs}(t), r_i(t)\right)$ is the interference-free cost. Each term of the linear combination accounts for individual robot information, hence the dependency of this function on the state and decision variables related to one robot (see Appendix C, Chapters 4, for details).

[5] As is common in the literature [13], an instance of the MRTA with n robots and m tasks can be formulated as a square problem (which typically makes the problem more convenient to work with) by adding dummy robots or tasks whenever the number of idle robots differs from the number of tasks.

- $\mathscr{F}\left(\mathscr{P}^{\text{all}}, \cup_{i,j,s} z_{ijs}(t), \cup_i r_i(t)\right)$ is the interference cost. This considers the additional cost associated with the interference between robots in the shared space, hence the dependency of this function on the states and decision variables related to all robots (see Appendix C, Chapters 4, for details).
- The user-defined parameter α is used to weigh the two components $\mathscr{B}$ and $\mathscr{F}$ of objective function 3a.
- The set $\mathscr{S} \subseteq \{1,\ldots,n\} \times \{1,\ldots,m\} \times \{1,\ldots,p_{ij}\}$ is used to model both capability and capacity constraints: $(i,j,s) \in \mathscr{S}$ either if robot r_i cannot perform task π_j (e.g., if π_j requires a picking operation and robot r_i is not equipped with picking tools) or if r_i is still performing a task assigned at a previous time.

The complexity of the OAP stated above depends on the form of the function $\mathscr{F}$, the part of the objective function that considers how robot motions interfere with each other. In general, there are several ways to estimate $\mathscr{F}$. When the evaluation of the interference is computable in polynomial-time, the problem is known as the Multiple-choice assignment problem with polynomial-time computable function [13]. Since the penalization function cannot be computed cheaply, the problem is NP-hard. If $\mathscr{F}$ can be computed in linear time, the resulting problem class is tractable, while it remains NP-hard if $\mathscr{F}$ is a general convex function.

In the real world of robots moving in a shared environment, interference between robots depends on the trajectories robots follow to reach their goals. Hence, computing interference cost involves computing the possible spatio-temporal intersections between robots as they jointly navigate the shared space. Even if the possible paths of robots to their possible goal assignments are known in advance (e.g., if a roadmap were given), computing interference cost would therefore still require a combination of geometrical and temporal inference, the computational overhead of which is at least polynomial.

A concrete implementation of functions $\mathscr{B}$ and $\mathscr{F}$ is discussed in Appendix C (Chapters 4 and 5, respectively). In the following, we focus on general mechanisms for computing optimized task allocations, given these components of the objective function.

5.2 Systematic and local search for MRTA

As shown in Appendix C, the definition of a function $\mathscr{F}$ that can compute the interference cost for a global task assignment and path selection for all robots in the fleet would be impractical (a look-up table representation of the function would occupy $O(n!)$ space in the worst case). However, a more practical interference cost function $\mathscr{F}$ that considers pairs of robot allocations can be defined relatively easily. This can be used to define a systematic algorithm for computing a solution to the OAP(n,m,α), shown in Algorithm 3. The algorithm decomposes the problem of finding a low-cost assignment in two steps. First, an optimal solution to the relaxed problem OAP$(n,m,\alpha=1)$ is found[6] (line 7). This solution is optimal with respect to the individual cost function $\mathscr{B}$, therefore, it ignores the interference cost. The cost of the obtained assignment with respect to the pair-wise interference cost $\mathscr{F}$ is then calculated by evaluating each pair-wise interference, for every possible choice of paths for that pair (line 15). If the resulting cost is less than the previously-known best cost c^*, the assignment is stored as the current best (line 17). Note that the optimal solution to the relaxed problem may not be optimal with respect to the desired combination of $\mathscr{B}$ and $\mathscr{F}$ (so, the given α). In order to further search the solution space, the algorithm attempts to improve on the found solution by excluding it from

[6] Using an off-the-shelf optimization toolkit (Google OR Tools).

Algorithm 3: Systematic Task Assignment Algorithm

Input: $\{s_i\}_{i=1}^{n}$ status of all n robots, with each s_i containing robot r_i's current status; $\mathscr{P}$ roadmap; $\mathscr{G}$ current set of m tasks to be assigned; $\mathscr{S}$ capabilities and previous assignments.

Output: An optimal assignment z^* and its cost c^*.

1 $n^d, m^d \leftarrow$ get number of dummy robots and/or tasks from $\mathscr{S}$ and $\mathscr{G}$;
2 $n^a = n + n^d$;
3 $m^a = m + m^d$;
4 OAP$(n^a, m^a, 1) \leftarrow$ build the problem defined in (3a) to (3e) with $n = n^a$, $m = m^a$ and $\alpha = 1$;
5 $z^* \leftarrow \emptyset$, $c^* \leftarrow \infty$;
6 **while** true **do**
7 $z^- \leftarrow$ find an optimal solution to OAP$(n^a, m^a, 1)$;
8 **if** $z^- = \emptyset$ **then break**;
9 $w_B \leftarrow$ evaluate$(\mathscr{B}, z^-)$;
10 $c \leftarrow 0$;
11 $w_F \leftarrow 0$;
12 **forall** $i \in \{1, \ldots, n^a\}$ **do**
13 **forall** $j \in \{1, \ldots, m^a\}$ **do**
14 **forall** $s \in \{1, \ldots, p_{ij}\}$ **do**
15 $w_F \leftarrow w_F +$ evaluate$(\mathscr{F}, i, j, s, z^-)$;
16 $c \leftarrow \alpha w_B + (1-\alpha) w_F$;
17 **if** $c \leq c^*$ **then**
18 $z^* \leftarrow z^-$;
19 $c^* \leftarrow c$;
20 Add to OAP$(n^a, m^a, 1)$ a constraint that excludes the current solution z^-;
21 Add to OAP$(n^a, m^a, 1)$ a max-cost constraint with value $c^* - \epsilon$;
22 **return** (z^*, c^*);

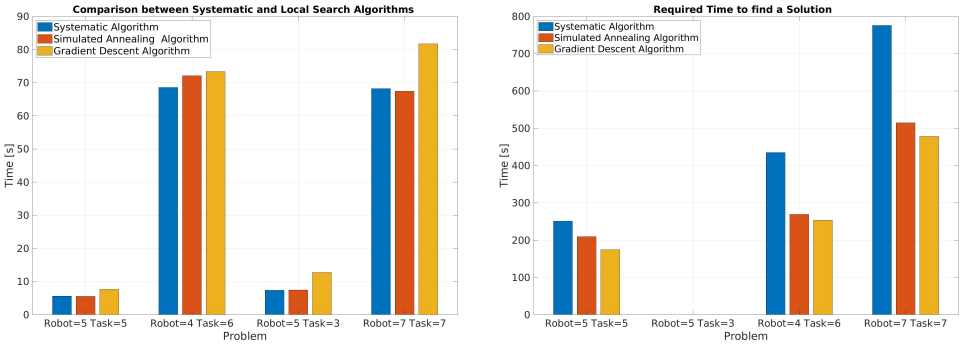
the search space (line 20) and imposing that the cost of further solutions is less than the current best cost (line 21). Once no further improvement is possible (line 8) the iterative improvement loop is interrupted and a the optimal solution is returned (line 22).

Systematic search is but one way to solve the OAP defined above. As is often the case in optimization, local search strategies are worth investigating, as they often achieve better performance at the cost of solution quality. Appendix C outlines two such solutions: one based on Simulated Annealing (SA), and one based on Gradient Descent (GD). Both algorithms exploit the same principle of the above systematic algorithm of computing the cost of an assignment by means of sums of pair-wise evaluations of $\mathscr{F}$. An empirical evaluation of the three algorithms was conducted in several artificial and realistic settings (see Appendix C). The evaluation confirms that the systematic algorithm achieves better solution quality than the local search strategies, at the expense of computation time (see, e. g., Figure 3, which shows the results in one of the scenarios considered). This finding is consistent as the size of the fleet is increased and across scenarios.

6 Software framework

The formal and algorithmic tools introduced above are combined into a framework for integrated motion planning, coordination and control with the following features:

- the framework ensures provable safe and live coordination of heterogeneous robotic platforms subject to kinodynamic constraints with communication disturbances;



(a) Time to completion of missions

(b) Computation time

Figure 3: Comparison between systematic and local search algorithms, with varying number of robots and tasks. Computation time in the case of 5 robots and 3 tasks was in the order of a few milliseconds.

- the framework is general with respect to motion planners and controllers;
- goals can be posted asynchronously to robots when they become known;
- precedences among robots (which may be space- or time- dependent) can be decided according to any user-defined ordering heuristic;
- safety and liveness are guaranteed under bounded spatial and temporal deviations from nominal trajectories.

The framework is implemented in Java and available as open source [16] under the GPLv3 license. At the time of writing, there have been 55 releases since first publication of the framework on Github. We provide below an overview of the major changes in the framework's functionality over the past two years (since it was first described in Deliverable D5.4).

6.1 Minimal usage example

We provide here a somewhat minimal example which exemplifies the most common usage of the framework. This is implemented in class `MinimalExample` in package `se.oru.coordination.coordination_oru.tests`. The example a set of robots with given robotIDs, each with a randomly-generated polygonal geometry. The robots are placed in random initial poses, and assigned to reach a random goal. We omit the implementations of methods `makeRandomFootprint` and `makeRandomStartGoalPair`, which are used to compute the initial pose and goal. For each robot, a path is computed between initial pose and goal. This path is used to define two `Missions`, one to reach the goal from the initial pose, and one to come back. Each mission is enqueued in a global queue management class called `Missions`. This class provides support for queue management and for dispatching robot missions when robots are idle.

```
1  public static void main(String[] args) {
2
3    //Maximum acceleration/deceleration and speed for all robots
4    double MAX_ACCEL = 1.0;
5    double MAX_VEL = 4.0;
6
7    //Create a coordinator with interfaces to robots
```

```
8       //in the built-in 2D simulator
9       final TrajectoryEnvelopeCoordinatorSimulation tec = new
            TrajectoryEnvelopeCoordinatorSimulation(MAX_VEL,MAX_ACCEL);
10
11      //Provide a heuristic (here, closest goes first)
12      tec.addComparator(new Comparator<RobotAtCriticalSection> () {
13        @Override
14        public int compare(RobotAtCriticalSection o1,
              RobotAtCriticalSection o2) {
15          CriticalSection cs = o1.getCriticalSection();
16          RobotReport robotReport1 = o1.getRobotReport();
17          RobotReport robotReport2 = o2.getRobotReport();
18          return ((cs.getTe1Start()-robotReport1.getPathIndex())-(cs.
                getTe2Start()-robotReport2.getPathIndex()));
19        }
20      });
21
22      //Define a network with uncertainties
23      NetworkConfiguration.setDelays(0,3000);
24      NetworkConfiguration.PROBABILITY_OF_PACKET_LOSS = 0.1;
25
26      //Tell the coordinator
27      // (1) what is known about the communication channel, and
28      // (2) the accepted probability of constraint violation
29      tec.setNetworkParameters(NetworkConfiguration.
            PROBABILITY_OF_PACKET_LOSS, NetworkConfiguration.
            getMaximumTxDelay(), 0.01);
30
31      //Set up infrastructure that maintains the representation
32      tec.setupSolver(0, 100000000);
33
34      //Start the thread that revises precedences at every period
35      tec.startInference();
36
37      //Robot IDs can be non-sequential (but must be unique)
38      int[] robotIDs = new int[] {22,7,54,13,1,14};
39
40      for (int robotID : robotIDs) {
41
42        //Use a random polygon as this robot's geometry (footprint)
43        Coordinate[] fp = makeRandomFootprint(...);
44        tec.setFootprint(robotID,fp);
45
46        //Set a forward model (all robots have the same here)
47        tec.setForwardModel(robotID, new
              ConstantAccelerationForwardModel(MAX_ACCEL, MAX_VEL, tec.
              getTemporalResolution(), tec.getControlPeriod(), tec.
              getTrackingPeriod()));
48
49        //Define start and goal poses for the robot
50        Pose[] startAndGoal = makeRandomStartGoalPair(...);
51
52        //Place the robot in the start pose
53        tec.placeRobot(robotID, startAndGoal[0]);
54
55        //Path planner for each robot (with empty map)
56        ReedsSheppCarPlanner rsp = new ReedsSheppCarPlanner();
57        rsp.setRadius(0.2);
58        rsp.setTurningRadius(4.0);
59        rsp.setDistanceBetweenPathPoints(0.5);
60        rsp.setFootprint(fp);
61
62        //Plan path from start to goal and vice-versa
63        rsp.setStart(startAndGoal[0]);
64        rsp.setGoals(startAndGoal[1]);
```

```
65        if (!rsp.plan()) throw new Error ("No path between " +
              startAndGoal[0] + " and " + startAndGoal[1]);
66        PoseSteering[] path = rsp.getPath();
67        PoseSteering[] pathInv = rsp.getPathInv();
68
69        //Define forward and backward missions and enqueue them
70        Missions.enqueueMission(new Mission(robotID,path));
71        Missions.enqueueMission(new Mission(robotID,pathInv));
72
73        //Path planner to use if re-planning is needed
74        tec.setMotionPlanner(robotID, rsp);
75      }
76
77      //Avoid deadlocks via global re-ordering
78      tec.setBreakDeadlocks(true, false, false);
79
80      //Start a visualization (will open a new browser tab)
81      BrowserVisualization viz = new BrowserVisualization();
82      viz.setInitialTransform(49, 5, 0);
83      tec.setVisualization(viz);
84
85      //Start dispatching threads for each robot, each of which
86      //dispatches the next mission as soon as the robot is idle
87      Missions.startMissionDispatchers(tec, robotIDs);
88
89    }
```

Note that the example uses features introduced in [10] (see lines 22–29) to ensure safety under communication uncertainty. Liveness is ensured via the use of the global precedence constraint re-ordering method described in [11] (see line 78). As discussed in Section 4, non-live situations may also be due to blocking, which in turn can be avoided globally by goal scheduling or locally via re-planning. While a global goal scheduling strategy is not currently implemented, local re-planning to resolve blockings is. This is achieved in the current implementation by providing a path planner that can be used for online re-planning when needed (see line 74).

6.2 Mission management and support for roadmaps

As shown already in the previous section, robots are assigned objects of class `Mission`, each of which is constructed using a path in the form of a `PoseSteering` array. The framework provides a simple mission management and dispatching capability via the static methods of class `Missions`. These were used in the previous example to manage a mission queue and to dispatch enqueued missions to idle robots (lines 70–71 and 87). Mission dispatchers can be started for individual robots and with or without mission re-enqueing, e.g.,

```
1   Missions.startMissionDispatchers(tec, true, 1, 54);
2   Missions.startMissionDispatchers(tec, false, 22, 14, 13);
```

would start dispatchers for robots 1, 13, 14, 22 and 54; however, the dispatchers for robots 1 and 54 would re-enqueue missions as soon as they are dispatched, while the other dispatchers would not.

Another convenience method provided in the `Missions` class concerns the management of known paths. The method `Missions.addPathToRoadMap(String start, String goal, PoseSteering[] path)` can be used to add known locations and a path connecting them to an internal data structure representing a roadmap. The roadmap can be queried via the method `Missions.getShortestPath(String ... locations)` to obtain the shortest path connecting the given locations. This is computed using Dijkstra's algorithm on a graph $G = (V, E)$ where each $v \in V$ is a known location,

$(u,v) \in E$ iff there is a path in the roadmap between locations u and v, and the weight of $(u,v) \in E$ is the length of the known path. Locations can be added to the roadmap without adding paths by using the method `Missions.addLocationToRoadMap(String locationName, Pose pose)`.

6.3 Path planning

As shown in the example above, an interface to a path planning tool is provided via class `ReedsSheppCarPlanner`. This class provides access to several implementations of common path planning methods in the Open Motion Planning Library (OMPL) [17]. Different path planning algorithms can be selected by invoking the constructor with one argument of type `PLANNING_ALGORITHM`, the value of which can be of `RRTConnect`, `RRTstar`, `TRRT`, `SST`, `LBTRRT`, `PRMstar`, `SPARS`, `pRRT`, and `LazyRRT`. For details of each algorithm, please consult the OMPL documentation at https://ompl.kavrakilab.org/planners.html.

The `ReedsSheppCarPlanner` extends the abstract class `AbstractMotionPlanner`. This abstract class implements all logic related the use of path planning/re-planning in the `coordination_oru` framework, with the exception of the planning algorithm. Most notably, the `AbstractMotionPlanner` implementation provides initial and goal pose management, re-sampling of computed paths, map management, and mechanisms for keeping track of obstacles that become known online (e. g., obstacle used to achieve deadlock avoidance via re-planning). Maps are represented internally as instances of the `OccupancyMap` class, which also provides methods for parsing maps from images.

6.4 Visualization

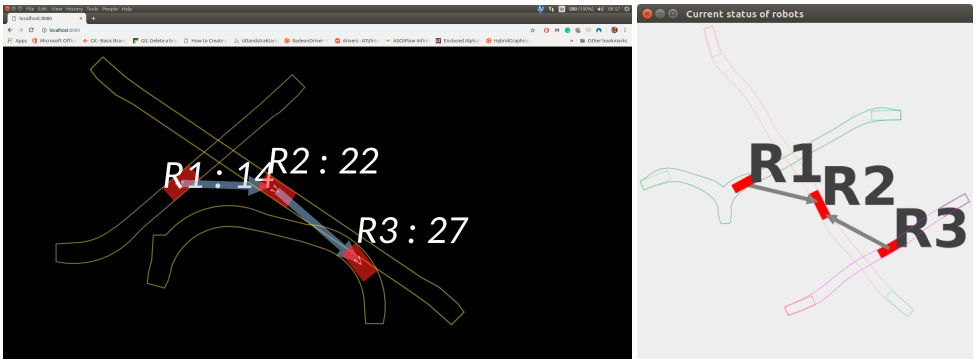
The `coordination_oru` framework provides three visualization methods: a browser-based visualization (class `BrowserVisualization`), a Swing-based visualization (class `JTSDrawingPanelVisualization`), and a visualization based on the ROS tool RViz (class `RVizVisualization`). All visualizations extend class `FleetVisualization`, which can be used as a basis to create new visualization methods.

The browser-based visualization was added in 2019 and is the most effective way to quickly visualize the the status of the fleet. Once instantiated and attached to the the coordinator (see lines 81–83 in the previous listing), a new browser tab (pointing to the URL http://localhost:8080) is opened showing the state of the fleet. Figure 4(a) shows an example of the `BrowserVisualization` for an example provided with the framework. As in all other visualization methods, an arrow from robot Rx to robot Ry indicates that Rx has been instructed to yield to Ry at the next critical section.

The Swing-based GUI provided by class `JTSDrawingPanelVisualization` is shown in Figure 4(b). This GUI allows to take screenshots in SVG, EPS and PDF formats by pressing the `s`, `e` and `p` keys, respectively (while focus is on the GUI window). Screenshots are saved in files named with a timestamp, e. g., `2020-08-13-11:13:17:528.svg`. Saving PDF and EPS files is computationally demanding and will temporarily interrupt the rendering of robot movements, whereas SVG rendering is more efficient.

The `RVizVisualization` visualization publishes visualization markers that can be visualized in RViz. The class also provides the static method `writeRVizConfigFile(int ... robotIDs)` for writing an appropriate RViz configuration file for a given set of robots. An example of the visualization is shown in Figure 5.

The visualization with least computational overhead is the `RVizVisualization`, and is recommended for fleets of many robots. The `BrowserVisualization` class serves an HTML page with a Javascript which communicates with the coordinator via websockets. Although rendering in this solution is less efficient than in RViz, the rendering occurs on the client platform (where the browser is running), so its computational overhead does not



(a) (b)

Figure 4: (a) `BrowserVisualization` and (b) `JTSDrawingPanelVisualization` extensions of the `FleetVisualization` class.

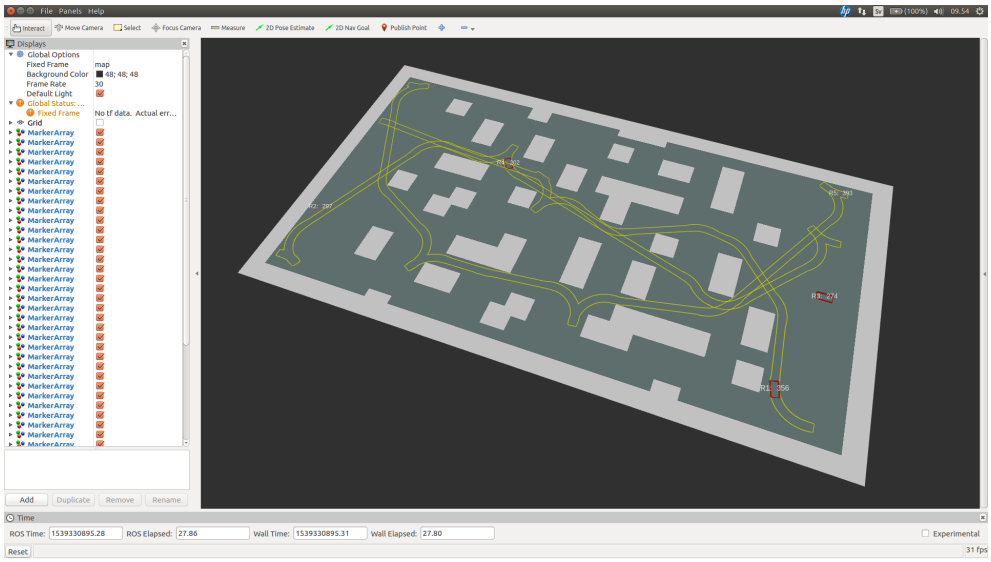


Figure 5: The `RVizVisualization` implementation of the `FleetVisualization` class.

necessarily affect the coordination algorithm. The `JTSDrawingPanelVisualization` is rather slow and not recommended for fleets of more than a handful of robots, however it is practical (not requiring to start another process/program for visualization) and relatively well-tested.

6.5 Interfacing simulated and real robots

As mentioned, the framework includes a low-fidelity 2D simulation engine in which the dynamics of robots are limited to trapezoidal speed profiles. This is used in the example above (see line 9) by means of the `TrajectoryEnvelopeCoordinatorSimulation` class. A similar implementation for interfacing ROS-based robots (or Gazebo simulations thereof) is also provided, in a separate Github repository [4]. Although the implementations of both the simulation subsystem and the ROS interface classes have evolved significantly, there have been few relevant API changes, hence we refer the reader to Deliverable D5.4 or the online documentation on Github [16, 4] for further details.

7 Summary and Conclusions

The results reported in this deliverable have been achieved by work carried out in tasks T5.1, T5.3, T5.4 and T5.5. The work has also contributed to the development of results in tasks T5.2 and T5.6. In the paragraphs below, we circle back to the original aims as stated in the DoW, commenting on their level of achievement.

T5.1 "Efficient task allocation". The integration of task allocation methods described in Section 5 achieves the stated goal of T5.1, which is to achieve an automated, optimizing task allocation method that can be integrated with the coordination and motion planning logic of the framework. The task calls for considering the fact that travel times for individual robots depend on motion plans and coordination with other vehicles traversing the same areas. This is considered in our method via the interference cost term in the objective function (see Section 5). As explained in more detail in Appendix C, systematic search can be substituted with local search methods (e. g., Simulated Annealing) in order to trade solution quality for computational efficiency and scalability.

T5.2 "Quantitative motion planning". A central feature of the approach described in this deliverable is its generality with respect to motion planning. The framework has been integrated with off-the-shelf motion planners available in the research community, as well as with those developed in T5.2 by partner Bosch. We consider this to be a strong advantage of the approach, as it renders the method compatible with application contexts requiring different forms of planning (including existing/legacy route planning methods used in many industries).

T5.3 "Efficient vehicle coordination". As stated in the description of T5.3, our aim was to develop efficient constraint-based solutions to coordination that could be used online. As explained throughout this deliverable, we have achieved a novel, computationally-effective and general method for multi-robot coordination. The method is efficient also in terms of the application context, as it enables the use of any user-defined heuristic. The publications achieved during ILIAD [15, 10, 11, 12] attest to these characteristics and showcase several use-cases beyond that of warehouse automation. Our most recent work (Section 4) has also achieved the sought increase in scalability and provable ability to guarantee liveness.

T5.4 "Integrated task allocation, motion planning and coordination". In this task, we have addressed the long-standing challenge of obtaining a general method for automating and optimizing all aspects of fleet planning and control. The `coordination_oru` framework described in this deliverable achieves this. The integrated method and its generality are grounded on a principle of least commitment, whereby as few assumptions as possible are made on the robot platforms in the fleet, their kinodynamics, the environment, and by ensuring that tasks can be posted online while the fleet is in motion (see also T5.5). Concrete examples of the integration with task allocation are shown in Appendix C. A publication showcasing the integrated system is currently under preparation and is expected to be submitted before the end of the project.

T5.5 "Continuous online requirement posting". It is stated in T5.5 that the integrated task allocation, motion planning and coordination framework should be capable of dealing with requirements that are posted online, while the fleet is in motion. As shown in this deliverable, we have focused on two types of requirements, namely, on the temporal profile of trajectories in execution, and on the tasks. Regarding the former, the mechanism itself by which coordination occurs is that of online constraint posting and revision, where the constraints are precedences between robots that regulate access to critical sections (and which result, at a lower level of abstraction, in critical points posted to the robot controllers). Regarding the latter, we have designed the coordination algorithm (Sections 3 and 4) as well as the task optimization method (Section 5) to accept tasks that are posted online. Furthermore, we have seen how online re-planning can be exploited to avoid blockings and deadlocks without requiring intractable online computation [11].

T5.6 "Distributed Fleet Coordination and Management". The framework described in this deliverable has also contributed to achieving one of the distributed fleet coordination methods in developed in T5.6. Specifically, a distributed variant of the coordination framework was achieved by partner UNIPI using as a basis the implementation of the framework described in Section 6. This is described in Deliverable D5.6.

References

[1] H. Andreasson, J. Saarinen, M. Cirillo, T. Stoyanov, and A. J Lilienthal. Fast, continuous state path smoothing to improve navigation accuracy. In 2015 IEEE International Conf. Robotics & Autom. (ICRA), pages 662–669. IEEE, 2015.

[2] M. Čáp, J. Gregoire, and E. Frazzoli. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In 2016 IEEE/RSJ Int. Conf. Intell. Robots & Syst. (IROS), pages 5113–5118. IEEE, 2016.

[3] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. IEEE Trans. Autom. Sci. Eng, 12(3):835–849, 2015.

[4] Pecora F. The ROS package for an online multi-robot coordination algorithm based on trajectory envelopes (for Kinetic distro), 2019. https://github.com/FedericoPecora/coordination_oru_ros.

[5] R. Ghrist and S. M Lavalle. Nonpositive curvature and pareto optimal coordination of robots. SIAM J. Control & Optimization, 45(5):1697–1713, 2006.

[6] J. Gregoire, S. Bonnabel, and A. de La Fortelle. Optimal cooperative motion planning for vehicles at intersections. arXiv preprint arXiv:1310.7729, 2013.

[7] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. The International Journal of Robotics Research, 32(12):1495–1512, 2013.

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

[9] T. Lozano-Perez. Spatial planning: A configuration space approach. In Autonomous robot vehicles, pages 259–271. Springer, 1990.

[10] A. Mannucci, L. Pallottino, and F. Pecora. Provably safe multi-robot coordination with unreliable communication. IEEE Robotics and Automation Letters, 4(4):3232–3239, 2019.

[11] A. Mannucci, L. Pallottino, and F. Pecora. On provably safe and live multi-robotcoordination with online goal posting. IEEE Transactions on Robotics, 2020. (second revision submitted).

[12] M. Mansouri, B. Lacerda, N. Hawes, and F. Pecora. Multi-robot planning under uncertain travel times and safety constraints. In Proc. 28th Int. Conf. on Artificial Intell., IJCAI-19, pages 478–484. International Joint Conf.s on Artificial Intell. Organization, 7 2019.

[13] Changjoo Nam and Dylan A Shell. Assignment algorithms for modeling resource contention in multirobot task allocation. IEEE Transactions on Automation Science and Engineering, 12(3):889–900, 2015.

[14] L. Pallottino, V. G Scordio, A. Bicchi, and E. Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. IEEE Trans. Robotics, 23(6):1170–1183, 2007.

[15] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov. A loosely-coupled approach for multi-robot coordination, motion planning and control. In Proc. 28th Int. Conf. Autom. Planning & Scheduling, 2018.

[16] F. Pecora, A. Mannucci, and C.S. Swaminathan. A Framework for Multi-Robot Motion Planning, Coordination and Control, 2020. https://github.com/FedericoPecora/coordination_oru, version 0.6.1.

[17] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. IEEE Rob. & Autom. Mag., 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[18] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine, 29(1):9–9, 2008.

Appendix A Provably Safe Multi-Robot Coordination With Unreliable Communication (R-AL 2019)

Provably Safe Multi-Robot Coordination With Unreliable Communication

Anna Mannucci, Lucia Pallottino, and Federico Pecora

***Abstract*—Coordination is a core problem in multi-robot systems, since it is a key to ensure safety and efficiency. Both centralized and decentralized solutions have been proposed, however, most assume perfect communication. This letter proposes a centralized method that removes this assumption, and is suitable for fleets of robots driven by generic second-order dynamics. We formally prove that: first, safety is guaranteed if communication errors are limited to delays; and second, the probability of unsafety is bounded by a function of the channel model in networks with packet loss. The approach exploits knowledge of the network's non-idealities to ensure the best possible performance of the fleet. The method is validated via several experiments with simulated robots.**

***Index Terms*—Multi-robot systems, planning, scheduling and coordination, formal methods in robotics and automation.**

I. Introduction

MODELING and accounting for the limitations of communication is extremely important in multi-robot systems, both for performance and for safety purposes. Wireless technologies are necessary for robot mobility; however, they suffer from connectivity loss, spectrum interference and high latency handover [1]. Fleets of AGVs are particularly important in harsh environments such as underground mines, where it is notoriously problematic to guarantee reliable communication [2]. Communication standards providing low delays and packet loss have been proposed for use in automotive [3] and industrial applications [4]. However, approaches that tackle these limitations in conjunction with the multi-robot coordination problem make strong assumptions on paths and on robot kinodynamics (see, e.g., [5], [6]).

Conversely, the literature on multi-robot coordination overlooks the realities of the communication infrastructure, typically focusing on one or more of the following key requirements of real-world applications [7]: the robots in the fleet are heterogeneous and subject to non-trivial kinodynamic constraints; goals become known and are posted asynchronously (e.g., by a separate and pre-existing workflow management system); the need to discretize the environment and/or robot paths should be minimized, as this raises the cost of deployment; it should be possible to regulate the motions of robots through shared regions according to application-specific priorities; methods should scale to dozens of robots and large environments; and coordination should guarantee safety (no collisions) and liveness of the fleet (robots will eventually reach their goals).

To the best of our knowledge, no existing approach adheres to the requirements above while accounting for communication limitations. Table I summarizes how selected approaches[1] to multi-robot coordination relate to these requirements and to communication-related aspects. The specific limitations of existing approaches (see detailed notes in Table I) reveal the reason why centralized decision making structures still dominate in industrial practice: while distributed or decentralized solutions are by nature relatively robust to communication failures [9], [11]–[15], centralized methods deliver predictable fleet behavior, provable safety and liveness, and high performance [8], [10], [16]–[18].

In this letter, we explore the level of safety that can be guaranteed with knowledge of a model of the communication channel. Our algorithm builds on a supervisory control method proposed in [18], where precedences for potentially colliding pairs of robots are updated continuously, taking into account robot kinodynamics. While maintaining its good properties (see Table I), we extend [18] in the following ways: (i) the assumption of perfect communication (necessary for safety in [18]) is removed; (ii) the probability of robots not respecting a precedence is guaranteed to be below a desired threshold (knowing an upper-bound of the packet loss probability and the maximum transmission delay); (iii) we prove that this threshold is zero if network disturbances are limited to delays; (iv) we provide a set of rules to design the network to ensure a desired level of safety, avoiding network congestion. All properties are validated formally, and an experimental evaluation highlights the behavior of the algorithm in realistically-sized fleets, with typical channel

Manuscript received February 24, 2019; accepted June 4, 2019. Date of publication June 24, 2019; date of current version July 15, 2019. This letter was recommended for publication by Associate Editor J. Alonso-Mora and Editor N. Y. Chong upon evaluation of the reviewers' comments. This work was supported in part by the EU's Horizon 2020 research and innovation program under Grant 732737 (ILIAD) and in part by the Swedish Knowledge Foundation (KKS) under the Semantic Robots research profile. *(Corresponding author: Anna Mannucci.)*

A. Mannucci and L. Pallottino are with the Research Center "E. Piaggio," University of Pisa, 56122 Pisa, Italy (e-mail: anna.mannucci@ing.unipi.it; lucia.pallottino@unipi.it).

F. Pecora is with the Center for Applied Autonomous Sensor Systems, Örebro University, SE-701 82 Örebro, Sweden (e-mail: federico.pecora@oru.se).

This letter has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The material consists of a video showing a portion of one of the simulated runs performed for each of the tests reported in the quantitative evaluation in Section VII.

Digital Object Identifier 10.1109/LRA.2019.2924849

[1] A comprehensive overview of multi-robot coordination methods is beyond the scope of this letter, and the interested reader is referred to [8], [9]. We also exclude approaches for solving the task allocation and/or motion planning problem jointly with coordination, e.g., Multi-Agent Path Finding (MAPF) methods; the problem at hand in these cases is intractable [10], hence approaches tend to adhere to few of the requirements listed above.

TABLE I
CLASSIFICATION OF RELATED WORK ACCORDING TO KEY REQUIREMENTS OF REAL-WORLD APPLICATIONS [7]

	Centralized coordination					Decentralized or distributed coord.				Reactive	Distr. MPC[(n)]
Feature / Reference	[16]	[8][(e)]	[17]	[18]	[10][(e)]	[11][(e)]	[12][(e)]	[13]	[9][(e)]	[14]	[15]
Heterogeneous fleets	✓[(a)]	✓	✓	✓	×	✓[(a)]	✓[(a)]	×	✓	✓[(b)]	✓[(b)]
Avoid environment or path discretization	×	✓	✓	✓	✓	×	×	×	✓	✓	✓
Generic motion planners	×[(c)]	✓	✓	✓	✓	×[(c)]	×[(c)]	×[(d)]	✓	-[(k)]	✓
Kinodynamic constraints	×	✓	✓	✓	×	×	×	✓	✓	✓	✓
Asynchronous goal posting	✓	×	✓	✓	×	×	✓	✓	✓	✓	✓
Avoiding static priorities	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓[(l)]
Computation time (coordination)	n.s.	O(C)[(j)]	O(e^{R})	O(C)[(j)]	O(C)[(j)]	O(R)	O(SR)	O(SR)	O(PRS^2)	O($\mathcal{O}$)	n.s.
Scalability (# robots tested in sim.)	n.s.	12	10	30	50	9	100	10	32	100	6
Safe with delays in trajectory execution	×	×	×[(f)]	✓	✓	✓	✓	✓	×	-[(k)]	×[(m)]
Safe with unreliable communication	×	×	×[(f)]	×	×	×[(i)]	×[(g)]	×[(g)]	×[(h)]	×[(k)]	×[(m)]
Safe with clock de-synchronization	×	×	×	✓	×	×	✓	✓	✓	×[(k)]	×
Detailed commun. requirements for safety	×	×	×	×	×	×	×	×	×[(p)]	×	×
Provable liveness with ideal commun.	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×

a: Assuming robots fit within allotted spatial resources. b: Assuming disc-shaped robots. c: Graph search w/o motion primitives. d: Graph search w/ motion primitives. e: Tested only in simulation. f: Only for limited, bounded delays. g: No formal relationship between communications and speed, sensing neighbor states, or sensing distance provided; unsuccessful sensing/occlusion of neighbors not handled. h: Assumed that robots can detect dropped messages; collisions may occur due to occlusions. i: Communication radius required to be greater than a specific parameter of the graph. j: Limited to critical section computation. k: No path is required, but assumes perfect reciprocal visibility and does not ensure absence of collisions. l: Assumes static priorities. m: Assumes environment free of static obstacles. n: Model Predictive Control. **n.s.**: Not specified. **R**: # robots. **C**: # pairwise critical sections (Section II). **S**: # path segments. **P**: # possible equivalent plans. $\mathcal{O}$: # obstacles.

models. Note that properties (iii) and (iv) provide a means to design the communication infrastructure considering the requirements of the fleet. To the best of our knowledge, this issue has never before been studied in the multi-robot research area.

II. NOTATION AND PRELIMINARIES

We first introduce key concepts and a high-level description of the algorithm in [18] as a basis for our approach. Hence, for now, perfect communication is assumed. This assumption will be relaxed in Sections III and IV, while preserving safety with a computable probability of violation.

Paths and spatial envelopes. Consider a fleet of n (possibly heterogeneous) robots sharing an environment $\mathcal{W} \subset \mathbb{R}^3$. We use $(\cdot)_i$ to indicate that variable $(\cdot)$ refers to robot i. Let $\mathcal{Q}_i$ be the robot's configuration space, and $R_i(q) \subset \mathbb{R}^3$ its collision space when in configuration $q \in \mathcal{Q}_i$. Consider a set of obstacles $\mathcal{O} \subset \mathcal{W}$, so that $\mathcal{Q}_i^{\text{free}} = \{q \in \mathcal{Q}_i : R_i(q) \cap \mathcal{O} = \emptyset\}$ is the set of feasible (i.e., collision free) configurations. Let $\boldsymbol{p}_i : [0,1] \to \mathcal{Q}_i$ be a path in the configuration space parametrized using the arc length $\sigma \in [0,1]$. Then, *path planning* is the problem of finding a (possible executable) path $\boldsymbol{p}_i(\sigma) \in \mathcal{Q}_i^{\text{free}}$ from one feasible starting configuration q^{start} to a final one $q^{\text{goal}} \in \mathcal{Q}_i^{\text{free}}$, such that $q^{\text{start}} = \boldsymbol{p}_i(0)$ and $q^{\text{goal}} = \boldsymbol{p}_i(1)$, typically subject to a set of kinematic constrains $f_i(q, \dot{q}) \leq 0$ (Fig. 1(a)). Furthermore, for each $\boldsymbol{p}_i$, the *spatial envelope* $\mathcal{E}_i$ is defined as a set of constraints such that $\cup_{\sigma \in [0,1]} R_i(\boldsymbol{p}_i(\sigma)) \subseteq \mathcal{E}_i$. If the equality holds (which we assume from now on), a spatial envelope is the sweep of the robot's footprint along its path (Fig. 1(b)). Henceforth, let $\mathcal{E}_i^{\{\sigma', \sigma''\}} = \cup_{\sigma \in [\sigma', \sigma'']} R_i(\boldsymbol{p}_i(\sigma))$.

Note that $\mathcal{E}_i \cap \mathcal{O} = \emptyset \ \forall i \in \{1, \dots, n\}$ by construction, that is, collisions between robots and the set of obstacles $\mathcal{O}$ are avoided via path planning. Also, we assume that robots are provided with a low-level safety system for detecting and avoiding obstacles that are not other robots and are not included in $\mathcal{O}$. The focus of the fleet controller proposed in this letter is therefore to avoid inter-robot collisions, not other unforeseen obstacles.

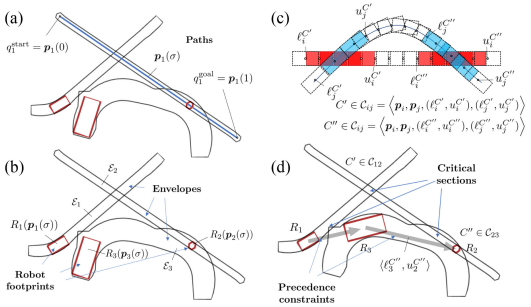


Fig. 1. Preliminary concepts.

Critical sections. Given a pair of paths $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$, collisions may happen only in the set $\{q_i \in \mathcal{Q}_i, q_j \in \mathcal{Q}_j \mid R_i(q_i) \cap \mathcal{E}_j \neq \emptyset \ \vee \ R_j(q_j) \cap \mathcal{E}_i \neq \emptyset\}$. In particular, let $\mathcal{C}_{ij}$ be the decomposition of this set into its largest contiguous subsets, each of which is called a *critical section* (Fig. 1(c) and 1(d)). For each critical section $C \in \mathcal{C}_{ij}$, let $\ell_i^C \in [0,1]$ be the highest value of σ_i before robot i enters C; similarly, let $u_i^C \in [0,1]$ be the lowest value of σ_i after robot i exits C. Considering two temporal profiles $\sigma_i(t)$ and $\sigma_j(t)$, if there exists a time t' such that $R_i(\boldsymbol{p}_i(\sigma_i(t'))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t'))) \neq \emptyset$ (i.e., the robots collide while laying in their envelopes), then $\ell_i^C < \sigma_i(t') < u_i^C$ and $\ell_j^C < \sigma_j(t') < u_j^C$. Hence, given a set of paths $\mathcal{P}$, the *coordination problem* is the problem of synthesizing, for each pair $(i, j \neq i)$ such that $\mathcal{E}_i \cap \mathcal{E}_j \neq \emptyset$, a constraint on temporal profiles $\sigma_i(t)$ and $\sigma_j(t)$ such that $R_i(\boldsymbol{p}_i(\sigma_i(t'))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t'))) = \emptyset$ for all t'. We assume that, when idle, a robot i is placed in a parking position defined by a path $\boldsymbol{p}_i$ of length one. This entails that idle robots are considered in the computation of critical sections.

Precedence constraints and critical points. Precedence constraints are relations among the temporal profiles of two robots. A precedence constraint is a pair $\langle m_i, m_j \rangle$, with $m_i, m_j \in [0,1]$,

stating that robot i is not allowed to navigate beyond arc length m_i along its path until robot j has reached arc length m_j along its path — formally, $\sigma_j(t) < m_j \Rightarrow \sigma_i(t) < m_i$. As explained in [18], m_i changes over time to reflect updated precedences and to allow for robots to "follow each other" through critical sections. In general, collisions are avoided if, for each $C \in \mathcal{C}_{ij}$ and for each t, $\sigma_i(t)$ and $\sigma_j(t)$ adhere to the constraint $\langle m_i(t), u_j^C \rangle$, that is, robot i yields for robot j at an appropriately computed arc length $m_i(t)$ along its reference path; this arc length depends on whether robot j has exited critical section C (that is, reached arc length u_j^C) and on its current progress through the critical section:

$$m_i(t) = \begin{cases} \max\left\{\ell_i^C, r_{ij}(t)\right\} & \text{if } \sigma_j(t) \leq u_j^C \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

where $r_{ij}(t)$ is defined as

$$\sup_\sigma \left\{ \sigma \in [\sigma_i(t), u_i^C] : \mathcal{E}_i^{\{\sigma_i(t),\sigma\}} \cap \mathcal{E}_j^{\{\sigma_j(t),u_j^C\}} = \emptyset \right\}. \quad (2)$$

Let $\mathcal{T}$ be the set of precedence constraints regulating the motion of the robots in the fleet. A constraint $\langle m_i, u_j^C \rangle \in \mathcal{T}$ defines unambiguously which robot should yield, where it should yield, and until when yielding is necessary for critical section C. We use $(i <_C j) \in \mathcal{T}$ to indicate that robot j has precedence over robot i at a critical section C. A key feature of the approach is that $\mathcal{T}$ can be updated while robots are in motion. In particular, any heuristic function can be used to determine the precedence constraints in $\mathcal{T}$, as long as a conservative model of each robot's dynamics is employed to filter out ordering decisions that may not be physically realizable (as detailed in [18]).

Let $T_i(t) = \{m_i \mid \exists j : \langle m_i, u_j^C \rangle \in \mathcal{T}(t)\}$ be the set of all the arc lengths at which robot i may be required to yield. We define the *critical point* $\bar{\sigma}_i(t)$ of robot i at time t as the value of σ corresponding to the last reachable configuration along $\boldsymbol{p}_i$ which adheres to the set of constraints $\mathcal{T}(t)$, i.e.,

$$\bar{\sigma}_i(t) = \begin{cases} \arg\min_{m_i \in T_i(t)} m_i & \text{if } T_i(t) \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \quad (3)$$

Then, *coordination* is the problem of computing and updating periodically the set of critical points $\bar{\Sigma} = \{\bar{\sigma}_1, \ldots, \bar{\sigma}_n\}$, such that collisions do not occur. Algorithm 1 shows the main body of the supervisory control loop proposed in [18].

Under the assumption of a perfect communication (messages are not delayed or lost), and conservative models of the robots' dynamics, the algorithm ensures that collisions never happen (see [18] for a formal proof). If the assumptions on the channel are removed, then safety no longer holds.

Consider, for instance, a change in the order of access to a critical section $C \in \mathcal{C}_{ij}$ between two consecutive cycles of coordination. Let T_c be the control period of the coordinator, and assume that robot j has precedence over robot i at C at time $t - T_c$. Hence, $\bar{\sigma}_i(t - T_c) = \ell_i^C$ and $\bar{\sigma}_j(t - T_c) > \ell_j^C$. If robot j has not already entered C, i.e., $\sigma_j(t - T_c) < \ell_j^C$, and j can stop before ℓ_j^C according to its dynamic model, then the coordinator may decide to reverse precedence, so that $\bar{\sigma}_j(t) = \ell_j^C$. In this case, if the message to j containing the new critical point $\bar{\sigma}_j(t)$ is delayed or lost, then a collision may happen. The same issue may occur whenever i is starting a new path while j is already driving and $(j <_C i) \in \mathcal{T}$ for some $C \in \mathcal{C}_{ij}$. This undesired situation can occur since the coordination algorithm does not explicitly reason about the non-idealities of the network. The following sections will show how prior knowledge about the channel can be included in the algorithm so that safety is preserved.

Algorithm 1: Coordination at Time t.

```
sample states[2];
if new goals have been posted then
    update the set of paths P (using appropriate planners);
    update the set C of critical sections;
revise the set T(t) of precedence constraints;
compute the set of critical points Σ̄(t);
communicate changed critical points;
sleep until control period T_c has elapsed;
```

III. Problem Formulation

Given a set of paths $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$, we aim to define an algorithm to coordinate the fleet via message-passing. The boundary conditions of the multi-robot system are summarized as follows. We assume a duplex point-to-point communication via wireless network (subject to delays and/or message loss) between the coordinator and each agent. Message order can be reconstructed [19] via time-stamps, and message replicas can be filtered out (whether these are sent to increase probability of message reception, or due to multi-path phenomena). Each robot i has a control period of T_i seconds (robots may have different control periods) and sends to the coordinator an update on its state $s_i(t_i)$ sampled within its control period of (clock-driven system). The state report contains the tuple $(q_i(t_i), \dot{q}_i(t_i), \ddot{q}_i(t_i))$, as well as the last critical point $\bar{\sigma}_i$ received by the robot. We also assume that the coordinator receives at least one update of each agent's state every T_c seconds, and that $T_c \geq \max_{i \in 1,\ldots,n} T_i$. Note that robots are not required to be synchronized on a common Coordinated Universal Time (UTC).

A. Channel Model

Wireless networks are susceptible to a number of factors that may corrupt packets in transit such as radio frequency interference (RFI), radio signals that are too weak due to distance or multi-path fading, faulty networking hardware, faulty network drivers, or network congestion. Different levels of network-induced imperfections may affect the communication such as time delays, packet loss and disorder, or clock desynchronization [20]. We model the network considering three parameters: the *maximum bandwidth*, the *packet loss probability*, and the *maximum transmission delay*.

Let $\mathcal{B}(t)$ be the bandwidth of the channel at time t, and let $\mathcal{B}_M$ be the maximum bandwidth (bit/s). Packet loss occurs when one or more packets of data traveling across a network fail to reach their destination. It is measured as the percentage of packets lost

[2] Assuming an ideal communication network, current robot states are available via message passing without delays, message loss or disorder.

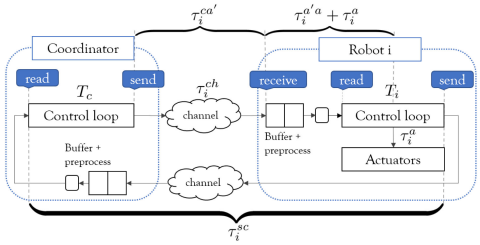


Fig. 2. Delays of a point-to-point communication between the coordinator and each robot i.

with respect to packets sent. We model this phenomenon using a Bernoulli distribution, and assuming the upper-bound of the packet loss probability η to be independent from the identities and locations of the source and destination [20]. Hence, assuming $\mathcal{B}_M = \infty$ (no congestion), the minimal number of replicas (messages containing the same information) required for a successful delivery with probability almost $\bar{p}$ can be computed as

$$N \geq \left\lceil \frac{\log(1-\bar{p})}{\log \eta} \right\rceil. \tag{4}$$

Let $\tau^{ch}_{\max}$ represent the maximum transmission delay (sec), that is, the upper-bound of time elapsed between a send event and the related receive event. This delay is usually the sum of network access delay (i.e., the time required by a queued network packet to be sent out) and the transmission delay through the network medium. While the first is related to the channel's bandwidth, the second is intrinsic of the transmission, and the longer the distance, the longer the delay under the same conditions (bandwidth, protocols, etc.). Also, let $\tau_i^{ch}(t) \leq \tau^{ch}$ be its current realization in a point-to-point communication from/to robot i (Fig. 2). As in [21], we assume $\tau^{ch}_{\max}$ to be constant, symmetric and independent from the source and the destination. Due to congestion, the packet loss probability and the transmission delay may not be independent from the load of the channel. However, for our field of application, it is reasonable to assume the network to be dedicated for fleet management.[3] Hence, given the number of robots, the required bandwidth can be explicitly computed to design the network to avoid congestion (see Section VI). Under this assumption, we assume η and $\tau^{ch}_{\max}$ to be uncorrelated with $\mathcal{B}(t)$.

B. The Coordination Problem

At each control cycle T_c, the coordinator should decide a *correct* (i.e., collision free) and *feasible* (i.e., physically executable) set of critical points $\bar{\Sigma}$, according to its current view. In doing so, to preserve safety, it should reason about delays affecting the control system. For each robot, we define the maximum delay between sensing and actuation $\tau_i^{sa} = \tau_i^{sc} + \tau_i^{ca'} + \tau_i^{a'a} + \tau_i^a$ whose components are defined as follows (see also Fig. 2).

τ_i^{sc} is the time elapsed between robot i's state being transmitted and read by the coordinator (sensing delay); assuming robots to be asynchronous, and at least one updated set $\{s_1, \ldots, s_n\}$ to be available within each T_c, then $\tau_i^{sc} = T_c + \tau^{ch} + T_i$ in the worst case. $\tau_i^{ca'}$ is the time required for a critical point $\bar{\sigma}_i$ to be received by robot i; assuming the communication protocol requires to send a burst of M packets for each $\bar{\sigma}_i$, and letting T_p be the period between two consecutive packet deliveries, then $\tau_i^{ca'} = \tau_i^{ch} + (M-1)T_p$; since reasonably $(M-1)T_p \ll T_c$, we assume $\tau_i^{sc} + \tau_i^{ca'} = T_c + 2\tau^{ch} + T_i$ in the worst case. $\tau_i^{a'a} + \tau_i^a$ is the delay between when robot i receives a message and the corresponding action is executed (e.g., to yield); this is a function of T_i, the message queue length Q_i, and the robot control system; if $Q_i = 1$ as reasonable, then $\tau_i^{a'a} = T_i$ in the worst case. We also assume no actuation delay, $\tau_i^a = 0$.

Let τ_i^{break} be the maximum breaking time for robot i, which depends on its dynamics and maximum speed. For each $C \in \mathcal{C}_{ij}(t)$, we can assess the feasibility of changing the priority of access to C by looking into the future. In the worst case, a command to yield sent by the coordinator at time t will make the robot stop at time $t + \Delta_i^{\text{stop}}$, where $\Delta_i^{\text{stop}} = \tau_i^{sa} + \tau_i^{\text{break}}$. Hence, if $(i <_C j) \in \mathcal{T}(t - T_c)$, and the coordinator has received the state of robots i and j at times t_i and t_j, it can decide to change the precedence of the two robots to $(j <_C i) \in \mathcal{T}(t)$ iff $\sigma_j(t_j + \Delta_j^{\text{stop}}) \leq \ell_j^C$ (i.e. the robot which loses the priority can effectively stop before entering the critical section). Algorithm 2 implements this feasibility check, given a conservative dynamic model g_i and maximum acceleration/deceleration $u_i^{\text{acc}}, u_i^{\text{dec}}$ of the robot. Specifically, the robot is clearly capable of stopping (line 1) if it was already constrained to stop at a critical point preceding the beginning of the critical section C at the previous control period (i.e., at time $t - T_c$). If this is not the case, the robot's dynamic model g_i is used to compute whether it can achieve zero velocity before the critical point (lines 3–10). In doing so, it assumes that the robot has progressed with maximal acceleration from its last reported state $(q_i(t_i), \dot{q}_i(t_i), \ddot{q}_i(t_i))$ for a period of Δ_i^{stop} (lines 3–6). Note that a robot state report with $\dot{q}(t_i) = 0$ is not sufficient to conclude that the robot can stop moving, as this state was sampled at time t_i and the robot may have started moving in the meantime.

Algorithm 2: The canStop Function.

Input: $(q_i(t_i), \dot{q}_i(t_i), \ddot{q}_i(t_i))$ last known state; $\bar{\sigma}_i(t - T_c)$ last communicated critical point (or -1 if none was communicated); $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i, t)$ dynamic model of the robot; $u_i^{\text{acc/dec}}$ maximal acceleration/deceleration; Δ_i^{stop} look-ahead; ℓ_i^C stopping point at critical section C; $\boldsymbol{p}_i$ current path; Δt integration time step.

Output: true iff robot i can stop before entering C

1 **if** $\bar{\sigma}_i(t - T_c) \neq -1 \wedge \bar{\sigma}_i(t - T_c) \leq \ell_i^C$ **then return** true ;
2 $t \leftarrow t_i$;
3 **while** $t < t_i + \Delta_i^{\text{stop}}$ **do**
4 $\quad t' \leftarrow t + \Delta t$;
5 $\quad (q_i(t'), \dot{q}_i(t'), \ddot{q}_i(t')) \leftarrow g_i(q_i(t), \dot{q}_i(t), \ddot{q}_i(t), u_i^{\text{acc}}, t')$;
6 $\quad t \leftarrow t'$;
7 **while** $\dot{q}_i(t) > 0$ **do**
8 $\quad t' \leftarrow t + \Delta t$;
9 $\quad (q_i(t'), \dot{q}_i(t'), \ddot{q}_i(t')) \leftarrow g_i(q_i(t), \dot{q}_i(t), \ddot{q}_i(t), u_i^{\text{dec}}, t')$;
10 $\quad t \leftarrow t'$;
11 **return** $\boldsymbol{p}_i^{-1}(q_i(t)) \leq \ell_i^C$;

[3] So it is not necessary to model the network queue as a Markov process.

IV. The Coordination Algorithm

In (1) and (2), we assume that r_{ij} (the furthest σ_i the yielding robot is allowed to reach) is computed with knowledge of the current progress $\sigma_i(t)$, $\sigma_j(t)$ of the two robots. Due to possible delays in communication, we must now rely on $\sigma_i(t_i)$ and $\sigma_j(t_j)$, where $t - \tau_i^{sc} \leq t_i < t$ and $t - \tau_j^{sc} \leq t_j < t$. Hence,

$$m_i(t) = \begin{cases} \max\left\{\ell_i^C, r_{ij}(t)\right\} & \text{if } \sigma_j(t_j) \leq u_j^C \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

and $r_{ij}(t)$ is computed as

$$\sup_\sigma \left\{ \sigma \in [\sigma_i(t_i), u_i^C] : \mathcal{E}_i^{\{\sigma_i(t_i),\sigma\}} \cap \mathcal{E}_j^{\{\sigma_j(t_j),u_j^C\}} = \emptyset \right\}. \tag{6}$$

With this revised formulation of precedence constraints, we can now define an algorithm for coordination which guarantees correct behavior of the fleet in the presence of non-ideal communication channels. Algorithm 3 is based on the principle of Algorithm 1: at each T_c, the states s_i of all robots are updated with the last report received, each at a potentially different time t_i (line 4); paths are computed for idle robots for which a new goal has been posted (lines 5–9); critical sections are updated (lines 10–11); and constraints are revised (lines 12–13) and communicated to the robots (line 14). At the core of the coordination algorithm is a call to the function *revise*, detailed in Algorithm 4. For each critical section, this function decides the order of traversal, according to the current state of the involved robots (lines 2–8), using the aforementioned *canStop* function (see Algorithm 2). Then, it updates the set of precedence constraints $\mathcal{T}(t)$ according to (5) and (6) (lines 9–10).

In particular, for each critical section C, if neither of the two involved robots is known to have passed the critical section's upper bound (line 3), and both of them can achieve zero velocity before entering it (line 4), then an ordering is heuristically[4] decided. If one of the two robots has entered or cannot stop before entering C (lines 6–7), then that robot is given precedence. If this is the case for both robots, the previously decided ordering is re-imposed (line 8). Note that an idle robot involved in a critical section is necessarily already inside it, hence, will always have precedence over the other robot.

V. Safety Analysis

In this analysis, we make the following assumptions:

A1. Paths do not start or end in critical sections.

A2. Robots always stay within their envelopes.

A3. The channel delay τ_i^{ch} is bounded.

AV and AV are made to simplify the analysis — the algorithm can be easily modified to include these specific cases. AV is a reasonable assumption to make on any real network. We start by considering $\eta = 0$ (no packet loss), while the case of $\eta > 0$ is considered in Section VI.

Lemma 1: Algorithms 4 and 2 satisfy the preposition: *if* $\mathcal{T}(t')$ is a set of feasible constraints, *then* $\mathcal{T}(t'')$ is a set of feasible constraints $\forall t'' > t'$.

[4]Note that, as in [18], any heuristic can be chosen here.

Algorithm 3: The Coordination Algorithm.

Input: $\mathcal{G}$ set of goals posted for robots $\{1, \ldots, n\}$.

1 $\mathcal{P} \leftarrow \emptyset, \mathcal{C} \leftarrow \emptyset, \mathcal{T} \leftarrow \emptyset, \bar{\Sigma} \leftarrow \{-1\}^n$;
2 **while** true **do**
3 $t \leftarrow$ getCurrentTime();
4 **for** $i \in [1, \ldots, n]$ **do** $s_i(t_i) \leftarrow$ getStatusMsg(i) ;
5 **for** $i : g_i \in \mathcal{G} \wedge$ isIdle$(s_i(t_i))$ **do**
6 $\mathcal{G} \leftarrow \mathcal{G} \setminus \{g_i\}$;
7 remove robot i from $\mathcal{P}$, $\bar{\Sigma}(t - T_c)$ and $\mathcal{C}$;
8 $\boldsymbol{p}_i \leftarrow$ computePath$(s_i(t_i), g_i)$;
9 $\mathcal{P} \leftarrow \mathcal{P} \cup \{\boldsymbol{p}_i\}$;
10 **for** $(\boldsymbol{p}_i, \boldsymbol{p}_{j\neq i}) \in \mathcal{P}^2$ **do**
11 $\mathcal{C} \leftarrow \mathcal{C} \cup$ getIntersections$(\mathcal{E}_i(\boldsymbol{p}_i), \mathcal{E}_j(\boldsymbol{p}_j))$;
12 $\mathcal{T}(t) \leftarrow$ revise$(\mathcal{P}, \mathcal{C}, \mathcal{T}(t-T_c), \bar{\Sigma}(t-T_c), \{s_1(t_1), \ldots, s_n(t_n)\})$;
13 $\bar{\Sigma}(t) \leftarrow \forall i$ compute $\bar{\sigma}_i$ as in (3) or -1 if i is idle;
14 **for** $i \in [1, \ldots, n]$ **do** send$(i, \bar{\sigma}_i(t))$;
15 $\Delta t =$ getCurrentTime$() - t$;
16 **if** $\Delta t < T_c$ **then** sleep(Δt) ;

Algorithm 4: The Revise Function.

Input: $\mathcal{P}$ current set of paths; $\mathcal{C}$ (possibly empty) set of pairwise critical sections; $\mathcal{T}(t - T_c)$ (possibly empty) previous set of precedence constraints; $\bar{\Sigma}(t - T_c)$ previous set of critical points; $s_i(t_i)$ last received robot state, including $\sigma_i(t_i)$.

Output: $\mathcal{T}_{\text{rev}}$ set of revised precedence constraints.

1 $\mathcal{T}_{\text{rev}} \leftarrow \emptyset$;
2 **for** $\mathcal{C}_{ij} \in \mathcal{C}, C \in \mathcal{C}_{ij}$ **do**
3 **if** $\sigma_i(t_i) < u_i^C \wedge \sigma_j(t_j) < u_j^C$ **then**
4 **if** $\sigma_i(t_i + \Delta_i^{\text{stop}}) \leq \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\text{stop}}) \leq \ell_j^C$ **then**
5 $(h <_C k) \leftarrow$ compute ordering with a heuristic;
6 **else if** $\sigma_i(t_i + \Delta_i^{\text{stop}}) > \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\text{stop}}) \leq \ell_j^C$ **then**
$(h <_C k) \leftarrow (j <_C i)$;
7 **else if** $\sigma_i(t_i + \Delta_i^{\text{stop}}) \leq \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\text{stop}}) > \ell_j^C$ **then**
$(h <_C k) \leftarrow (i <_C j)$;
8 **else** $(h <_C k) \leftarrow$ get previous ordering from $\mathcal{T}$;
9 $\langle m_h, u_k^C \rangle \leftarrow$ compute as in (5) and (6);
10 $\mathcal{T}_{\text{rev}} \leftarrow \mathcal{T}_{\text{rev}} \cup \{\langle m_h, u_k^C \rangle\}$;
11 **return** $\mathcal{T}_{\text{rev}}$;

Proof: The proof is given by induction. $\mathcal{T}(0)$ is feasible because robots do not move if $\bar{\Sigma} = \{-1\}^n$ and we assume AV. Thanks to Algorithm 2, precedences can be changed only if the yielding robot i can effectively stop before ℓ_i^C.

Basic step: $\mathcal{T}(0)$ feasible $\Rightarrow \mathcal{T}(T_c)$ feasible. The proof is given by contradiction: note that $\mathcal{T}(T_c)$ is unfeasible iff $\exists \langle m_i, u_j^C \rangle \in \mathcal{T}(T_c)$ such that $m_i(T_c) < \bar{\sigma}_i(0)$ and, given $t_j \in [0, T_c)$, i cannot stop at

$$m_i(T_c) = \begin{cases} \max(\ell_i^C, r_{ij}(T_c)) & \text{if } \sigma_j(t_j) \leq u_j^C \quad \text{(c1)} \\ 1 & \text{otherwise} \quad \text{(c2)} \end{cases}$$

Condition (c1) can happen only in the following cases:

a. $(i <_C j) \in \mathcal{T}(0) \wedge (i <_C j) \in \mathcal{T}(T_c)$, i.e., the previously-decided order is held.

b) $(j <_C i) \in \mathcal{T}(0) \wedge (i <_C j) \in \mathcal{T}(T_c)$, i.e., the previously-decided order is changed.

c) $C \notin \mathcal{C}(0) \wedge C \in \mathcal{C}(T_c)$, i.e., the order is decided for the first time.

However, feasibility holds for all of them: a. Since σ_j is monotone increasing, then according to (6) $\max(\ell_i^C, r_{ij}(0)) \leq \max(\ell_i^C, r_{ij}(T_c))$. But then, $m_i(T_c)$ is feasible since $\bar{\sigma}_i(0) \leq m_i(0) \leq m_i(T_c)$. b. $\langle m_j, u_i^C \rangle \in \mathcal{T}(0)$, and according to AV, $m_j(0) = \ell_j^C$. Moreover, due to (3), $\bar{\sigma}_j(0) \leq \ell_j^C$, so at time T_c robot j can stop. Hence, according to Algorithm 4, $(i <_C j) \in$

$\mathcal{T}(T_c)$ iff $\sigma_i(T_c + \Delta_i^{\text{stop}}) \leq \ell_i^C$, i.e., iff the constraint is feasible. c. Robot j is assigned to a new goal at time T_c, so it is assumed to be not in motion (idle). Then, according to AV, $(j <_C i) \in \mathcal{T}(T_c)$ is feasible, so as in the previous case, $(i <_C j) \in \mathcal{T}(T_c)$ may be decided iff feasibility holds.

In condition (c2), by definition, $\bar{\sigma}_i(t) \in [0, 1]$, and since i can stop at $\bar{\sigma}_i(0) \leq 1$, then $\bar{\sigma}_i(T_c) = 1$ is feasible. Hence, $\nexists \langle m_i, u_j^C \rangle \in \mathcal{T}(T_c)$ that is unfeasible if $\mathcal{T}(0)$ is feasible, proving the preposition in the basic step.

Inductive step: Note that the previous proof holds for every consecutive pair of $\mathcal{T}(t')$ and $\mathcal{T}(t' + T_c)$, if $\mathcal{T}(t')$ is feasible. Then, $\mathcal{T}(0)$ feasible $\Rightarrow \mathcal{T}(T_c)$ feasible $\Rightarrow \cdots \Rightarrow \mathcal{T}(kT_c)$ feasible, $\forall k \in \mathbb{N}^+$. □

As a result, we can prove that:

Theorem 1 (Feasibility): The set $\bar{\Sigma}(t)$ is feasible.

Proof: According to (3), $\mathcal{T}(t)$ feasible implies $\bar{\Sigma}(t)$ feasible. Hence, the proof follows from Lemma 1. □

We use this result to prove the correctness of Algorithm 3:

Theorem 2 (Correctness): The set $\bar{\Sigma}(t)$ is collision free.

Proof: A collision happens iff $\exists t$ such that $R_i(\boldsymbol{p}_i(\sigma_i(t))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) \neq \emptyset$. According to AV, this may happen iff both robots are inside a critical section. Hence, correctness holds if, for every $C \in \mathcal{C}_{ij}(t)$, Algorithm 4 ensures collision-free access to C (c1), and (5) and (6) ensure collision-free progress through C (c2).

Case c1. If no goals are posted at time 0, then robots are all idle and there are no critical sections due to AV. This is a safe starting configuration. Let t_0 be the time such that $g_i \in \mathcal{G}(t_0)$ is assigned to robot i. Let the set of robots for which $\exists C \in \mathcal{C}_{ij}(t_0)$ be partitioned in the sets $\mathcal{D}$ containing the robots which are already driving, and $\mathcal{I}$ containing the robots which are still idle. We can prove that $R_i(\boldsymbol{p}_i(\sigma_i(t_0))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t_0))) = \emptyset$, $\forall j \in \mathcal{I}$ as follows. According to Algorithm 2, $\sigma_i(t_0 + \Delta_i^{\text{stop}}) \leq \ell_i^C$ (robot i can stop) and robot j is idle, so, Algorithm 4 will decide for $\langle \ell_i^C, u_j^C \rangle$ according to (5) and (6). Note that, since i is still not moving, feasibility holds at time t_0. Moreover, (3) will ensure $\bar{\sigma}_i(t) \leq \ell_i^C$ $\forall t$ as long as j remains idle. Finally, due to AV, we can assume that j would not be assigned a goal $\forall t \geq t_0$, preventing j to collide with robot i. We can also prove that $R_i(\boldsymbol{p}_i(\sigma_i(t_0))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t_0))) = \emptyset, \forall j \in \mathcal{D}$, as follows. According to AV, we can assume $R_i(\boldsymbol{p}_i(0)) \cap \mathcal{E}_j(t_0) = \emptyset$. Note that at time t_0 there exists at least a feasible and correct ordering $(i <_C j) \in \mathcal{T}(t_0)$, $\forall C \in \mathcal{C}_{ij}(t_0)$. Also, Algorithm 4 may decide for $(j <_C i) \in \mathcal{T}(t_0)$ iff $\sigma_j(t_0 + \Delta_j^{\text{stop}}) \leq \ell_j^C$. Hence correctness holds: due to feasibility, mutual access to the critical section is collision-free, as either $\bar{\sigma}_i(t_0) \leq \ell_i^C \Rightarrow \sigma_i(t_0 + T_c) \leq \ell_i^C$, or $\bar{\sigma}_j(t_0) \leq \ell_j^C \Rightarrow \sigma_j(t_0 + T_c) \leq \ell_j^C$. Also, at time $t > t_0$ the previously decided ordering can be changed iff the yielding robot can stop, so correctness holds even for $t > t_0$.

Case c2. Let $(i <_C j) \in \mathcal{T}(t)$ be the order of accessing a critical section $C \in \mathcal{C}_{ij}(t)$. Assume that the last received status messages reports that the robots are both inside C. It is easy to show that safety is preserved if the status message of either robot is delayed. Let $\sigma_i(t_i)$ and $\sigma_i(t)$ be the last notified and the current value of σ of robot i respectively (the same for j). Since σ is a monotone increasing function, then for any interval $[\sigma_i', \sigma_i'']$, $\sigma_i'' > \sigma_i'$, if $\mathcal{E}_i^{\{\sigma_i', \sigma_i''\}} \cap \mathcal{E}_j^{\{\sigma_j(t_j), u_j^C\}} = \emptyset$, then $\mathcal{E}_i^{\{\sigma_i', \sigma_i''\}} \cap \mathcal{E}_j^{\{\sigma_j(t), u_j^C\}} = \emptyset$, that is, a delay of the leading robot preserves safety. Moreover, given $\sigma_j(t_j)$, (6) will give the same $r_{ij}(t)$, $\forall \sigma_i : \ell_i^C < \sigma_i < u_i^C$, that is, the same conclusion holds in case of a delay of the waiting robot. □

The proposed algorithm thus maintains a key feature of [18] even in the case of arbitrary (bounded) channel delays:

Corollary 1: For any robot i in the fleet, any realization of $\sigma_i(t)$ that adheres to $\bar{\Sigma}(t)$ is correct. This includes unforeseen stops or changes in velocity due to low-level control and/or safety mechanisms.

VI. The Communication Protocol

In order to minimize the possibility of robots colliding due to packet loss ($\eta \geq 0$), we could use a protocol with message acknowledgment. Although this would ensure the deterministic outcome of message sending, a single non-acknowledged message could invalidate the consistency of the set $\bar{\Sigma}(t)$. Thus, we would have to modify the algorithm in a non-trivial (and potentially computationally expensive) manner in order to properly handle such situations. We adopt here an approach which preserves the relative simplicity of the algorithm, namely, an unreliable (UDP-like) protocol. Specifically, given a model of the channel, this can be used to compute the smallest N such that a burst of N equal messages will result in a probability of successful delivery that is greater than a threshold $\bar{p}$.

Assume that $\langle m_i, u_j^C \rangle \in \mathcal{T}(t - T_c)$, $\langle m_j, u_i^C \rangle \in \mathcal{T}(t)$, and that $\bar{\sigma}_j(t) = m_j$. According to (4), in the worst case,

1) a correct change of priority may happen with probability $\bar{p}^2$ (i.e., both $\bar{\sigma}_i(t)$ and $\bar{\sigma}_j(t)$ are successfully delivered);
2) the probability of maintaining the old constraint is equal to $(1 - \bar{p})^2$ (i.e., both $\bar{\sigma}_i(t)$ and $\bar{\sigma}_j(t)$ are lost);
3) a collision may happen with probability $\bar{p}(1 - \bar{p})$ (i.e., $\bar{\sigma}_i(t)$ is successfully delivered while $\bar{\sigma}_j(t)$ is lost);
4) a temporary starvation may happen with probability $\bar{p}(1 - \bar{p})$ (i.e., $\bar{\sigma}_i(t)$ is lost while $\bar{\sigma}_j(t)$ is successfully delivered).

We consider a constraint to be violated whenever a communication fails to happen as it would without packet loss; hence, the probability of constraint violation is $\bar{p}_u = 1 - \bar{p}^2$. Note that constraint violations do not necessarily lead to collisions.

Bandwidth. As mentioned in Section III-A, congestion can be avoided by explicitly considering the maximum bandwidth required for successful coordination. Specifically, at each time t the required bandwidth is given by:

$$\mathcal{B}(t) = \sum_{i=1}^{n_d(t)} N_i \frac{b_i}{T_i} + N_c(t) N \frac{b_c}{T_c},$$

where N_i is the number of replicas sent by robot i every T_i; b_i is the number of bits of each s_i message; b_c is the number of bits of each $\bar{\sigma}_i$ message; $N_c(t)$ is the number of $\bar{\sigma}_i(t)$ that are updated at time t; and $n_d(t)$ is the number of driving robots at time t. Then, the maximum load of the network can be computed assuming $N_c(t) = n_d(t) = n$ (i.e., all robots are driving and all the critical points are updated). However, the subset of $\bar{\Sigma}(t)$ that is effectively communicated may be smaller (since only the

changes with respect to $\bar{\Sigma}(t - T_c)$ are really informative), and the effective load may be lower.

Let $\gamma \in [0, 1]$ be a desired percentage of bandwidth dedicated to coordination.[5] Assuming (b_i, T_i, N_i) to be equal for each robot, congestion is avoided if

$$n \left(N_i \frac{b_i}{T_i} + N \frac{b_c}{T_c} \right) \leq \gamma \mathcal{B}_M. \tag{7}$$

We can use (7) to relate the number of robots that can be coordinated to the maximum bandwidth $\mathcal{B}_M$, with a probability of constraint violation lower than a the given threshold. The goal is then to define a function to compute the maximum n assuming as parameters the maximum bandwidth $\mathcal{B}_M$, the control periods T_i, T_c, the upper bound of packet loss probability η, and a desired threshold for the probability of constraint violation. For this purpose, we define $\alpha \in \mathbb{R}^+$ such that $T_c = \alpha T_i$; hence, $N_i = \lceil N/\alpha \rceil$ is the minimum number of replicas needed to ensure that at least one s_i will be received from each robot at each T_c with probability almost $\bar{p}$. Then, from (7), we have that:

$$n^{\max} = \left\lfloor \frac{\mathcal{B}_M T_i}{b_i + b_c} \right\rfloor, \text{ hence } n(\alpha, \gamma, N) = \gamma \frac{\alpha}{N} n^{\max}.$$

Decreasing α, T_c, and Δ_i^{stop} allows to react faster to changes, confirming the intuitive fact that a fleet that reacts faster to changes also imposes a higher average load on the network. α can be used by the designer to tune this trade-off as desired, possibly defining an optimization problem.

Controller Synthesis. The following process can be used to design a fleet controller that accounts for a given channel model $(\mathcal{B}_M, \gamma, \eta, T_i, \alpha)$: (i) choose the desired upper-bound $\bar{p}_u$ on the probability of constraint violation; (ii) compute $\bar{p}$ so that $\bar{p}_u \leq 1 - \bar{p}^2$; (iii) compute the minimal number of replicas needed to ensure a probability of receiving at least one replica greater than $\bar{p}$ as $N \geq \lceil \log(1 - \bar{p}) / \log \eta \rceil$; (iv) set $T_c = \alpha T_i$ and $N_i = \lceil N/\alpha \rceil$. Then, the maximal number n of robots which can be coordinated using a UDP-like protocol is given by $n = \gamma \frac{T_c}{N} \frac{\mathcal{B}_M}{b_i + b_c}$.

VII. Experimental Validation

The implementation of Algorithm 3 evaluated here maintains the original good properties of the one proposed in [18], hence fulfills all requirements stated in Table I. The computational overhead of the approach remains unchanged, as the scalability analyses of [18] remain valid. Hence, the simulations presented here focus on communication, and, in particular, aim to confirm the formal properties stated in the previous theoretical discussion. In all experiments, Algorithm 3 was run on an Intel Core i7-5500U CPU @ 2.40GHz × 4 processor. The algorithm is implemented in Java and is available as open source [22].

Setup. The simulator back-end presented in [18] was used for all tests. Uniformly distributed random variables were used for injecting different realizations of $\tau_i^{ch}(t) \in [\tau_{\min}^{ch}, \tau_{\max}^{ch}]$. For simplicity, simulations considered homogeneous robots (although Algorithm 3 is designed for general heterogeneous platforms).

[5] Usually, $\gamma < 1$ to reserve bandwidth for synchronization or QoS.

TABLE II
Simulation Parameters

	Test 1	Test 2
Environment	empty space	Map 1 and 2
Motion planning	off-line	online
Channel model:		
η	0	0.2
$\tau_{\max}^{\text{ch}}$	[0.1, 2] sec	2 sec
$\tau_{\min}^{\text{ch}}$	$\tau_{\max}^{\text{ch}}$	0.01 sec

In all tests: $T_c = 1$s, $T_i = 0.03$ s, $v_i^{\max} = \pm 4$ m/s, $u_i^{\max} = \pm 3$ m/s^2.

Paths were computed using a sampling-based motion planner (RRTConnect). Robot motion synthesis and the conservative model g_i used in Algorithm 2 were both based on a trapezoidal velocity profile with maximum velocity $v_i^{\max}$ and constant acceleration/deceleration $u_i^{\text{acc}} = u_i^{\text{dec}} = u_i^{\max}$. Goals were dispatched asynchronously to robots, requiring them to navigate 10 times from their current location to the opposite side of the environment and back. Deadlocks were handled by re-planning, via a prioritized planning method [23]. Simulation parameters are listed in Table II, and tests were repeated 10 times to obtain statistically significant results. Moments of all tests are shown in the video attachment.

Evaluation metrics. Given the probability $\bar{p}_u$ (and so, the maximum $\bar{p} = \sqrt{1 - \bar{p}_u}$), the probability of the system being in an unsafe state is upper bounded by $\bar{p}(1 - \bar{p})$ (i.e., the message to the newly yielding robot is lost, not the other). This situation involves a pair of events, and is difficult to measure in a distributed setting. Conversely, if a collision happens, then it is certain that the previous pair of events has indeed occurred. Hence, defining as collision rate the ratio between the number of collisions observed and the number of critical sections traversed, we expect the collision rate to be less than $\bar{p}(1 - \bar{p})$. It should be remarked that the formal proof of this is given in the previous sections; this experimental evaluation is intended to support the theoretical findings. Note also that the collision rate tends to $\bar{p}(1 - \bar{p})$ as the number of observations approaches infinity.

A. Test 1: Injecting Channel Delays ($\tau_i^{ch} > 0$, $\eta = 0$)

The goal is to validate the claim that prior knowledge of the channel delay is required to ensure safety. The collision rate obtained using the algorithm proposed in [18] (which is *uninformed* of the channel model) is compared with the one obtained using our implementation of Algorithm 3. To provide the same testing conditions, the simulation uses a fixed set of paths and a constant channel delay ($\tau_i^{ch} = \tau_{\max}^{ch}, \forall i$). Results are shown in Fig. 3, which highlight the unsafety of the uninformed algorithm [18] and validate the safety of the proposed solution.

B. Test 2: Random Paths ($\tau_i^{ch} > 0$, $\eta > 0$)

The goal is to validate safety when the network's non-idealities are modeled as described in Section III-A. In order to stress Algorithm 3 as much as possible, we provoke random delays and packet loss rate within an upper bound, and generate random paths for the robots (so that the geometry of critical sections is unpredictable). We simulate a particularly bad

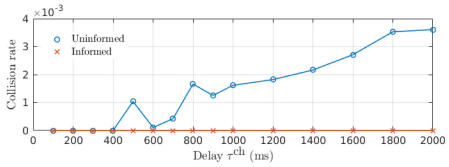


Fig. 3. Test 1: collision rate using the uninformed algorithm [18] and the one proposed in this article.

TABLE III
TEST 2: RESULTS USING MAP 1 (TOP) AND 2 (BOTTOM)

$\bar{p}_u = 2\%$	Upper bound	Measured		
		avg.	max	min
Packet Loss	0.2	0.201	0.209	0.191
Message Loss	1.0e-2	8.9e-3	1.3e-2	4.6e-3
Collision Rate	9.9e-3	3.8e-4	1.3e-3	0
$\bar{p}_u = 10\%$	Upper bound	Measured		
		avg.	max	min
Packet Loss	0.2	0.182	0.192	0.176
Message Loss	5.1e-2	3.6e-2	4.2e-2	3.1e-2
Collision Rate	4.9e-2	3.1e-4	8.5e-4	0

$\bar{p}_u = 2\%$	Upper bound	Measured		
		avg.	max	min
Packet Loss	0.2	0.198	0.209	0.190
Message Loss	1.0e-2	8.3e-3	1.1e-2	6.0e-3
Collision Rate	9.9e-3	4.9e-4	1.1e-3	0
$\bar{p}_u = 10\%$	Upper bound	Measured		
		avg.	max	min
Packet Loss	0.2	0.185	0.202	0.149
Message Loss	5.1e-2	3.5e-2	4.2e-2	2.6e-2
Collision Rate	4.9e-2	4.4e-4	1.2e-3	0

communication channel, with high packet loss and high delays with great variance (which causes the "jerky" motions visible in the video). Simulations are run considering two environments, with a total amount of critical sections analyzed equal to 110693 (Map 1) and 42748 (Map 2). Results are shown in Table III; as expected, the measured collision, packet and message loss rates are smaller then the expected values.

VIII. CONCLUSION

We have presented a centralized coordination algorithm (based on [18]) which does not assume perfect communication, allowing message disorder, bounded message delays, and message loss. The approach enables periodic, heuristically-guided constraint revision while guaranteeing a desired maximum rate of constraint violation. We have shown formally that this probability is zero with no packet loss and arbitrary delay. We have also provided a set of rules to relate communication infrastructure requirements, number of robots, controller parameters, and maximum rate of constraint violation. These rules can be used for fleet controller synthesis, as well as for fleet and network dimensioning. We have validated our formal findings quantitatively via simulations. Preliminary tests with two real robots subject to random communication delays and packet loss confirm[6] the applicability of the method to real-world use-cases. Future work will focus on deployment in environments that are affected by severe network problems (e.g., quarries and underground mines). We will also analyze the problem of deadlock avoidance and resolution.

[6]Video available at https://youtu.be/-rBK_Qgcj28.

REFERENCES

[1] V. M. G. Martínez *et al.*, "Ultra reliable communication for robot mobility enabled by SDN splitting of WiFi functions," in *Proc. IEEE Symp. Comput. Commun.*, 2018, pp. 527–530.

[2] H. Kunsei, K. S. Bialkowski, M. S. Alam, and A. M. Abbosh, "Improved communications in underground mines using reconfigurable antennas," *IEEE Trans. Antennas Propag.*, vol. 66, no. 12, pp. 7505–7510, Dec. 2018.

[3] S. Chen *et al.*, "Vehicle-to-everything (v2x) services supported by LTE-based systems and 5G," *IEEE Commun. Standards Mag.*, vol. 1, no. 2, pp. 70–76, 2017.

[4] P. Park, S. C. Ergen, C. Fischione, C. Lu, and K. H. Johansson, "Wireless network design for control systems: A survey," *IEEE Commun. Surv. Tut.*, vol. 20, no. 2, pp. 978–1013, Apr.–Jun. 2018.

[5] V. Milanes, J. Villagra, J. Godoy, J. Simo, J. Pérez, and E. Onieva, "An intelligent V2I-based traffic management system," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 49–58, Mar. 2012.

[6] E. Nett and S. Schemmer, "Reliable real-time communication in cooperative mobile applications," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 166–180, Feb. 2003.

[7] H. Andreasson *et al.*, "Autonomous transport vehicles: Where we are and what is missing," *IEEE Robot. Autom. Mag.*, vol. 22, no. 1, pp. 64–75, Mar. 2015.

[8] J. Peng and S. Akella, "Coordinating multiple robots with kinodynamic constraints along specified paths," *Int. J. Robot. Res.*, vol. 24, no. 4, pp. 295–310, 2005.

[9] K. E. Bekris, D. K. Grady, M. Moll, and L. E. Kavraki, "Safe distributed motion coordination for second-order systems with different planning cycles," *Int. J. Robot. Res.*, vol. 31, no. 2, pp. 129–150, 2012.

[10] M. Čáp, J. Gregoire, and E. Frazzoli, "Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 5113–5118.

[11] M. P. Fanti, A. M. Mangini, G. Pedroncelli, and W. Ukovich, "A decentralized control strategy for the coordination of AGV systems," *Control Eng. Pract.*, vol. 70, pp. 86–97, 2018.

[12] S. Manca, A. Fagiolini, and L. Pallottino, "Decentralized coordination system for multiple AGVs in a structured environment," *IFAC Proc. Vol.*, vol. 44, no. 1, pp. 6005–6010, 2011.

[13] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljević, and S. Bogdan, "Decentralized control of multi-AGV systems in autonomous warehousing applications," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 4, pp. 1433–1447, Oct. 2016.

[14] D. Bareiss and J. Van den Berg, "Generalized reciprocal collision avoidance," *Int. J. Robot. Res.*, vol. 34, no. 12, pp. 1501–1514, 2015.

[15] M. Kamel, J. Alonso-Mora, R. Siegwart, and J. Nieto, "Robust collision avoidance for multiple micro aerial vehicles using nonlinear model predictive control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 236–243.

[16] N. Smolic-Rocak, S. Bogdan, Z. Kovacic, and T. Petrovic, "Time windows based dynamic routing in multi-AGV systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 1, pp. 151–155, Jan. 2010.

[17] F. Pecora, M. Cirillo, and D. Dimitrov, "On mission-dependent coordination of multiple vehicles under spatial and temporal constraints," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5262–5269.

[18] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov, "A loosely-coupled approach for multi-robot coordination, motion planning and control," in *Proc. 28th Int. Conf. Autom. Planning Scheduling*, 2018, pp. 485–493. [Online]. Available: https://aaai.org/Library/ICAPS/icaps18contents.php

[19] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[20] L. Zhang, H. Gao, and O. Kaynak, "Network-induced constraints in networked control systems—A survey," *IEEE Trans. Ind. Inform.*, vol. 9, no. 1, pp. 403–416, Feb. 2013.

[21] J. Nilsson, "Real-time control systems with delays," Ph.D. dissertation, Dept. Autom. Control, Lund Inst. Technol., Lund, Sweden, 1998.

[22] F. Pecora and A. Mannucci, "An online multi-robot coordination algorithm based on trajectory envelopes (branch UDP)," 2019. [Online]. Available: https://github.com/FedericoPecora/coordination_oru

[23] M. Čáp, P. Novák, A. Kleiner, and M. Seleckỳ, "Prioritized planning algorithms for trajectory coordination of multiple mobile robots," *IEEE Trans. Autom. Sci. Eng.*, vol. 12, no. 3, pp. 835–849, Jul. 2015.

Appendix B On Provably Safe and Live Multi-Robot Coordination with Online Goal Posting (T-RO 2020, under revision)

On Provably Safe and Live Multi-Robot Coordination with Online Goal Posting

Anna Mannucci[1,2], Lucia Pallottino[1], Federico Pecora[2]

***Abstract*—A standing challenge in multi-robot systems is to realise safe and efficient motion planning and coordination methods that are capable of accounting for uncertainties and contingencies. The challenge is rendered harder by the fact that robots may be heterogeneous and that their plans may be posted asynchronously. Most existing approaches require constraints on the infrastructure or unrealistic assumptions on robot models. In this paper, we propose a centralised, loosely-coupled supervisory controller that overcomes these limitations. The approach responds to newly-posed constraints and uncertainties during trajectory execution, ensuring at all times that planned robot trajectories remain kinodynamically feasible, that the fleet is in a safe state, and that there are no deadlocks or livelocks. This is achieved without the need for hand-coded rules, fixed robot priorities, or environment modification. We formally state all relevant properties of robot behaviour in the most general terms possible, without assuming particular robot models or environments, and provide both formal and empirical proof that the proposed fleet control algorithms guarantee safety and liveness.**

***Index Terms*—Multi-robot systems; planning, scheduling and coordination; formal methods in robotics and automation; intelligent and flexible manufacturing.**

I. Introduction

An important challenge in industrial transport automation is to effectively coordinate heterogeneous fleets of robots in dynamic environments while ensuring safety, liveness, and good overall fleet performance. While methods exist for managing fleets of hundreds of robots in static, dedicated environments (e.g., [1]), such methods cease to work if key assumptions are dropped. Specifically, (R1) robots are subject to complex kinodynamic constraints, (R2) tasks (and therefore goals) become known only at run time, (R3) discretising the environment and/or robot paths is too costly or curtails flexibility too much, (R4) robot priorities may change over time, (R5) robot motions, (R6) and communications may be subject to disturbances. Reactive techniques can be used to deal with some of these uncertainties, however, they lack predictability and may lead to deadlocks. Deliberative methods, where collisions and deadlocks are accounted for in motion planning, suffer from severe computational overhead. Experienced by decades of collaborations with industrial partners [2], a centralised supervisory coordinator for possibly heterogeneous robotic platforms was proposed in [3] to account for requirements (R1–R5), and extended in [4] to ensure safety under communication disturbances (R6). The approach is designed for applications in which robots are *loosely-coupled* [5] (i.e., they share the same workspace and are subject to non-cooperative tasks), and assumes decoupled motion planning and control (which holds, e.g., when robots are driven by car-like or differential-drive kinodynamics). Precedence constraints are computed online and revised periodically (while accounting for kinodynamic feasibility) to regulate access to and progress through pairwise contiguous overlapping configurations of the two paths while ensuring safety.

Despite fulfilling all requirements R1–R6, the approach described in [3, 4] does not guarantee liveness, that is, there are conditions under which one or more robots in the fleet do not reach their intended targets. The main contribution of this article is overcoming this limitation. Towards this aim, we proceed in two steps:

1. We formally define the conditions necessary to ensure liveness by design, that is, avoiding blocking, deadlocks and livelocks. This is achieved by generalising the notion of well-formed infrastructure introduced in [6] to heterogeneous fleets, introducing new formal properties of the heuristics used to decide precedences among robots, and imposing conditions on re-planning.
2. We then devise several methods for imposing these conditions, each of which is suitable under a different set of boundary conditions. Specifically, we propose (a) two variants of an online feasibility check that can be used to discard goals and paths that lead to blocking; (b) three extensions of the original algorithm that prevent and/or recover from deadlocks. Each method is validated both formally and empirically, and the trade-off between computational complexity and boundary conditions under which the method is applicable are discussed.

The contributions of this paper are combined into a framework for integrated motion planning, coordination and control with the following features:

- the framework ensures provable safe and live coordination of heterogeneous robotic platforms subject to kinodynamic constraints with communication disturbances;
- the framework is general with respect to motion planners and controllers;
- goals can be posted asynchronously to robots when they become known;
- precedences can be decided online and revised according to *any* user-defined ordering heuristic;

*This work is funded by the EU's Horizon 2020 research and innovation program under agreement no. 732737 (ILIAD), the Swedish Knowledge Foundation (KKS) under the Semantic Robots research profile, and Vinnova under project AutoHauler. *(Corresponding author: Anna Mannucci.)* Email: anna.mannucci@oru.se, lucia.pallottino@unipi.it, federico.pecora@oru.se.

[1]Research Center "E. Piaggio", University of Pisa, Italy and Dipartimento di Ingegneria dell'Informazione, University of Pisa, Italy.

[2]Center for Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden.

- safety and liveness are guaranteed under bounded spatial deviations from nominal paths and any velocity profile that satisfies the precedences.

Experiments confirm the effectiveness of the approach for realistically sized fleets of robots (≤ 40 tested) with reasonable computational complexity.

The rest of the paper is organised as follows. Relevant state of the art is presented in Section II. The problem tackled in this paper is formally stated in Section III. Section IV recalls necessary notation and concepts from in [3, 4] and grounds the problem into this framework. Sections V, VI and VII formally analyse the factors that may prevent the supervisory controller of [3, 4] from ensuring liveness, proposing strategies for overcoming this limitation while ensuring safety. Simulations and conclusion are shown in Sections VIII and IX respectively.

II. State of the art

The literature addressing multi-robot coordination is vast[1], spanning multi-arm coordination [15], air-traffic control [16–18], traffic management [19], mobile robot coordination in warehouses [1, 20, 21], and many other application scenarios. The problem has been investigated from a *reactive* perspective, which is mostly concerned with the issue of avoiding collisions among robots while they execute previously planned motions; or from a *deliberative* perspective, in which motion planning itself accounts for the presence of multiple robots. Some methods are general, whereas others have been designed for particular applications and are difficult to apply in other contexts. For example, obstacles other than the robots themselves are typically ignored in air-traffic control [16–18], where the problem is usually approached from a control-theoretic point of view.

Reactive techniques [22, 23] are appealing due to their low computational overhead; however, many make strong assumptions, such as homogeneous fleets of holonomic robots, the absence of dynamic obstacles, or simplified kinodynamic constraints. Moreover, due to their locality, these methods cannot ensure that robots will eventually reach their goals (liveness) and their extension to more general settings [7] does not guarantee that collisions never happen.

Deliberative approaches leverage longer planning horizon to plan trajectories that are safe and deadlock-free by construction [24]. These methods are usually not specific to the robot model and extend to higher degrees of freedom. They can be either *coupled* or *decoupled*. The former search for a solution in the joint configuration space of all robots in the fleet. They ensure completeness (and sometimes optimality), but at the price of exponential computation time[2] in the number of robots. This issue is partially solved by techniques such as M* [26], which first plans for each robot separately, and only couples sets of robots after they have been found to interact (thus minimizing the dimensionality of the search space). However, complexity is still exponential with respect to the number of robots involved in each sub-conflict.

Decoupled approaches are usually incomplete, but an order of magnitude faster than coupled ones, as each robot computes its own path, and conflicts are solved a posteriori. In *prioritised planning* [27], high priority robots are considered as moving obstacles by lower priority ones and collision-free trajectories are computed using techniques for motion planning in dynamic environments, e.g., [28]. As observed in [29, 30], the choice of (static) priorities has a great impact on whether a solution is found and on its quality. Prioritised planning is revisited in [6], where the concept of *well-formed infrastructure* is proposed to guarantee that a feasible trajectory is always computable (thus, the completeness of multi-robot motion planning depends on that of the decoupled motion planners). The technique is extended in [9] to allow goals to be posted online and trajectories to be computed in a decentralised fashion with a token-based approach. However, both methods assume holonomic disc-shaped fleets, and require robots to be synchronised on a common time.

Another decoupled approach *tunes velocity* along precomputed paths [31]. The approach is extended in [32] to obtain optimal collision-free trajectories for generic robot models subject to kinodynamic constraints. In particular, collision avoidance constraints for pairs of robots in a common collision zone are expressed in a Mixed Integer Nonlinear Program formulation of the problem. However, complexity remains exponential in the number of collision zones. Similarly, [11] leverages the notion of least commitment to obtain easily revisable, deadlock- and collision-free trajectories for fleets of possibly heterogeneous robots subject to kinodynamic constraints. While this approach also requires exponential time, it exploits the concept of *spatial envelope*, which generalises the concept of path to account for spatial uncertainties in tracking. This is a key tool of our line of research [3, 4, 11, 33, 34], and is also used in this article.

To overcome the computational complexity of [11, 32], the approach in [35] proposes a decentralised, prioritised, receding horizon version of the velocity tuning method to manage possibly heterogeneous non-holonomic vehicles at traffic intersections. Priorities are used to define the sequence at which each robot solves its own optimisation problem (passing before all or after all the other robots which have already decided). Although this leads to sub-optimal solutions, they can be computed in polynomial time. Decision orders may be revised sequentially (through agreement), and a model-based heuristic that accounts for robot dynamics is proposed to enforce feasibility. However, the paper does not consider uncertainties in trajectory execution (even though the receding horizon character of the approach may enable this). Also, as other solutions conceived for traffic management [36], the approach was tested only in simulation and for simple intersections.

The main issue of trajectory-based coordination methods is that they rely on synchronised clocks and accurate execution of trajectories both in space and time to ensure safety and liveness [12], and hence are unsuitable whenever this assumption does not hold (e.g., when actual speeds cannot be accurately predicted). The *coordination space* [15] has been shown to

[1]The scope of this section is not to give a comprehensive overview of multi-robot coordination methods, but to guide the reader in understanding the design choices behind our coordination method. The most recent survey (with references up to 2013) can be found in [14].

[2]Deciding if the multi-robot path finding problem is feasible is NP-hard for disc-shaped robots in environments admitting polygonal obstacles [8], while it is PSPACE-hard for rectangular robots in empty environments [25].

Method		Type	Time complexity	Safe?	Live?	R1[a]	R2	R4	R5-R6[b]
Reactive:	[7]	D	$\mathsf{poly}(n + \lvert\mathcal{O}\rvert)$	×	×	✓	✓	✓	✓
Coupled:	[8]	C	$2^{\mathsf{poly}(n)}$	✓	✓	✓	×	✓	×
Decoupled:			**(only of coordination)**						
Prio. planning	[6, 9]	[C/D, D]	$O(n)$	✓	✓	×	×	[×, ✓]	×
Veloc. tuning	[10, 11]	[C, C]	[$2^{\mathsf{poly}(\lvert\mathcal{C}\rvert)}$, $O(\lvert\mathcal{C}\rvert)$ / $O(2^{\lvert\mathcal{C}\rvert})$ opt.]	✓	✓	✓	[×, ✓]	[×, ✓]	partially
Coord. space	[12, 13]	[D, D]	$O(\lvert\mathcal{C}_i\rvert)$	[✓, ×]	[✓, ×]	×	[✓, ×]	[×, ✓]	✓
Priority-based	[3, 4]	[C, C]	$O(\lvert\mathcal{C}\rvert)$	✓	×	✓	✓	✓	✓
	This paper	C	$O(\lvert\mathcal{C}\rvert)$ / $O(\lvert\mathcal{C}\rvert n^2)$ / $O(\lvert\mathcal{C}\rvert 2^{n \log n})$	✓	✓	✓	✓	✓	✓

TABLE I: State of the art: a summary of selected strategies.
Legend: **D** decentralized or distributed; **C** centralized; $\mathcal{O}$: set of obstacles other then robots; n: number of robots; $\mathcal{C}$, $\mathcal{C}_i$: set of pairwise critical sections (all, involving robot i) – see Section IV; **a**: heterogeneous fleets of robots subject to kinodynamic constraints (for which it is possible to decouple motion planning from control); **b**: limited to temporal uncertainties in trajectory execution. Note that only [3, 4, 7] are robust to clock de-synchronization.

be a useful tool to overcome this limitation [12, 13, 15, 37]. Combined with *precedence orders*, this tool allows to design simple control laws to safely schedule the motions of pairs of robots while dealing with temporal uncertainties in trajectory execution. Given a set of intersecting paths, there is a finite set of orders of traversal of their intersections [35] which avoids conflicts. Each such ordering identifies a specific homotopic class [38] of conflict-free trajectories. To also enforce liveness, this information can be encoded in a directed graph (the *priority graph* [37]) which allows to avoid deadlocks by preventing particular cycles [3, 37] corresponding to circular waits [39]. Thanks to its generality, the coordination space has been successfully applied to coordinate robot manipulators [15] and vehicles at traffic intersections [37], and to handle uncertainties in following pre-computed conflict-free trajectories for holonomic disc-shaped robots [12, 13] (without considering kinodynamic constraints). To ensure both safety and liveness, the approach of [12] forces each yielding robot to always wait for the leading one before accessing a collision zone, even if the leading robot was delayed (thus respecting the previously-decided static priorities, i.e., the delayed trajectories are homotopic with the original ones). The extension given in [13] allows priorities to be swapped. However, formal proofs with dynamic priorities are not given in [13]. Conversely, such proofs are given in this article, where spatial envelopes are used to deal with uncertainties in the spatial component of trajectories, and deadlock-free priorities are determined dynamically ensuring both safe and live progress of the fleet through intersecting areas.

Table I summarises the main features of the reactive and deliberative approaches found in the literature. We partition the latter category into coupled and decoupled due to the complexity/feature trade-off. The approaches summarised in the table are those that maximize achievement of the requirements (R1–R6) outlined in Section I.

III. Problem definition

Multi-robot fleet. We describe our multi-robot system with a set $\mathcal{R} = \{1, \dots, n\}$ of (possibly heterogeneous) robots sharing an environment $\mathcal{W} \subset \mathbb{R}^3$ with obstacles $\mathcal{O} \subset \mathcal{W}$. Each robot i is identified by a tuple $r_i = \langle \mathcal{Q}_i, R_i, f_i, g_i, s_i \rangle$[3], where: $\mathcal{Q}_i$ is the robot's configuration space; $R_i(q_i)$ a geometry describing the space occupied by the robot when placed in configuration $q_i \in \mathcal{Q}_i$; $f_i(q_i, \dot{q}_i) \leq 0$ is a set of kinematic constraints on the robot's motion; $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i^{\mathrm{acc}}, u_i^{\mathrm{dec}}, t)$ is a model of the robot's dynamics, with maximum acceleration/deceleration $u_i^{\mathrm{acc/dec}}$; and s_i is the robot's status, containing information about its current mission. We assume R_i to be independent from $(\dot{q}_i, \ddot{q}_i)$. Also, let $\mathcal{Q}_i^{\mathrm{free}} = \{q_i \in \mathcal{Q}_i : R_i(q_i) \cap \mathcal{O} = \emptyset\}$ be the set of obstacle-free configurations of robot i.

[3]The notation $(\cdot)_i$ is used in the following to indicate that the variable $(\cdot)$ is related to the robot i.

Goals. When idle, a robot i may be assigned to a *non-cooperative, asynchronously posted* task which involves moving from its starting configuration $q_i^{\mathrm{s}} \in \mathcal{Q}_i^{\mathrm{free}}$ to a *goal* configuration $q_i^{\mathrm{g}} \in \mathcal{Q}_i^{\mathrm{free}}$ and stay there. This concept is general and may be easily extended to account for more complex tasks including non-cooperative operations (e.g., pick-and-place) or interim configurations to be reached.

Paths and trajectories. Given a pair $(q_i^{\mathrm{s}}, q_i^{\mathrm{g}}) \in \mathcal{Q}_i^{\mathrm{free}} \times \mathcal{Q}_i^{\mathrm{free}}$, a path $\boldsymbol{p}_i : [0, 1] \to \mathcal{Q}_i^{\mathrm{free}}$ (parametrized using the arc length $\sigma \in [0, 1]$) is a sequence of $q_i \in \mathcal{Q}_i^{\mathrm{free}}$ so that $\boldsymbol{p}_i(0) = q_i^{\mathrm{s}}$, $\boldsymbol{p}_i(1) = q_i^{\mathrm{g}}$, satisfying the set of kinematic constraints $f_i(q_i, \dot{q}_i) \leq 0$ (see Fig. 1.a). Idle robots are associated with a path of length one corresponding to their current configuration. For each $\boldsymbol{p}_i$, the trajectory planning problem is the problem of synthesizing an executable temporal profile $\sigma_i(t)$ typically considering the robot's kinodynamic constraints.

Problem 1 (Coordination Problem)**.** *Given $\mathcal{W}$, $\mathcal{O}$, $\{r_i\}_{i=1}^n$, asynchronously posted $\{q_i^{\mathrm{g}}(t)\}_{i=1}^n$, the coordination problem is the problem of synthesizing and revising during time a set of spatio-temporal constraints on robot trajectories $\bigcup_{i \in \mathcal{R}} \{\boldsymbol{p}_i, \sigma_i(t)\}$ so that both of the following two properties are satisfied:*

P1. Safety. *Robots never collide:*

$$\forall (i, j \neq i) \in \mathcal{R}^2, \forall t \; R_i(\boldsymbol{p}_i(\sigma_i(t))) \cap R_j(\boldsymbol{p}_j(\sigma_j(t))) = \emptyset.$$

P2. Liveness. *All robots eventually reach their destination:*

$$\forall i \in \mathcal{R}, \; \exists t < \infty \text{ such that } \sigma_i(t) = 1.$$

IV. A heuristic, priority-based coordinator

To solve Problem 1 while accounting for online posted constraints (either due to contingencies or new posted goals), our approach relies on a centralized decoupled priority-based supervisory coordinator [3, 4] whose main body, running at each discrete time $\mathrm{k} \in \mathbb{N}$, is listed in Algorithm 1. From now on, we use discrete time k to indicate any time $t \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$, where T_c is the period of the coordination

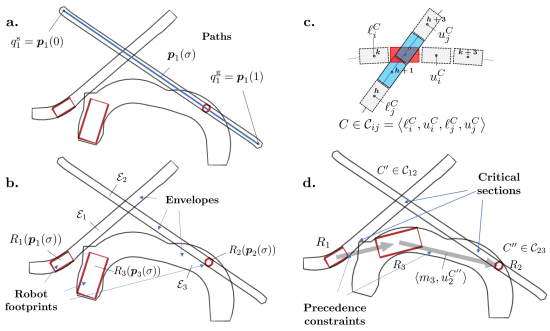


Fig. 1: Main concepts of the approach in [3, 4].

loop. The approach uses *precedence constraints* to regulate access to, and progress through, pairwise overlapping portions of *spatial envelopes*, called *critical sections*.

Spatial envelopes. For each path $\boldsymbol{p}_i$, a *spatial envelope* $\mathcal{E}_i$ is a set of constraints such that $\bigcup_{\sigma\in[0,1]} R_i(\boldsymbol{p}_i(\sigma)) \subseteq \mathcal{E}_i$ [40]. If the equality holds (which we assume for simplicity from now on), a spatial envelope is the sweep of the robot's geometry along its path (Fig. 1.b). Henceforth, given $S \subseteq [0,1]$, let $\mathcal{E}_i^S = \bigcup_{\sigma\in S} R_i(\boldsymbol{p}_i(\sigma))$. Also, we say that $\boldsymbol{p}_i$ is W-avoiding, where $W \subset \mathcal{W}$, if $\mathcal{E}_i \cap W = \emptyset$. Note that each path $\boldsymbol{p}_i$ is $\mathcal{O}$-avoiding by definition. Similarly in configuration space, $\boldsymbol{p}_i$ is Q-avoiding if $Q \subset \bigcup_{i\in\mathcal{R}} \mathcal{Q}_i \implies \mathcal{E}_i \cap \bigcup_{q_i\in Q} R_i(q_i) = \emptyset$.

Critical sections. A critical section C is a tuple $\langle \ell_i^C, u_i^C, \ell_j^C, u_j^C\rangle$ of continuous intervals of the arc lengths σ_i and σ_j such that for every $\sigma_i \in (\ell_i^C, u_i^C)$, there exists $\sigma_j \in (\ell_j^C, u_j^C)$ such that $R_i(\boldsymbol{p}_i(\sigma_i)) \cap R_j(\boldsymbol{p}_j(\sigma_j)) \neq \emptyset$, and vice versa. Specifically, ℓ_i^C is the highest value of σ_i before robot i enters C, and u_i^C is the lowest value of σ_i after robot i exits C (analogously for j) — see Fig. 1.c for examples. Let $\mathcal{C}_{ij}$ be the set of all critical sections pertaining to the two robots i and j. We say that $C \in \mathcal{C}_{ij}$ is *active* while $\sigma_i(t) < u_i^C \wedge \sigma_j(t) < u_j^C$, that is, when neither robot has exited C. Given the set of robot paths $\mathcal{P} = \bigcup_{i\in\mathcal{R}} \boldsymbol{p}_i$, let $\mathcal{C} \subseteq \bigcup_{(i,j>i)\in\mathcal{R}^2} \mathcal{C}_{ij}$ be the set of all active critical sections. Henceforth, let $\mathcal{P}(\mathrm{k})$ and $\mathcal{C}(\mathrm{k})$ be the values of these sets after all paths have been computed, that is, after executing line 4 of Algorithm 1.

Precedence constraints. A precedence constraint $\langle m_i, u_j^C\rangle$ is a constraint on the temporal evolution of $\sigma_i(t)$ such that $\ell_i^C \leq m_i < u_i^C$ and $\sigma_j(t) < u_j^C \implies \sigma_i(t) \leq m_i$, that is, robot i cannot navigate beyond $\boldsymbol{p}_i(m_i)$ along its path until robot j has exited the critical section C (see Fig. 1.d). In other words, a precedence constraint defines *which* robot should yield, *where*, and *until when*. Let $\mathcal{T}(\mathrm{k})$ be the set of precedence constraints regulating access to the set of critical section $\mathcal{C}(\mathrm{k})$. If $\forall C \in \mathcal{C}(\mathrm{k})$, either $\langle m_i, u_j^C\rangle \in \mathcal{T}(\mathrm{k})$ or $\langle m_j, u_i^C\rangle \in \mathcal{T}(\mathrm{k})$, then $\mathcal{T}(\mathrm{k})$ is a *complete ordering* of robots through $\mathcal{C}(\mathrm{k})$, which ensures that **P1** holds. Given the set of paths $\mathcal{P}(\mathrm{k})$, a complete ordering $\mathcal{T}(\mathrm{k})$ defines in fact the selected homotopic class of collision-free trajectories [41, 42]. In particular, for each $C \in \mathcal{C}(\mathrm{k})$, the precedence constraint $\langle m_i, u_j^C\rangle \in \mathcal{T}(\mathrm{k})$ is computed as

$$m_i(\mathrm{k}) = \begin{cases} \max\left\{\ell_i^C, r_{ij}(\mathrm{k})\right\} & \text{if } \sigma_j \leq u_j^C \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

$$r_{ij}(\mathrm{k}) = \sup_{\sigma\in[\sigma_i(t_i),u_i^C]} \left\{ \mathcal{E}_i^{[\sigma_i(t_i),\sigma]} \cap \mathcal{E}_j^{[\sigma_j(t_j),u_j^C]} = \emptyset \right\},$$

where $\sigma_i(t_i)$ and $\sigma_j(t_j)$ are the last known positions of robot i and robot j, received by the coordinator at time t_i, $t_j \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$, respectively. Note that m_i is updated at each control period (line 5 in Algorithm 1), allowing robots to "follow each other" through critical sections.

Critical points. Let $\Psi_i = \{m_i \mid \exists j : \langle m_i, u_j^C\rangle \in \mathcal{T}(\mathrm{k})\}$ be the set of all the arc lengths at which robot i may be required to yield. We define the *critical point* $\bar{\sigma}_i(\mathrm{k})$ of robot i at discrete time k as the value of σ corresponding to the last reachable configuration along $\boldsymbol{p}_i$ which adheres to the set of constraints $\mathcal{T}(\mathrm{k})$, i.e.,

$$\bar{\sigma}_i(\mathrm{k}) = \begin{cases} \underset{m_i\in\Psi_i(t)}{\arg\min}\, m_i & \text{if } \Psi_i \neq \emptyset, \\ 1 & \text{otherwise.} \end{cases} \tag{2}$$

Hence, Problem 1 is equivalent to that of finding an appropriate $\mathcal{P}(\mathrm{k})$, $\mathcal{C}(\mathrm{k})$, $\mathcal{T}(\mathrm{k})$ and $\bar{\Sigma}(\mathrm{k}) = \bigcup_{i\in\mathcal{R}} \bar{\sigma}_i(\mathrm{k})$, and revising these sets appropriately at each control period.

Algorithm 1: Coordination at time k.

1 get last received status messages $\bigcup_{i\in\mathcal{R}} s_i(t_i)$;
2 **if** *new goals have been posted* **then**
3 update the set of paths $\mathcal{P}$ (using appropriate planners[4]);
4 update the set $\mathcal{C}$ of critical sections;
5 revise the set $\mathcal{T}(\mathrm{k})$ of precedence constraints;
6 compute the set of critical points $\bar{\Sigma}(\mathrm{k})$;
7 communicate changed critical points;
8 sleep until control period T_c has elapsed;

Heuristic scheduling. A key feature of our method is that the selected collision-free homotopic class of trajectories may change online. In other words, precedence orders may be dynamically updated according to *any* user-defined heuristic-based ordering function $\mathrm{h}(t)$ [3] (let $i \prec_{\mathrm{h}(t)} j$ indicate that i yields for j according to $\mathrm{h}(t)$ at a given $C \in \mathcal{C}_{ij}$), while guaranteeing that safety is preserved. This is done in Algorithm 2, which illustrates the functioning of line 5 of Algorithm 1. The algorithm is used to filter changes of precedence orders that may results in a collision. For this purpose, we define the *lookahead* Δ_i^{stop} as the interval of time such that a command to yield sent by the coordinator at time t will make robot i stop at most at time $t+\Delta_i^{\mathrm{stop}}$. A conservative estimate of this value allows to ensure safety in the presence of bounded uncertainties in the robot's dynamics [3], and in the communication network [4]. Then, at discrete time $\mathrm{k} \geq 1$ a constraint $\langle m_i, u_j^C\rangle \in \mathcal{T}(\mathrm{k}-1)$ can be replaced with $\langle m_j, u_i^C\rangle \in \mathcal{T}(\mathrm{k})$ (reversed) only if $\sigma_j(t_j + \Delta_j^{\mathrm{stop}}) \leq \ell_j^C$,

[4]Note that paths may be asynchronously computed by private planners running in parallel. This speeds up the computation, allows to better explore the heterogeneity of the fleet and allows planning parameters (such as kinematic constraints, gains, type of planner used, etc.) to be private to robots.

$t_j \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$ (i.e., the new yielding robot has not already entered the critical section and can stop before entering it if asked to). This feasibility check is implemented via a conservative forward propagation of the two robots' dynamics (see the *canStop* function of [4] for details).

Algorithm 2: The revise function at time k (implements line 5 of Alg. 1).

Input: $\mathcal{P}$ current set of paths; $\mathcal{C}$ (possibly empty) set of pairwise critical sections; $\mathcal{T}(\mathrm{k}-1)$ (possibly empty) previous set of precedence constraints; $\bar{\Sigma}(\mathrm{k}-1)$ previous set of critical points; $s_i(t_i)$ last received robot status message, including $\sigma_i(t_i)$; h heuristic-based ordering function.

Output: $\mathcal{T}^{\mathrm{rev}}$ set of revised precedence constraints.

1 $\mathcal{T}_{\mathrm{rev}} \leftarrow \emptyset$;
2 **for** $\mathcal{C}_{ij} \in \mathcal{C}, C \in \mathcal{C}_{ij}$ **do**
3 **if** $\sigma_i(t_i) < u_i^C \wedge \sigma_j(t_j) < u_j^C$ **then**
4 **if** $\sigma_i(t_i + \Delta_i^{\mathrm{stop}}) \leq \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\mathrm{stop}}) \leq \ell_j^C$ **then**
5 $(h,k) \leftarrow$ get ordering according to h;
6 **else if** $\sigma_i(t_i + \Delta_i^{\mathrm{stop}}) > \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\mathrm{stop}}) \leq \ell_j^C$ **then** $(h,k) \leftarrow (j,i)$;
7 **else if** $\sigma_i(t_i + \Delta_i^{\mathrm{stop}}) \leq \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\mathrm{stop}}) > \ell_j^C$ **then** $(h,k) \leftarrow (i,j)$;
8 **else** $(h,k) \leftarrow$ get previous ordering from $\mathcal{T}$;
9 $\langle m_h, u_k^C \rangle \leftarrow$ compute as in (1);
10 $\mathcal{T}_{\mathrm{rev}} \leftarrow \mathcal{T}_{\mathrm{rev}} \cup \{\langle m_h, u_k^C \rangle\}$;
11 **return** $\mathcal{T}^{\mathrm{rev}}$;

A. Communication requirements

We assume a duplex point-to-point communication through a dedicated wireless network (subject to bounded delays and/or message loss) between a central unit (coordinator) and each robot in the fleet. We require each robot to send an update on its status s_i sampled within its control period T_i (clock-driven system) every T_i seconds (robots may have different control periods). The status message contains the tuple $(q_i(t_i), \dot{q}_i(t_i), \ddot{q}_i(t_i))$, as well as the last critical point $\bar{\sigma}_i$ received by the robot. Also, we assume the coordinator receives at least one update from each agent every T_c seconds, and that $T_c \geq \max_{i \in \mathcal{R}} T_i$. Robots are not required to be synchronised on a common clock (so communication may be asynchronous).

Algorithm 1 is by construction robust to communication failures such that all the critical points in the last $\bar{\Sigma}(\mathrm{k})$ before the failure are either successfully communicated or lost. In order to preserve safety also in case of asymmetric disturbances (delays, or some messages are lost and not others), we assume the use of a UDP-like protocol as described in [4], which ensures safety by relating number of re-transmissions to the properties of the communication channel.

B. Assumptions for safety.

As in [4], we introduce the following set of assumptions:

A1. Paths do not start or end in *active* critical sections.
A2. Robots are idle at time $\mathrm{k} = 0$, placed in a safe starting configuration. Also, robots are not in motion when idle.
A3. Robots always stay within their envelope (that is $q_i(t) \in \bigcup_{\sigma \in [0,1]} \boldsymbol{p}_i(\sigma)$).
A4. Robots do not back up along their paths.
A5. Each lookahead Δ_i^{stop} is a conservative estimate of the time required by robot i to yield if required to. This entails that: (i) $\Delta_i^{\mathrm{stop}} \geq T_c$, as in Algorithm 1 sampling occur at the beginning (line 1), while communication occur at the end (line 7); (ii) $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i^{\mathrm{acc}}, u_i^{\mathrm{dec}}, t)$ is a conservative model of robot i's dynamic; (iii) a conservative model of the communication network is known, e.g., finite upper bounds of transmission delay and the packet loss probability $(\tau_{\max}^{\mathrm{ch}}, \eta)$.

Also, as in [12], we assume

A6 Prohibitive disturbances, i.e., uncontrollable events requiring human intervention in order to recover from them[4], not to happen.

Note that A1 and A6 are standard assumptions in the literature, since they ensure that a solution of the coordination problem exists.

C. Formal properties

Our aim is to formally characterise Algorithms 1 and 2 so that both **P1** (safety) and **P2** (liveness) are jointly satisfied.

Ensuring P1. As proved in [4], it can be shown that

Theorem 1 (Sufficient conditions for **P1**)**.** *Under A1–A6, Algorithm 2 ensures* **P1** *to be verified for any heuristic* h *and for any set* $\mathcal{E} = \bigcup_{i \in \mathcal{R}} \mathcal{E}_i$.

Ensuring P2. As reported in the literature, several factors may prevent robots from reaching their destinations:

F1. *Blocking:* a robot should stop its mission for an unbounded time because another robot has parked along its path [6];
F2. *Deadlocks:* there is a subset of robots such that each waits for another one to proceed along its path [42];
F3. *Livelocks:* similar to deadlocks, but where robot configurations constantly change, none progressing [18].

However, Algorithms 1 and 2 fail to consider these factors, and hence **P2** is not guaranteed. To overcome this, in the remainder of this article we will alter lines 2–5 of Algorithm 1.

In order to ensure that **P1** and **P2** can be verified formally, we map these to properties of the spatial and the temporal component of the problem i.e., the sets of trajectory envelopes $\mathcal{E}(\mathrm{k}) = \bigcup_{j \in \mathcal{R}} \mathcal{E}_j(\mathrm{k})$ and of precedence constraints $\mathcal{T}(\mathrm{k})$.

Definition 1 (Admissibility of $\mathcal{E}(\mathrm{k})$)**.** $\mathcal{E}(\mathrm{k})$ *is admissible iff there exists* $\mathcal{T}(\mathrm{k})$ *s.t. both* **P1** *and* **P2** *hold (see Fig. 2).*

Note that while $\mathcal{E}(\mathrm{k})$ does not change, any temporal evolution of σ_i, $i \in \mathcal{R}$ satisfying $\mathcal{T}(\mathrm{k})$ and computed according to (1) belongs to the same homotopic class, hence it maintains the same properties. Therefore, starting from $\mathcal{E}(0)$, which is admissible thanks to A1–A3, our aim is to define *how* to ensure that $\mathcal{E}(\mathrm{k})$ remains admissible for all k.

[4] e.g., an unpredictable obstruction along the path making the current goal unreachable (there does not exist an executable path leading to it), a failure of one or more robots, of the overall communication network or the coordinator, or a malicious dynamic obstacle.

Fig. 2: Admissible (left) and not admissible (right) envelopes.

A possible approach is to map the approach of [9] (trajectory planning with online goal posting) into our priority-based framework, that is, whenever a new goal q_i^g is posted at time k, the new trajectory is searched for in $\mathcal{Q}_i^{\text{free}} \times \mathbb{R}$, considering all the possible trajectories of other robots defined by $\mathcal{T}(\mathrm{k})$ as dynamic obstacles. If a solution is found, then both $\mathcal{E}_i$ and $\mathcal{T}$ satisfying Definition 1 can be obtained as a byproduct. Otherwise, q_i^g is delayed or rejected. The approach is complete (if complete trajectory planners are used) [9] and, under A1–A6, it guarantees **P1** and **P2** by construction. However, trajectory planning may require exponential computation time in the worst case (and hence may not be suitable for revising priorities dynamically). Also, the planning phase must account for bounded delays in trajectory execution in order to guarantee safety. Hence, we investigate a different strategy.

Note that to preserve admissibility, it is sufficient to check the validity of the property only when the pair $(\mathcal{E}, \mathcal{T})$ is updated. Specifically, $\mathcal{E}(\mathrm{k})$ may change only when goals are assigned (line 2, Algorithm 1), or if we allow paths to be re-planned. In both cases, we should prevent updates which may lead to blocking (F1). Precedences in $\mathcal{T}(\mathrm{k})$ may change due to new critical sections (new paths), or when precedences are revised (line 5, Algorithm 1). Assuming $\mathcal{E}(\mathrm{k}-1)$ to be admissible and $\mathcal{E}(\mathrm{k}) = \mathcal{E}(\mathrm{k}-1)$, we can avoid/filter out precedences that may lead to deadlocks (F2).

We therefore proceed in three steps (see Table II), progressively defining the conditions and methods needed to *avoid*, *prevent*, or *repair* blocking, deadlocks and livelocks. By avoidance, we mean ensuring boundary conditions that completely avoid the possibility of blocking/deadlock/livelock. When such conditions cannot be guaranteed, we resort to prevention, that is, algorithms that actively prevent blocking/deadlock/livelock from happening. Prevention strategies may require knowledge of, and affect, the motions of all robots in the fleet. If such access cannot be guaranteed, repair actions may be warranted. We hence distinguish global prevention from local prevention, where the former is able to guarantee liveness, while the latter may require a local repair strategy to re-establish admissibility. All of the prevention/repair strategies preserve the key features of the online setting, that is, goals become known only at runtime, and precedences can be changed online.

In Section V, we analyse the conditions on the set of active goals $\mathcal{G} = \bigcup_{i\in\mathcal{R}} \boldsymbol{p}_i(1)$ under which blocking (F1) is avoided or can be globally prevented. Avoidance is achieved by using a generalisation of the well-formed infrastructure concept; prevention is achieved via an admissibility check for goals.

In Section VI we address F2, characterising deadlocks and nonlive states (i.e., states which may lead to deadlocks [43]), and relating them to the set $\mathcal{T}$. We analyse the conditions under which the heuristic function h avoids deadlocks and we formalise strategies for deadlock prevention/repair.

In Section VII, we characterise livelocks (F3), defining the conditions under which it is possible to avoid them.

V. Blocking

As recognised in [6, 9], the concept of blocking is strictly related to the exclusive spatio-temporal ownership of destinations. In this section we define a set of sufficient conditions that avoid/prevent robots from blocking each other. Conversely, necessary conditions cannot be stated since we consider asynchronously posted goals with instantaneous assignment (i.e., with no planning for future allocations).

A. Avoidance

Let $\mathbb{G}_i \subseteq \mathcal{Q}_i^{\text{free}}$ be the set of all the possible end-points of robot i, $\mathbb{G} = \bigcup_{i\in\mathcal{R}} \mathbb{G}_i$, and $\mathbb{G}^{j\neq i}(q_i^{\mathrm{s}}) = \{q_j^{\mathrm{g}} \in \mathbb{G}_j : R_i(q_i^{\mathrm{g}}) \cap R_j(q_j^{\mathrm{g}}) = \emptyset\}$. The absence of blocking (F1) can be ensured by design, requiring the environment $\mathcal{W}$ and the set $\mathbb{G}$ to form a *well-formed infrastructure*, which we define[5] as follows:

Definition 2 (Well-formed infrastructure)**.** *The pair $(\mathcal{W}, \mathbb{G})$ is a well-formed infrastructure if $\forall i \in \mathcal{R}$, $\forall (q_{i_j}, q_{i_h}) \in \mathbb{G}_i \times \mathbb{G}_i$ there exists a $\mathbb{G}^{j\neq i}(q_{i_h})$-avoiding path from q_{i_j} to q_{i_h} that lies in $\mathcal{Q}_i^{\text{free}}$ and adheres to $f_i(q_i, \dot{q}_i) \leq 0$.*

Theorem 2 (Admissibility Check in Well-Formed Infrastructure)**.** *Assume $(\mathcal{W}, \mathbb{G})$ verifies Definition 2 and A2–A6. In particular, assume that at time 0 each robot i starts from $q_i^{\mathrm{s}}(0) \in \mathbb{G}_i$. At time t, a new path $\boldsymbol{p}_i$ to $q_i^{\mathrm{g}} \in \mathbb{G}_i$ can be accepted while ensuring* **P1** *and $\neg$(**F1**) if $\boldsymbol{p}_i$ is $\mathbb{G}^{j\neq i}(q_i^{\mathrm{g}})$-avoiding and $R_i(q_i^{\mathrm{s}}) \cap R_j(q_j^{\mathrm{g}}(t)) = \emptyset$, $\forall j \neq i$.*

Proof. If $\mathcal{G}(0) \subseteq \mathbb{G}$, then, under A2–A3 and according to Definition 2, $\mathcal{E}(0)$ is admissible. The proof is then given by induction: $\forall t$, $\forall i \in \mathcal{R}$, $\boldsymbol{p}_i \in \mathcal{P}(t) \iff \boldsymbol{p}_i$ is $\mathbb{G}^{j\neq i}$-avoiding, i.e., $\mathcal{E}_i(t)$ will *never* cross *any possible* end-point of another robot, preventing the blocking situation. This also implies A1 and hence **P1** (see Theorem 1) to be both verified $\forall t$. □

Note that Theorem 2 ensures that $\mathcal{E}(t)$ is admissible for *any temporal schedule* $G(t) \subseteq \mathbb{G}$ (modulo assignment).

B. Global prevention

If we admit that goal positions become known at runtime (R2), then Theorem 2 is too conservative (see Fig. 3.a). To address this limitation, similarly to [9], we formulate Theorems 3 and 4 below to provide lighter yet sufficient requirements for preserving admissibility while relying only on the current set of goals $\mathcal{G}$. The resulting feasibility check is then plugged into lines 2–4 of Algorithm 1 as described in the following.

1) Sequential planning: Let us first introduce a binary semaphore Θ on the set $\mathcal{E}$ to ensure consistency while allowing paths to be computed asynchronously by several decoupled motion planners (as allowed in Algorithm 1). Specifically, we assume that Θ is locked (if possible), when a path $\boldsymbol{p}_i$ *should be updated*, that is, if robot i is idle and a new goal is posted to it, or if robot i should re-plan its path

[5]The definition given in [6] is here extended to tackle kinematic constraints and generic robot footprints, while considering, for simplicity, $\mathcal{E}_i = \bigcup_{\sigma\in[0,1]} R_i(\boldsymbol{p}_i(\sigma))$.

	Blocking (lines 2–3, 5)	**Deadlocks (line 5)**	**Livelocks**
Avoidance	Generalized well-formed infrastructure (**Definition 2**) + **Theorem 2**	Totally-ordering heuristics + synchronous goal posting, or FCFS heuristic (**Theorem 7**)	No re-planning and no backtracking along paths (**Theorem 11**)
Global prevention	Online goal checking + rejection/delay (**Theorem 3 or 4**)	Global re-ordering (**Algorithm 3**) + **Theorem 8**	Re-plan only if no backtracking and the new path is shorter (**Theorem 12**)
Local prevention & repair	Re-planning (**Algorithms 6–7**) + **Theorem 9**	Partial re-ordering (**Algorithm 5**) + re-planning (**Algorithms 6–7**)	—

TABLE II: A guide to results of Sections V, VI and VII.

to its current goal to recover from an undesired situation. Whenever Θ is successfully locked, a set of paths ($\boldsymbol{p}_i$ and all the others paths required to check $\boldsymbol{p}_i$'s admissibility, as we will see) may be concurrently computed. Timeouts are used for ensuring termination, and we denote with Δ^{plan} the maximum waiting time for each planning round. According to the returned values, the coordinator checks if the new $\boldsymbol{p}_i$ can be accepted (according to an opportune admissibility criteria), and the sets $\mathcal{P}, \mathcal{C}, \mathcal{T}$ are updated accordingly. Note that Δ^{plan} may be lower or greater than T_c; if $\Delta^{\text{plan}} < T_c$ (assuming A5), more paths may be sequentially updated at each k. Hence, the conditions for preventing F1 which we discuss in the following are stated using continuous time.

2) Feasibility check for new goals: For simplicity, we start by assuming paths to be updated only whenever robots are idle; we will remove this assumption in Section VI-C2.

From now on, let $\mathcal{C}^{\text{end}}(\mathcal{E}_h, \mathcal{E}_k)$ be the set of critical sections determined by $\mathcal{E}_h \cap \mathcal{E}_k^{\{1\}} \neq \emptyset$, $h, k \in \mathcal{R}$, $h \neq k$.

Theorem 3 (Online Admissibility Check)**.** *Assume A2–A6, $\mathcal{E}(t_0)$ is admissible, robot i is idle at time t_0, and a new path $\boldsymbol{p}_i$ has been successfully computed within time $t_0 + \Delta^{\text{plan}}$. $\boldsymbol{p}_i$ can be accepted while ensuring* **P1** *and $\neg$(**F1**) if:*

(i) $\mathcal{C}^{\text{end}}(\mathcal{E}_i, \mathcal{E}_j) \cap \mathcal{C}^{\text{end}}(\mathcal{E}_j, \mathcal{E}_i) = \emptyset$ for all $(i, j \neq i) \in \mathcal{R}^2$;
(ii) for each $C \in \mathcal{C}^{\text{end}}(\mathcal{E}_j, \mathcal{E}_i)$, it is possible to impose $\langle m_j, m_i \rangle C \in \mathcal{T}$ while C is active;
(iii) for each $C \in \mathcal{C}^{\text{end}}(\mathcal{E}_i, \mathcal{E}_j)$, it is possible to impose $\langle m_i, m_j \rangle C \in \mathcal{T}$ while C is active.

Proof. The sketch of proof is given in the following.

P1: By assumption, $\mathcal{E}(0)$ is a safe starting configuration; hence, sequential planning, (ii) and (iii) ensure robots never start from an active critical section, which is a sufficient condition for safety (see [4] for the formal proof).

$\neg$(**F1**)**:** Condition (i) ensures the existence of a set $\mathcal{T}$ preventing at least one among each pair of robots from being blocked, (as exemplified in Fig. 3.a). The proof is trivial via contradiction. Conditions (ii) and (iii) ensure that a feasible ordering can be found for each pair (i, j) by Algorithm 2. □

3) Goal Scheduling: Theorem 3 provides a way to avoid blocking by ensuring that new paths do not interfere with currently posted goals or the envelopes traversed to reach them. Verifying condition (ii) and (iii) is computationally inexpensive, as it requires planning a path for one robot, and possibly imposing a fixed ordering for some critical sections. This is significantly less restrictive than the conditions imposed by Theorem 2, which makes it impossible, e.g., to assign goals $q_i^{\text{g}} \notin \mathbb{G}$. However, Theorem 3 has another pitfall, namely, it does not ensure that goals can always be accepted. Specifically, if $\mathcal{G}(t) \cup \mathbb{G}$ is not a well-formed infrastructure at each t, then

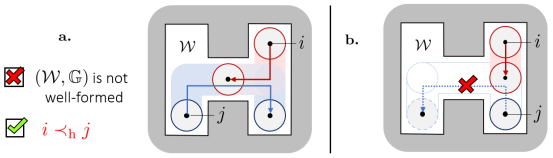


Fig. 3: **a.** Example where Theorem 2 may be too conservative: even if $(\mathcal{W}, \mathbb{G})$ is not well-formed, there exists a feasible $\mathcal{T}$ s.t. both the robots may reach their destination. **b.** Example of entrapment: robot j cannot accept the new mission (dotted arrow) to prevent being blocked by robot i.

there may exists some robots which will be forced to reject goals posted to them in order to prevent being blocked by other robots that are parked in $\mathcal{G} \setminus \mathbb{G}$. An example of this entrapment situation is shown in Fig. 3.b and occurring in real fleet in video [44]. While Theorem 2 ensures that this phenomenon does not happen by definition, this is not true using Theorem 3. As a consequence, the use of Theorem 3 requires either $\mathcal{G} \subseteq \mathbb{G}$ with $\mathbb{G}$ verifying Definition 2, or some form of task scheduling if goal poses are allowed to become known at run time, in order to guarantee admissibility while ensuring that entrapment does not happen. Several possible avenues can be pursued to overcome this limitation. For instance, similarly to [20], enlarged footprints for parking may let the robots navigate inside other robots' private zone. However, the solution of [20] is strongly constrained to a state lattice and is not suitable considering R3–R4 or heterogeneous fleets. .

In the following, we investigate another option which allows to post arbitrary goals $q_i^{\text{g}} \notin \mathbb{G}_i$ while preventing entrapment. For this purpose, let $(\mathcal{W}, \mathbb{G})$ be a well-formed infrastructure. Also, we assume robot i is idle at time t_0, and a new path $\boldsymbol{p}_i$ (with envelope $\mathcal{E}_i$) to have been successfully computed for i within time $t_0 + \Delta^{\text{plan}}$.

Theorem 4 (Revised Online Admissibility Check)**.** *Assume A2–A6, $\mathcal{E}(t_0)$ is admissible, robot i is idle at time t_0, and a new path $\boldsymbol{p}_i$ has been successfully computed within time $t_0 + \Delta^{\text{plan}}$. Also, let $(\mathcal{W}, \mathbb{G})$ be a well-formed infrastructure. $\boldsymbol{p}_i$ can be accepted while ensuring* **P1**, $\neg$(**F1**) *and preventing entrapment if*

(i) it satisfies Theorem 3.
(ii) if $q_i^{\text{g}} \notin \mathbb{G}_i$, there exists $q_i^{\text{b}} \notin \mathbb{G}_i$ and path $\boldsymbol{p}_i^{\text{b}}$ from q_i^{g} to q_i^{b} which is $\tilde{\mathcal{G}}^{j \neq i}(t_0)$-avoiding, with $\tilde{\mathcal{G}}$ being the current set of goals and bases updated with $\{q_i^{\text{g}}, q_i^{\text{b}}\}$;
(iii) there exists a path $\boldsymbol{p}'_j$ from $q_j^{\text{g}}(t_0)$ to a reachable $q_j^{\text{b}}(t_0) \in \mathbb{G}_j$ which is $\tilde{\mathcal{G}}^{k \neq j}(t_0)$-avoiding, q_i^{b} satisfying point (ii).

Proof. The sketch of proof builds upon Theorem 3, which

ensures that **P1** and $\neg$(**F1**) hold due to (i). Also, note that:

(a) Since paths are updated sequentially, (ii) ensures q_i^{b} to be reachable at t_0 and (iii) keeps it reachable over time. Therefore robots may be trapped only in their bases.

(b) Even if trapped, it is always possible to free the path for the trapped robot. According to Definition 2, in fact, if robot j is trapping a robot i, then $q_j^{\mathrm{g}} \notin \mathbb{G}_i$, i.e., $q_j^{\mathrm{g}} \neq q_j^{\mathrm{b}}$. Furthermore, (ii) and (iii) ensure that a path to base for each trapping robot can be executed. □

Note that the complexity of Theorem 4 is a function of the subset of $\mathbb{G}_i$ checked at each time (at most $1+\mathcal{R}$ plans for each check, whenever $|\mathbb{G}_i| = 1$ for all $i \in \mathcal{R}$ and all the previously computed $\boldsymbol{p}_j^{\mathrm{b}}(t_0)$ interfering with the new $\boldsymbol{p}_i \cup \boldsymbol{p}_i^{\mathrm{b}}$), and is valuable in the design phase as a possible trade off between computational overhead and performance (it may be useful at each time to choose the closest q_i^{b} from the current goal, so that a mission to the base is scheduled to let another robot reach a location, then "useless" covered distance is minimized). An extension of the base station concept to private base zone (i.e., bounded regions of the space verifying the requirement of base station) is not mandatory but may be useful to relax the assumption of accurate positioning in q_i^{b}.

In summary, Theorem 3 relaxes the overly conservative constraint required by Definition 2, but requires goal scheduling (to avoid entrapment), as well as appropriately deciding precedences (to enforce conditions (ii) and (iii)). Conversely, Theorem 4 allows to avoid entrapment without resorting to goal scheduling, rather via planning. Note that the choice of deploying a solution based on goal scheduling (Theorem 3) vs. one based on path planning (Theorem 4) depends on the computational overhead of scheduling vs. planning in the particular application at hand. Note also that enforcing Theorem 4 requires computing at least $1+|\mathcal{R}|$ plans for each posted goal. Furthermore, it is also worth considering that one could leverage decoupled prioritized path planning in order to avoid search in the joint configuration space of multiple robots [26] or in the spatio-temporal space [9].

VI. Deadlocks

Let $\mathcal{T}_\rho(t) = \bigcup_{i\in\mathcal{R}} \{\rho\text{-arg}\min_{m_i \geq \sigma_i(t_i)} \{\langle m_i, u_j^C\rangle \in \mathcal{T}(t)\}\}$ be the subset of $\mathcal{T}(t)$ containing the ρ closest yielding constraints for each robot (recall that $\sigma_i(t_i)$ is the last known position for robot i at time k). The parameter ρ defines a "lookahead on precedence constraints" and will be used for deadlock prevention. Specifically, $\mathcal{T}_\infty(\mathrm{k}) = \mathcal{T}(\mathrm{k})$, while $\mathcal{T}_1(\mathrm{k}) \subseteq \mathcal{T}(\mathrm{k})$ contains all the precedence constraints corresponding to the current critical points in $\bar{\Sigma}(\mathrm{k})$. As in [37], given a set $\mathcal{T}_\rho$, we define a dependency graph D_ρ as

Definition 3 (ρ-graph)**.** *The graph D_ρ induced by $\mathcal{T}_\rho$ is a simple digraph (V_ρ, E_ρ) where*

$$V_\rho = \{i \mid \boldsymbol{p}_i \in \mathcal{P}\}$$
$$E_\rho = \{e_{ij},\ (i,j) \in V_\rho \times V_\rho \mid \exists C \in \mathcal{C}_{ij} : \langle m_i, u_j^C\rangle \in \mathcal{T}_\rho\}.$$

Each edge $e_{ij} \in E_\rho$ is also associated to a weight $w_{ij} \in \mathbb{N}$ which encodes the number of precedence constraints $C \in \mathcal{C}_{ij}$ such that $\langle m_i, u_j^C\rangle \in \mathcal{T}_\rho$. In doing so, we can exploit graph tools to detect/prevent deadlocks by searching for cycles in D_ρ which verify a particular spatial condition.

Let $\{i_1, i_2, \ldots, i_{m-1}\} \subseteq \mathcal{R}$ be a subset of robot indices, $i_i \neq i_j$ for each (i_i, i_j) in the subset. Each cycle $w \subseteq D_\rho$, with vertex indices $V_\rho(w) = \{i_1, \ldots, i_{m-1}, i_1\}$, corresponds to at least one set of precedence constraints of $\mathcal{T}_\rho$:

$$\begin{aligned} &\langle m_{i_1}, u_{i_2}^C\rangle,\ C \in \mathcal{C}_{i_1 i_2} \\ &\langle m_{i_2}, u_{i_3}^{C'}\rangle,\ C' \in \mathcal{C}_{i_2 i_3} \\ &\vdots \\ &\langle m_{i_{m-1}}, u_{i_1}^{C''}\rangle,\ C'' \in \mathcal{C}_{i_1 i_{m-1}}. \end{aligned} \tag{3}$$

We will use the notation $\Phi(w)$ to refer to a generic set of $\mathcal{T}_\rho$ induced by the cycle $w \in D_\rho$ and $\mathcal{R}(w) \subseteq \mathcal{R}$ to refer to the set of robots involved in w, i.e., $\mathcal{R}(w) = V_\rho(w)$.

Definition 4 (Nonlive sets)**.** *A set of precedence constraints $\Phi(w) \subseteq \mathcal{T}_\rho$, corresponding to a cycle $w \in D_\rho$ such that $\mathcal{R}(w) = \{i_1, \ldots, i_{m-1}\}$ is* nonlive *iff $u_{i_j}^{C'} > m_{i_j}$ for all the dependencies in $\Phi(w)$, $i_j \in \mathcal{R}(w)$.*

Conversely, we say that $\Phi(w)$ is *live* if there exists at least a pair of precedence constraints $\langle m_{i_j}, u_{i_h}^{C'}\rangle$, $\langle m_{i_h}, u_{i_k}^{C''}\rangle$ such that $u_{i_h}^{C'} < m_{i_h}$ (i.e., the robot i_h can reach $u_{i_h}^{C'}$, allowing robot i_j to proceed along its path). Also, we will say that $\mathcal{T}_\rho(t)$ is live (nonlive) at time t iff it does not contain (it contains) nonlive sets. Note that nonlive sets directly map to nonlive states [43] in the set $\mathcal{T}$.

Furthermore, according to Definition 3, $D_1 = (V_1, E_1) \subseteq D_\infty$ is a dependency graph verifying the properties[6]:

$$V_1 \in D_1 \iff V_\infty \in D_\infty$$
$$E_1 \subseteq E_\infty,\ |E_1| \leq n,\ |E_\infty| \leq n(n-1)$$

If D_1 contains a cycle w, then $w \in D_\infty$ (not necessarily vice versa). Due to the definition of critical point, for all $i \in V_1$, $\mathrm{outdegree}(i) = 1$, so for each cycle $w \in D_1$, the related set of precedence constraints $\Phi(w) \subseteq \mathcal{T}_1$ is unique. Also, for each nonlive set $\Phi(w) \in \mathcal{T}_1$, let $\bar{\Sigma}(w) \subseteq \bar{\Sigma}$ be the corresponding set of critical points.

Definition 5 (Deadlocks)**.** *A deadlock happens whenever D_1 contains a cycle w, $\Phi(w) \subseteq \mathcal{T}_1$ is a nonlive set, and $\sigma_{i_j}(t) = m_{i_j}$ for all the $i_j \in \mathcal{R}(w)$, with each m_{i_j} defined according to* (3) *and Def. 4.*

In particular, it can be noticed that [3, 37]

Theorem 5 (Sufficient condition for the absence of deadlocks)**.** *Under A1, the absence of nonlive sets in the set $\mathcal{T}_\infty(\mathrm{k})$ at each time* k *implies that deadlocks never happen, i.e.,* $\neg$(**F2**).

However, the previous theorem no longer holds when shortening the horizon ρ. Let us prove this claim assuming $\rho = 1$. Even if $\mathcal{T}_1(\mathrm{k})$ does not contain nonlive sets at the current k, there may exist some nonlive sets $\Phi(w) \in \mathcal{T}_\infty \setminus \mathcal{T}_1$ such that $\forall \langle m_{i_j}, u_{i_k}^C\rangle \in \Phi(w)$, $C \in \mathcal{C}_{i_j i_k}$, $\ell_{i_k}^C < \sigma_{i_k}(t + \Delta_{i_k}^{\mathrm{stop}}) < u_{i_k}^C$ and $\sigma_{i_j}(t + \Delta_{i_j}^{\mathrm{stop}}) > m_i$ (i.e., none of the constraints in $\Phi(w)$

[6] The upper bound $n(n-1)$ corresponds to the case of having a complete digraph, i.e., each pair of vertices $(i,j) \in V_\rho \times V_\rho$, $i \neq j$, is joined by a pair of edges $e_{ij} \in E_\rho$ and $e_{ji} \in E_\rho$.

can be reversed and the yielding robot cannot be required to stop at a critical point $m'_{i_j} < m_{i_j}$). Note that whenever a nonlive set cannot be safely reversed according to line 5 of Alg. 1, if $\mathcal{E}$ does not change, then a deadlock will necessarily happen.

A. Avoidance

As a consequence of Theorem 5, imposing D_∞ acyclic every k is a sufficient condition for deadlock avoidance, that is, there should exist a topological order of the vertices V_∞, or, according to Definition 3, there should exist a total order of robots through the set of critical sections $\mathcal{C}(t)$. Our goal is to map this condition back into properties of the heuristic function h and of the task scheduler to ensure $\neg$(**F2**) by design while addressing requirements R1–R6. In particular, a heuristic function h is *totally ordering* if: (i) for every $(i,j) \in \mathcal{R}^2$, $i \neq j$, for all $(C, C') \in \mathcal{C}^2_{ij}$ then either $i \prec_\mathrm{h} j$ or $j \prec_\mathrm{h} i$ at both C and C'; (ii) $i \prec_\mathrm{h} j \wedge j \prec_\mathrm{h} k \implies i \prec_\mathrm{h} k$ for each $k \in \mathcal{R} \setminus \{i,j\}$. Also, we define h as *static* if it is not time dependent.

Theorem 6. *If goals are posted asynchronously to robots, Algorithm 2 cannot ensure D_∞ to be acyclic, even if h is static and totally ordering.*

Proof. Assume at time $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0+1)T_c)$ an idle robot i is assigned to a new path, and h to be static and totally ordering. Also assume there exists a robot j such that $j \prec_\mathrm{h} i$ and a critical section $C \in \mathcal{C}_{ij}(\mathrm{k}_0)$ such that $\sigma_j(t_0 + \Delta_j^{\text{stop}}) > \ell_j^C$. According to Algorithm 2, $\langle m_i, u_j^C \rangle \in \mathcal{T}(\mathrm{k}_0)$. However, j may not be able to exit C if $\exists C' \in \mathcal{C}_{jk}(\mathrm{k}_0)$, such that $u_j^C > \ell_j^{C'}$, $j \prec_\mathrm{h} k \prec_\mathrm{h} i$, and $\sigma_k(t_0 + \Delta_k^{\text{stop}}) \leq \ell_k^{C'}$. □

Consequently, deadlocks may happen if goals as posted asynchronously. However, nonlive sets never happen if, when $\mathcal{T}(t)$ is updated, there is no conflict between $\prec_{\mathrm{h}(t)}$ (with h totally ordering) and the ordering decided by Algorithm 2. In particular,

Theorem 7 (Sufficient heuristic and scheduling properties for deadlock avoidance)**.** *Under A1, assuming each $\mathcal{E}_i(t)$ to be updated only when robot i is idle, then h totally ordering ensures that deadlocks never happen if:*

(i) all the paths are posted synchronously to robots and $\mathcal{P}(t) = \mathcal{P}(t_0)$ for $t \in [t_0, t_1]$ implies $i \prec_{\mathrm{h}(t)} j$ static in $t \in [t_0, t_1]$ for each pair $(i,j) \in \mathcal{R}^2$, $j > i$;

(ii) whenever a new $\mathcal{E}_i(t)$ is accepted (asynchronous goal posting), $i \prec_{\mathrm{h}(t)} j$ for all $j \in \mathcal{R} \setminus \{i\}$. We refer to this heuristic function as First Come First Served (FCFS).

Proof. (i) and (ii) can be proved considering that every time the set $\mathcal{E}$ is updated with a new $\mathcal{E}_i$, robot i is not in motion. Hence, A1 ensures that $i \prec_{\mathrm{h}(t)} j$ is both feasible and safe according to Algorithm 2. As a consequence, $D_\infty(t)$ will be acyclic at each t. □

B. Global prevention

If on one hand Theorem 7 avoids deadlocks by design, on the other forcing D_∞ to be acyclic may be excessively binding according to Definition 4. Also, forcing h to be static may lead to low flexibility, low capability of handling contingencies and useless blocking time (e.g., the distance from critical sections does not affect precedence orders). To overcome this limitation, in this section we exploit Theorem 5 in a less conservative way: to allow heuristics to be dynamic while accounting both for **P1** and **P2**, we alter line 5 of Algorithm 1 to detect and recover from nonlive sets in the current $\mathcal{T}$ before they end up in a deadlock by reversing precedence orders (if dynamically feasible) to break the cycle. Since $\rho = \infty$, this ensures that deadlocks never happen — however complexity may be exponential.

The strategy can be implemented in two ways: (a) revise all the constraints in $\mathcal{T}$ and then check for nonlive sets, or (b) check for nonlive sets while revising each precedence constraint. Functionally, the aforementioned strategies are equivalent; however, in the worst case, (a) may require an exhaustive search over all reversible orderings, which clearly has exponential complexity. Hence, we propose an approach in line with (b) based on revising then filtering constraints. The proposed global strategy for deadlock prevention makes use of an incremental computation of cycles in D_∞ (see next paragraph) to significantly reduce the computational overhead of filtering.

1) Incremental computation of cycles: The detection of nonlive sets in $\mathcal{T}$ requires the computation of all the cycles in D_∞, resulting in time complexity $O(2^{n \log n})$ in the worst case[7]. However, since the number of updated edges between consecutive checks of D_∞ is usually smaller than $|\mathcal{T}|$, we leverage incremental computation to reduce the average time required to perform this step. For this purpose, we require the system to maintain an up-to-date list $\mathcal{L}(e_{ij})$ for each $e_{ij} \in E_\infty$. At each time t, the list $\mathcal{L}(e_{ij})$ contains the current cycles in $D_\infty(t)$ involving the edge e_{ij}. Note that the higher the density of the graph, the higher the number of cycles involving each e_{ij} and hence the size of $\mathcal{L}$. For instance, the number of cycles involving each e_{ij} in a complete graph is equal to $\sum_{i=1}^{n-2} \binom{n-2}{i}$, which results in an exponentially sized $\mathcal{L}$. In Section VIII-C we analyse empirically the practical feasibility of the proposed approach in terms of scalability with the number of robots.

2) The global re-ordering algorithm: Let us first prove the conditions under which checking for nonlive sets and re-ordering iteratively ensure $\mathcal{T}(t)$ to be live at each t.

Theorem 8. *Assume: A2, $\mathcal{T}(0) = \emptyset$, and paths to be planned sequentially only when robots are idle (no re-planning) and any of Theorem 2, Theorem 3 and Theorem 4 holds, i.e., $\neg$(**F1**). At each t, the following prepositions are satisfied:*

(A) if $\mathcal{E}(\mathrm{k}) = \mathcal{E}(\mathrm{k}-1)$ and $\mathcal{T}(\mathrm{k}-1)$ is live, then holding precedence orders for all $C \in \mathcal{C}(\mathrm{k})$ ensures $\mathcal{T}(\mathrm{k})$ to be live.

(B) for each $\mathcal{E}_i$ that is updated (let $[t_0, t_1]$ be the related planning interval, $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0+1)T_c)$, $t_1 \in [\mathrm{k}_1 T_c, (\mathrm{k}_1 +$

[7]One of the best state-of-the-art algorithm for computing all the cycles in a direct graph, Johnson's Algorithm [45], has time complexity $O((|V|+|E|)(c+1))$, where c is the number of cycles in the graph. Since a complete graph with n vertices has $\sum_{i=1}^{n-1} \binom{n}{n-i+1}(n-i)!$ cycles, the resulting time complexity is equal to $O\left(n^2\left[1 + \sum_{i=1}^{n-1} \binom{n}{n-i+1}(n-i)!\right]\right) \approx O(2^{n \log n})$.

$1)T_c)$*), there exists an ordering such that if* $\mathcal{T}(t_0)$ *is live then* $\mathcal{T}(t_1)$ *is live.*

(C) if $\mathcal{T}(\mathrm{k}-1)$ *is live, then sequential checks and dynamic heuristics can ensure* $\mathcal{T}(\mathrm{k})$ *to be live.*

Proof. (A) If $\mathcal{E}$ does not change, then $\mathcal{C}(\mathrm{k}) \subseteq \mathcal{C}(\mathrm{k}-1)$. Also, if no order is reversed, then $\mathcal{T}(\mathrm{k}) \subseteq \mathcal{T}(\mathrm{k}-1)$. Hence, $\mathcal{T}(\mathrm{k}-1)$ and $\mathcal{T}(\mathrm{k})$ belong to the same homotopic class of trajectories through the set $\mathcal{C}(\mathrm{k})$, so they have the same behaviour with respect to **P1** and **P2**.

(B) If $\mathcal{T}(t_0)$ is live, then holding the previously decided precedence orders and imposing that robot i yields for all the other robots ensures two properties. First, it preserves the liveness of the set. If paths are updated sequentially, then $\mathcal{E}(\mathrm{k}_1) - \mathcal{E}(\mathrm{k}_0) = \{\mathcal{E}_i\}$. Also, according to point A, $\mathcal{T}(t_0)$ live implies that if there exists a nonlive set $\Phi(w) \in \mathcal{T}(t_1)$, then it belongs to new critical sections involving robot i, i.e., $i \in \mathcal{R}(w)$. However, Theorems 3 and 4 ensure that the starting pose of robot i at time t_1 is outside every $C \in \mathcal{C}(\mathrm{k}_1)$. Consequently, at the start, no robot is yielding for i, so i yields for j at every $C \in \mathcal{C}(\mathrm{k}_1)$ ensures that $\mathcal{T}(t_1)$ is live. Second, it is in agreement with condition (iii) of Theorem 3 (and hence with condition (i) of Theorem 4). Consequently, if $\mathcal{T}(0)$ is live then $\mathcal{T}(\mathrm{k})$ is live.

Note: Theorem 5 is simply a particular instance of this Theorem, where *all* the precedence constraints are updated according to the FCFS heuristic.

(C) Let $(\mathcal{T}^0, \mathcal{T}^1, \ldots, \mathcal{T}^{|\mathcal{C}^{\mathrm{rev}}|-1})$ be a sequence containing all possible sets of precedence constraints on the same set $\mathcal{C}$, and assume that each set in the sequence differs from the previous one by at most one precedence constraint. At each time k, we assume $\mathcal{T}^0$ to be updated keeping the precedence order decided at time $\mathrm{k}-1$ for all $C \in \mathcal{C}(\mathrm{k}-1) \cap \mathcal{C}(\mathrm{k})$ and according to the FCFS heuristic for new critical sections; also, we will have $\mathcal{T}^{|\mathcal{C}^{\mathrm{rev}}|-1} = \mathcal{T}(\mathrm{k})$. Then, for each reversible constraint $\langle m_i, u_j^C \rangle$,

1) get the order given by the heuristic function while enforcing condition (ii) and (iii) of Theorem 3 (or (i) of Theorem 4);
2) if $j \prec_{\mathrm{h}} i$ is returned, then compute the reversed constraint $\langle m_j, u_i^C \rangle$ and check if $\mathcal{T}^{\mathrm{tmp}} \leftarrow (\mathcal{T}^{l-1} \setminus \{\langle m_i, u_j^C \rangle\}) \cup \{\langle m_j, u_i^C \rangle\}$ contains nonlive cycles. If not, $\mathcal{T}^l \leftarrow \mathcal{T}^{\mathrm{tmp}}$. Otherwise, $\mathcal{T}^l \leftarrow \mathcal{T}^{l-1}$, $l \in [1, |\mathcal{C}^{\mathrm{rev}}| - 1]$.

Therefore, $\mathcal{T}(\mathrm{k})$ is live by construction. □

Theorem 8 allows to design Algorithm 3 so that it ensures that $\mathcal{T}(t)$ is live for all t. At each coordination cycle k:

1) *Pre-loading.* Theorem 8.A and Theorem 8.B are used to *pre-load a complete set* $\mathcal{T}(\mathrm{k})$ which is *known to be live* (lines 3–19). Specifically, first, obsolete critical sections are filtered out in lines 4–7. Then, all the precedence constraints belonging to active critical sections are updated by holding the previous decided order according to Theorem 8.A (lines 14–16), while new critical sections are updated using the FCFS heuristic, as stated in Theorem 8.B (lines 9–12). Also, reversible constraints are tracked (lines 17–18).
2) *Revising.* The pre-loaded $\mathcal{T}$ is then refined according to heuristic decisions while enforcing condition (ii) and (iii) of Theorem 3 (or (i) of Theorem 4) as specified by the proof of Theorem 8.C (lines 22–36).

D_∞ and $\mathcal{L}$ are then updated according to changes in lines 21 and 27. Note that $\mathcal{T}(t)$ is guaranteed to be live *whatever the order* in which precedence constraints are revised and *whatever the heuristic function* used.

Algorithm 3: The *revise* function with global re-ordering (implements line 5 of Alg. 1).

Input: $\mathcal{P}$ current set of paths; $\mathcal{C}$ (possibly empty) set of pairwise critical sections; $\mathcal{C}^{\mathrm{new}}$ (possibly empty) set of new critical sections; $\mathcal{T}^{\mathrm{old}}$ (possibly empty) previous set of precedence constraints; D_∞ (possibly empty) previous dependency graph; Σ^{old} previous set of critical points; $\mathcal{L}$ previous set of detected cycles; $s_i(t_i)$ last received robot state, including $\sigma_i(t_i)$.

Output: $\mathcal{T}^{\mathrm{rev}}$ set of revised precedences constraints; D_∞ revised minimal dependency graph; $\mathcal{L}$ revised set of detected cycles.

1 $\mathcal{T}^{\mathrm{rev}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{del}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{add}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{upd}} \leftarrow \emptyset$, $\mathcal{T}^l \leftarrow \emptyset$;
2 $D_\infty = \emptyset$, $\mathcal{C}^{\mathrm{rev}} \leftarrow \emptyset$;
3 **foreach** $C \in \mathcal{C}_{ij}, \mathcal{C}_{ij} \in \mathcal{C}$ **do**
4 **if** $\sigma_i(t_i) \geq u_i^C \vee \sigma_j(t_j) \geq u_j^C$ **then**
5 $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$;
6 $(h, k) \leftarrow$ get order from $\mathcal{T}^{\mathrm{old}}$;
7 $\mathcal{T}^{\mathrm{del}} \leftarrow \mathcal{T}^{\mathrm{del}} \cup \{\langle m_h, u_k^C \rangle\}$;
8 **else**
9 **if** $C \in \mathcal{C}^{\mathrm{new}}$ **then**
10 $(h, k) \leftarrow$ get FCFS order;
11 $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
12 $\mathcal{T}^{\mathrm{add}} \leftarrow \mathcal{T}^{\mathrm{add}} \cup \{\langle m_h, u_k^C \rangle\}$;
13 **else**
14 $(h, k) \leftarrow$ get previous order from $\mathcal{T}^{\mathrm{old}}$;
15 $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
16 $\mathcal{T}^{\mathrm{upd}} \leftarrow \mathcal{T}^{\mathrm{upd}} \cup \{\langle m_h, u_k^C \rangle\}$;
17 **if** $\sigma_i(t_i + \Delta_i^{\mathrm{stop}}) \leq \ell_i^C \wedge \sigma_j(t_j + \Delta_j^{\mathrm{stop}}) \leq \ell_j^C$ **then**
18 $\mathcal{C}^{\mathrm{rev}} \leftarrow \mathcal{C}^{\mathrm{rev}} \cup \{C\}$;
19 $\mathcal{T}^l \leftarrow \mathcal{T}^l \cup \{\langle m_h, u_k^C \rangle\}$;
20 $\mathcal{T}^l \leftarrow \mathcal{T}^{\mathrm{upd}} \cup \mathcal{T}^{\mathrm{add}}$;
21 $(D_\infty, \mathcal{L}) \leftarrow$ updateGraph$(D_\infty, \mathcal{L}, \mathcal{T}^{\mathrm{old}}, \mathcal{T}^{\mathrm{del}}, \mathcal{T}^{\mathrm{add}})$;
22 **foreach** $C \in \mathcal{C}^{\mathrm{rev}}$ **do**
23 $(h, k) \leftarrow$ get order from heuristic while enforcing condition (ii) and (iii) of Theorem 3 (or (i) of Theorem 4);
24 **if** $\langle m_h, u_k^C \rangle \notin \mathcal{T}^l$ **then**
25 $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
26 $\mathcal{T}^{\mathrm{tmp}} \leftarrow \mathcal{T}^l \setminus \{\langle m_k, u_h^C \rangle\} \cup \{\langle m_h, u_k^C \rangle\}$;
27 $(D_\infty^{\mathrm{tmp}}, \mathcal{L}^{\mathrm{tmp}}) \leftarrow$ updateGraph$(D_\infty, \mathcal{L}, \mathcal{T}^l, \langle m_k, u_h^C \rangle, \langle m_h, u_k^C \rangle)$;
28 live $\leftarrow$ true;
29 **foreach** $w \in \mathcal{L}^{\mathrm{tmp}}(e_{hk}) - \mathcal{L}(e_{hk})$ **do**
30 **foreach** $\Phi(w) \subseteq \mathcal{T}^{\mathrm{tmp}}$ **do**
31 **if** $\Phi(w)$ *is nonlive* **then**
32 live $\leftarrow$ false;
33 **break**;
34 **if** *live* **then**
35 $\mathcal{T}^l \leftarrow \mathcal{T}^{\mathrm{tmp}}$;
36 $(D_\infty, \mathcal{L}) \leftarrow (D_\infty^{\mathrm{tmp}}, \mathcal{L}^{\mathrm{tmp}})$;
37 $\mathcal{T}^{\mathrm{rev}} \leftarrow \mathcal{T}^l$;
38 **return** $(D_\infty, \mathcal{L}, \mathcal{T}^{\mathrm{rev}})$

C. Local prevention and repair

In this section we present two local strategies for deadlock prevention and recovery, namely, partial re-ordering and re-planning. These limit the search for nonlive sets to $\mathcal{T}_1$, which allows to drastically reduce the complexity of computing cycles (from $O(2^{n \log n})$ to $O(n^2)$ in the worst case) at the price of losing completeness. Specifically, line 5 of Algorithm 1 is implemented by sequencing Algorithm 2 (revise $\mathcal{T}$) and Algorithm 4 (*check&repair* the revised $\mathcal{T}$).

In Section VIII-D we will analyse the practical effectiveness of the two methods, both when they are used on their own and jointly. Note that, in case of unsuccessful deadlock recovery, A1 (possibly relaxed according to Theorems 3 or 4) ensures that a solution exists at any t. This entails that, whenever this undesired situation happens, it is always possible to resort to complete (possibly coupled) strategies [26].

Algorithm 4: The *check&repair* function (to be invoked following Alg. 2).

Input: $\mathcal{T}$ (possibly empty) current set of precedence constraints; Ω list of robots' semaphores (Section VI-C2a); $\bigcup_{i\in\mathcal{R}} s_i(t_i)$ last communicated status (each containing $\sigma_i(t_i)$).
Parameters : $\mathrm{re-order}$ true if re-ordering is enabled; $\mathrm{re-plan}$ true if re-plan is enabled; $\mathrm{stat-replan}$ true if re-plan should be static (Section VI-C2b); $\mathrm{locking}$ true if each re-plan locks Θ.

1 $\mathcal{T}_1 \leftarrow$ compute the closest constraint in $\mathcal{T}$ for each $i \in \mathcal{R}$;
2 $\Psi \leftarrow$ compute the nonlive sets in $\mathcal{T}_1$;
3 **if** $\mathrm{re-order}$ **then**
4 $\upsilon \leftarrow 0$;
5 **while** $\upsilon < |\Psi|$ **do**
6 $\Phi(w) \leftarrow$ get one unvisited nonlive set in Ψ;
7 $\Psi \leftarrow$ mark $\Phi(w)$ as visited;
8 $(\mathcal{T}, \mathcal{T}_1, \Psi) \leftarrow$ reorderConstraints$^{(1)}(\mathcal{T}, \mathcal{T}_1, \Psi, \Phi(w), \bigcup_{i\in\mathcal{R}} s_i(t_i))$;
9 $\upsilon \leftarrow$ get the number of visited sets in Ψ;
10 **if** $\mathrm{re-plan}$ **then**
11 update and communicate the set Σ;
12 **foreach** $\Phi(w) \in \Psi$ **do**
13 start-replan $\leftarrow$ true;
14 **foreach** $i \in \mathcal{R}(w)$ **do**
15 **if** Ω_i *is locked or* $\mathrm{stat-replan} \wedge \sigma_i \neq \bar{\sigma}_i$ **then**
16 start-replan $\leftarrow$ false;
17 **break**;
18 **if** *start-replan* **then**
19 **foreach** $i \in \mathcal{R}(w)$ **do**
20 $\Omega \leftarrow$ lock the semaphore Ω_i and store $\bar{\sigma}_i$;
21 $\mathcal{T} \leftarrow \mathcal{T} \cup \{\langle \bar{\sigma}_i, \emptyset \rangle\}$;
22 **if** $\mathrm{locking}$ **then** lock the semaphore Θ;
23 startThread$^{(2)}$(rePlan$^{(3)}$, $\mathcal{R}(w)$);

(1) Implemented in Algorithm 5. (2) Start a thread with the given body function and arguments. (3) Implemented in Algorithm 6.

1) Partial re-ordering: Algorithm 5 implements a Breadth First Search for a live set of precedences $\mathcal{T}_1$; note that the use of a depth bound of one step ensures that the algorithm terminates after at most n rounds. Specifically, if nonlive sets are detected in the current $\mathcal{T}_1$ (line 2 of Algorithm 4), nonlive sets are checked one by one (line 6 of Algorithm 4). Reversible constraints belonging to the selected nonlive set (line 2) are temporarily reversed (lines 4–7 of Algorithm VI-C1), and the new order is maintained only when the number of nonlive cycles in the updated set $\mathcal{T}_1$ decreases (lines 8–9 of Algorithm VI-C1). The resulting time complexity is polynomial in the number of robots[8]. However, deadlocks may happen (whenever all the constraints of a nonlive set in $\mathcal{T}_1$ cannot be safely reversed).

Algorithm 5: The *reorderConstraints* function

Input: $\mathcal{T}$ current set of precedence constraints; $\mathcal{T}_1$ set of closest constraints; Ψ current set of nonlive sets in $\mathcal{T}_1$; $\Phi(w)$ a nonlive set; $\bigcup_{i\in\mathcal{R}} s_i(t_i)$ last communicated status.
Output: $(\mathcal{T}, \mathcal{T}_1, \Psi)$ revised set of precedence constraints, of closest constraints in $\mathcal{T}$ and of nonlive sets in $\mathcal{T}_1$.

1 $(\mathcal{T}^{\mathrm{tmp}}, \mathcal{T}_1^{\mathrm{tmp}}, \Psi^{\mathrm{tmp}}) \leftarrow (\mathcal{T}, \mathcal{T}_1, \emptyset)$;
2 $\Phi^{\mathrm{rev}} \leftarrow$ get the reversible constraints in $\Phi(w)$;
3 **foreach** $\langle m_h, u_k^C \rangle \in \Phi^{\mathrm{rev}}$ **do**
4 $m_k \leftarrow$ update according to (1), $s_k(t_k)$ and $s_h(t_h)$;
5 $\mathcal{T}^{\mathrm{tmp}} \leftarrow \left(\mathcal{T}^{\mathrm{tmp}} \setminus \{\langle m_h, u_k^C \rangle\}\right) \cup \{\langle m_k, u_h^C \rangle\}$;
6 $\mathcal{T}_1^{\mathrm{tmp}} \leftarrow$ update closest constraint in $\mathcal{T}^{\mathrm{tmp}}$ for h and k;
7 $\Psi^{\mathrm{tmp}} \leftarrow$ update nonlive sets in $\mathcal{T}_1^{\mathrm{tmp}}$;
8 **if** $|\Psi^{\mathrm{tmp}}| < |\Psi|$ **then**
9 **return** $(\mathcal{T}^{\mathrm{tmp}}, \mathcal{T}_1^{\mathrm{tmp}}, \Psi^{\mathrm{tmp}})$;
10 **return** $(\mathcal{T}, \mathcal{T}_1, \Psi)$

2) Re-planning: The approach aims to prevent/recover from deadlocks by changing the paths of robots involved in a nonlive set in $\mathcal{T}_1$. Note that Theorem 2, Theorem 3 and Theorem 4 ensure a solution of the coordination problem exists at each time and can always be computed imposing all the robots to stop and resorting to coupled motion planners. However, to overcome the exponential complexity of such approaches, Algorithm 6 investigates a decoupled approach to repair deadlocks, at the price of incompleteness.

To increase the probability of satisfying **P2**, similarly to [46], multiple solutions may be evaluated by the coordinator at each re-plan using a global cost function. As we will see in Section VII, this may also reduce the probability of livelocks to happen. All algorithms henceforth are designed to support this functionality, but are tested assuming that only one path per robot is computed at a time (we will address a possible extension in future work).

The rest of the Section is organised as follows. First, in Section VI-C2a, we formally state the condition under which Algorithm 1, with lines 2–4 specified according to Section V, safely supports re-planning. Then, in Section VI-C2b, we investigate the efficacy of re-planning in recovering from deadlocks. As in [4], both the analyses are given while assuming $\tau_{\max}^{\mathrm{ch}} \geq 0$ but $\eta = 0$, that is, messages can be delayed but not lost (i.e., $\eta = 0$).

[7] For each cycle $w \in D_1$, the related weakly connected component S_w is the set of $i \in \mathcal{R}$ such that the undirected graph induced by D_1 contains a path between v_i and a vertex $v_j \in V_1$, with $j \in \mathcal{R}(w)$.

[8] D_1 contains at most n vertices and n edges, so the number of cycles is upper-bounded by $\lfloor n/2 \rfloor$. Hence, in the worst case all the cycles in D_1 can be computed with Johnson's Algorithm with a time complexity of $O(n^2)$.

a) Integrating re-planning in Algorithm 1: From now on, we refer to *static re-planning* as the condition in which a robot i is required to yield at its critical point before re-planning can start. Conversely, *dynamic re-planning* is the condition by which re-planning can occur while robots are in motion.

a.1) Safety: Let $i \in \mathcal{R}$ be a robot which is computing a new path in the planning interval $[t_0, t_1]$, $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0+1)T_c]$, $t_1 \in [\mathrm{k}_1 T_c, (\mathrm{k}_1+1)T_c]$, $t_1 \leq t_0 + \Delta^{\text{plan}}$. Assume that a new path is successfully returned, and let $\boldsymbol{p}_i^{\text{old}} \in \mathcal{P}(t_0)$ and $\boldsymbol{p}_i \in \mathcal{P}(t_1)$ be the paths of robot i before and after re-planning. Also, let $\bar{q}_i(\mathrm{k}) = \boldsymbol{p}_i^{\text{old}}(\bar{\sigma}_i(\mathrm{k}))$.

Theorem 9 (Sufficient conditions for safe re-planning)**.** *Let $\mathcal{E}(t_0)$ be admissible and A2–A6 hold. Then, at time t_1, the new $\mathcal{E}_i(t_1)$ preserves* **P1** *if:*

(i) $q_i \in \bigcup_{\sigma=0}^{\bar{\sigma}_i(\mathrm{k}_0)} \boldsymbol{p}_i^{\text{old}}(\sigma) \implies q_i \in (\boldsymbol{p}_i^{\text{old}} \cap \boldsymbol{p}_i)$*, i.e., the new path should overlap the previous one till the configuration corresponding to the last critical point.*

(ii) $\bar{\sigma}_i(\mathrm{k}) \leq \bar{\sigma}_i(\mathrm{k}_0)$ *for* $\mathrm{k} \in \mathbb{N}$, $\mathrm{k}_0 \leq \mathrm{k} < \mathrm{k}_1$.

(iii) For each new active critical section $C \in \mathcal{C}(t_1)$ such that robot i cannot stop before entering it, the previous decided order should be maintained.

Proof. (i) The condition preserves continuity. (ii) If there exists $\mathrm{k} \in \mathbb{N}$, $\mathrm{k}_0 \leq \mathrm{k} < \mathrm{k}_1$, such that $\bar{\sigma}_i(\mathrm{k}) > \bar{\sigma}_i(\mathrm{k}_0)$, then the robot may have already reached a configuration $\boldsymbol{p}_i^{\text{old}}(\sigma_i(t)) \notin \bigcup_{q_i \in [\boldsymbol{p}_i(0), \boldsymbol{p}_i(1)]} \boldsymbol{p}_i(q_i)$, and the executed path may not be collision-free. The condition prevents this undesired situation. (iii) If the condition is not verified, then a collision may happen. Let $\mathcal{C}^{\text{old}}$ and $\mathcal{C}(t_1)$ be the sets of active critical sections before and after the new path has been accepted, respectively. Then, i may be safely required to yield (according to Algorithm 2) for all the active critical sections $C \in \mathcal{C}(t_1)$ such that $\bar{q}_i(\mathrm{k}_1 - 1)$ precedes $\boldsymbol{p}_i(\ell_i^C)$ along $\boldsymbol{p}_i$. If the condition is not satisfied, then there is only one pair of critical sections (C, C'), $C \in \mathcal{C}(t_1)$, $C' \in \mathcal{C}^{\text{old}}$, such that $\boldsymbol{p}_i^{\text{old}}(\ell_i^{C'}) = \boldsymbol{p}_i(\ell_i^C) \wedge \left(\ell_j^{C'} = \ell_j^C \vee u_j^{C'} = u_j^C\right)$. □

Corollary 1. *If $\mathcal{T}(t_1)$ is properly updated according to Theorem 9, then it is not necessary for $\bar{\sigma}_j(\mathrm{k})$ to be constant for all $\mathrm{k} \in \mathbb{N}$, $\mathrm{k}_0 \leq \mathrm{k} < \mathrm{k}_1$, $j \neq i$ in order for* **P1** *to hold.*

Proof. Since $\boldsymbol{p}_j$ may change only after t_1 (sequential planning), then $R_j(\boldsymbol{p}_j(\sigma)) \cap \mathcal{E}_i(\mathrm{k}_1) \neq \emptyset \implies \exists C \in \mathcal{C}_{ij}(\mathrm{k}_1)$ such that $\ell_j^C < \sigma < u_j^C$ for all $\sigma \in [0,1]$. □

a.2) Liveness: Theorems 3 and 4 are extended to support *any re-planning* by requiring also Theorem 9 to be satisfied. Note that, if $\mathcal{T}(t_1)$ is properly updated according to point (ii) of Theorem 9, then any robot j such that $R_j(q_i^g) \cap \mathcal{E}_i^{[0,\bar{\sigma}_i(t_1)]} \neq \emptyset$ will continue to be required to yield for robot i, preserving $\neg$(**F1**).

Since re-planning affects the set $\mathcal{E}$ without modifying the set $\mathcal{G}$, it does not explicitly require Θ to be locked. Condition (i), in fact, will preserve admissibility whether Θ is locked or not by rejecting paths (when returned) which may lead to blocking according to the current set $\mathcal{G}$. Specifically, if Θ is locked, then Theorem 9 and conditions (i–ii) can be encoded directly into the planning phase (since $\mathcal{G}(t_0) = \mathcal{G}(t_1)$, so the $\mathcal{G}(t_0)$-avoiding property can be added as a constraint for the planner). Consequently, it is ensured that whenever a path is successfully computed, it will be accepted.

b) Re-planning to handle deadlocks: Relying on results of Section VI-C2a, Algorithms 4 (lines 10–23), 6 and 7 realise a decoupled re-planning approach to safely prevent/repair deadlocks. Whenever a nonlive $\Phi(w)$ is detected (Algorithm 4 at lines 12–23), if all the robots involved in the deadlock are not already involved in re-planning (Algorithm 4 at lines 14–17), a re-planning thread is started (Algorithm 4 at lines 18–23). The coordinator checks for the existence of an alternative path from the last communicated critical point before re-planning to the current goal for each $i \in \mathcal{R}(w)$ is started (Algorithm 6 at line 3). The new path is computed considering the current waiting poses of robots which may be forced to wait if the deadlock will happen (i.e., in the weakly connected component S_w of the cycle w) and the current set of robot goals (to prevent blocking) as obstacles (Algorithm 7 at lines 4–7). When at least one alternative path is successfully computed (Algorithm 6 at lines 15–18), the sets $(\mathcal{E}, \mathcal{P}, \mathcal{C}, \mathcal{T})$ are updated with the best path and according to the result of the admissibility check (Algorithm 6 at lines 8–11).

Algorithm 6: The *rePlan* function

Input: $\mathcal{R}(w)$ deadlocked robots.
Parameters : c (possibly null) cost function to evaluate multiple solutions; locking true if each re-plan locks Θ.

```
1  replace ← false;
2  t_0 ← getTime();
3  foreach i ∈ R(w) do Request replanPath(i, p_i);
4  while ¬(replace) ∧ getTime() − t_0 < Δ^plan do sleep T_r;
5  if Π ≠ ∅ then
6      foreach i = 0 : |Π| − 1 do
7          (P^tmp, C^tmp) ← update P and C according to Π(i);
8          if (E^tmp, T) is admissible then
9              (P, C) ← (P^tmp, C^tmp);
10             T^old ← restore non reversible constraints for over-
                lapping (C, C'), C ∈ C_ij, C' ∈ C_ij^old, (i, j) ∈ R^2;
11             break;
12 if locking then unlock Θ;
13 foreach i ∈ R(w) do unlock Ω_i;
14 return

15 OnResponce replanPath(i, p_i):
16     add p_i to the list Π ordered by c;
17     if c ≠ null then replace ← |Π| < |R(w)|;
18     else replace ← Π ≠ ∅ ;
```

Remark 1. *To address the conditions (i) and (ii) of Theorem 9, we introduce the set $\Omega = \bigcup_{i \in \mathcal{R}} \langle \Omega_i, m_i \rangle$, where Ω_i is a binary semaphore and $m_i \in [0,1]$ is the last critical point $\bar{\sigma}_i$ sent to robot i before locking Ω_i. Lines 19–21 of Algorithm 4 are used for this purpose. While a semaphore Ω_i is locked, to ensure condition (ii) to hold, a fictitious precedence constraint $\langle m_i, \emptyset \rangle$ (namely, a stopping point) is added to the set $\mathcal{T}$ at each* k*, with m_i being the corresponding critical point stored in Ω. This stopping point is then removed only when the re-plan terminates (Algorithm 6 at line 13). This addresses condition (ii). Combined with line 8 of Algorithm 7 it also addresses*

Algorithm 7: The *replanPath* function

Input: i index of the robot for which the new plan is computed; $\boldsymbol{p}_i$ current path of robot i.
Output: a new planned path for robot i (empty if failure).

1 $\boldsymbol{p}_i^{\text{old}} \leftarrow \boldsymbol{p}_i$, $\mathcal{O}^{\text{tmp}} \leftarrow \mathcal{O}$;
2 $(\mathcal{T}, \bar{\Sigma}, \mathcal{D}_1) \leftarrow$ get current sets and graph$^{(1)}$;
3 $S_w \leftarrow$ get the weakly connected component of $\mathcal{R}(w)$ in $D_1{}^9$;
4 **foreach** $j \neq i$ **do**
5 $\quad \mathcal{O}^{\text{tmp}} \leftarrow \mathcal{O}^{\text{tmp}} \cup \{R_j(\boldsymbol{p}_j(1))\}$;
6 $\quad$ **if** $j \in \mathcal{R}(S_w)$ **then** $\mathcal{O}^{\text{tmp}} \leftarrow \mathcal{O}^{\text{tmp}} \cup \{R_j(\boldsymbol{p}_j(\bar{\sigma}_j))\}$;
7 $\boldsymbol{p}_i \leftarrow \text{planPath}(\boldsymbol{p}_i^{\text{old}}(\bar{\sigma}_i), \boldsymbol{p}_i^{\text{old}}(1), \mathcal{O}^{\text{tmp}})$;
8 **if** $\boldsymbol{p}_i \neq \emptyset \wedge \boldsymbol{p}_i \neq \boldsymbol{p}_i^{\text{old}}$ **then** $\boldsymbol{p}_i \leftarrow \bigcup_{\sigma=0}^{\bar{\sigma}_i} \boldsymbol{p}_i^{\text{old}}(\sigma) \cup \boldsymbol{p}_i$;
9 **return** $\boldsymbol{p}_i$

(1) Allowing multi-threading, hereby is assumed the variable to be opportunely locked during updates.

condition (i). Finally, condition (iii) of Theorem 9 is handled in line 10 of Algorithm 6.

In the following, we analyse different design choices for Algorithm 4 and 6.

b.1) Parallel vs. sequential re-plan: If at the same k there are two nonlive sets in $\Phi(v)$, $\Phi(u) \in \mathcal{T}_1(\mathrm{k})$, $u \neq v$, they affect the motion of completely different sets of robots. More formally,

Theorem 10. *Consider cycles $w, u \in D_1(\mathrm{k})$, $w \neq u$. Since $\mathcal{R}(w) \cap \mathcal{R}(u) = \emptyset$, then $S_w \cap S_u = \emptyset$.*

Proof. For all $i \in V_1$, $\text{outdegree}(i) = 1$, i.e., $\mathcal{T}_1$ contains just the closest precedence constraint for each robot, hence the condition holds. □

Hence, it is possible to parallelise re-planning for each nonlive set in $\mathcal{T}_1(\mathrm{k}_0)$. However, each re-planning will potentially couple two set S_w and S_u, and some re-planning may not be effective.

b.2) Static vs. dynamic re-planning:

If $\boldsymbol{p}_i$ is successfully re-planned, then by construction for all $j \in \mathcal{R}(S_w)$ $\mathcal{E}_i^{[\bar{\sigma}_i(\mathrm{k}_0),1]} \cap R_j(\bar{\sigma}_j(\mathrm{k}_0)) = \emptyset$, that is, all the robots $j \in \mathcal{R}(S_w)$ yielding for i according to the previous $\mathcal{T}$, can reach a location that is no longer in a critical section shared with i. As a consequence, at time k_1 their critical point is updated, i.e., $\bar{\sigma}_j(\mathrm{k}_1) \geq \bar{\sigma}_j(\mathrm{k}_1 - 1)$. If $\sigma_i(\mathrm{k}_0) = \bar{\sigma}_i(\mathrm{k}_0)$ for all $i \in \mathcal{R}(w)$ (static re-planning), then at time k_1, i is not inside any critical sections shared a $j \in \mathcal{R}(w)$, so its critical point is updated. However, due to the change of path (that may lead to discontinuities in σ), it may be that $\bar{\sigma}_i(\mathrm{k}_1) \leq \bar{\sigma}_i(\mathrm{k}_0)$.

In case of dynamic re-planning, at time k_1 robot i may be already inside a critical section shared with $j \in \mathcal{R}(S_w)$, so the critical point related to $\bar{q}_i(\mathrm{k}_1 - 1)$ may be re-communicated. Note that dynamic re-planning allows deadlocks to be anticipated, so it may reduce the probability of a robot being locally trapped.

Summarising, Algorithm 6 does not ensure that deadlocks will never happen (as we will see, the same holds for livelocks), hence **P2** may not hold in the local setting (see Fig. 4.b as a proof).

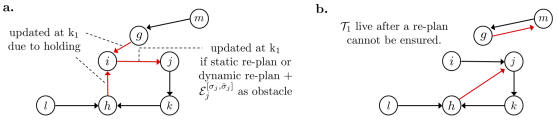


Fig. 4: Liveness of re-planning: **a.** static vs. **b.** dynamic.

VII. Livelocks

Definition 6 (Livelock)**.** *A livelock happens whenever there exists at least one robot i for which the executed trajectory $q_i(t)$ contains a sequence of states (usually circular) such that the state changes during time, but robot i will never reach its goal.*

A. Avoidance

Theorem 11 (Sufficient condition to avoid livelocks)**.** *Livelocks never happen if robots are not allowed to re-plan, nor to backtrack along paths (A4).*

Proof. Under A4, a path may be assigned to a robot only if idle. Hence, if $q_i^g \neq q_i(t)$, then $\dot{q}_i(t) = \boldsymbol{p}_i(\dot{\sigma}_i(t))$. Also, A4 implies that $\dot{q}_i(t) \geq 0$. Hence, $\dot{\sigma}_i(t) \geq 0$, so livelocks cannot happen by definition. □

B. Global prevention

Theorem 11 suggests the main reason for which coordination based on adjusting the temporal profile of trajectories instead of changing the spatial component (e.g., via re-planning) is usually more effective. Nevertheless, re-planning may be required to deal with contingencies, e.g., to overcome unexpected obstacles on the path. For this purpose, we can relax Theorem 11 as follows:

Theorem 12 (Sufficient condition for livelock prevention)**.** *Let $\boldsymbol{p}_i^{old} \in \mathcal{P}(t_0)$, $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0 + 1)T_c)$. Assume a new path $\boldsymbol{p}_i$ satisfying Theorem 9 and Theorem 3, $\boldsymbol{p}_i^{\text{old}}(1) = \boldsymbol{p}_i(1)$, is computed at time $t_1 \in [\mathrm{k}_1 T_c, (\mathrm{k}_1 + 1)T_c)$, $\mathrm{k}_1 > \mathrm{k}_0$. Also, let $\bar{q}_i = \boldsymbol{p}_i^{\text{old}}(\bar{\sigma}_i(\mathrm{k}_1 - 1))$. Under A4, a sufficient condition for livelock prevention is that $\left(\boldsymbol{p}_i^{\text{old}}\right)^{-1}(\bar{q}_i) \leq \boldsymbol{p}_i^{-1}(\bar{q}_i)$.*

Proof. Condition (i) of Theorem 9 requires $\boldsymbol{p}_i^{\text{old}}(q_i) = \boldsymbol{p}_i(q_i)$ for each $q_i \in \bigcup_{\sigma_i=0}^{\bar{\sigma}_i(\mathrm{k}_0)} \boldsymbol{p}_i^{\text{old}}(\sigma_i)$. Hence, the condition ensures that σ_i does not decrease when the path is updated. □

Note that Theorem 12 may also over-constrain the set of solvable problems, e.g., some static obstacles may be successfully avoided with a longer but livelock-free path. Hence, under A6, to minimise the probability of livelocks occurring, it may be reasonable to exploit re-planning strategies only to deal with static obstacles, but not for the purpose of coordination.

VIII. Experimental validation

We evaluate the proposed strategies from four points of view. Tests 1 and 4.2 investigate the performance achievable with different heuristics in a synthetic and in a realistic scenario respectively. Test 2 provides an empirical validation of Theorems 6 and 7. Test 3 evaluates the time required for computing cycles in D_∞ (global re-ordering) and D_1 (partial re-ordering) when increasing the number of interfering

envelopes (i.e., the graph density). Tests 4.1 and 4.2 investigate the trade-off between complexity vs. completeness of Algorithms 3 and 4 in a benchmark scenario with online path planning (4.1) and in a realistic application which leverages a manually defined roadmap. Selected moments during all experiments are shown in video [44].

a) Setup: All tests use a Java implementation of the coordination algorithm (Algorithm 1), available as open source [47]. We use the simulator back-end presented in [3, 4]. Robot controllers, as well as the conservative models g_i used to check the kinematic feasibility of precedence constraints, assume a trapezoidal velocity profile with maximum velocity $v_i^{\max}$ and constant acceleration/deceleration $u_i^{\text{acc}} = u_i^{\text{dec}} = u_i^{\max}$. Goals are dispatched asynchronously to robots, so that when a robot has reached its current goal, the next one is dispatched. Collision checking is performed in all experiments and results validate the theoretical claim in Theorem 1. To demonstrate the validity of the approach with unreliable communication, uniformly distributed random variables are used for injecting communication delays $\tau_i^{\text{ch}} \in [\tau_{\min}^{\text{ch}}, \tau_{\max}^{\text{ch}}]$. However, we do not simulate message loss ($\eta = 0$), as this may introduce bias in the results [4].

A. Test 1: Performance with different heuristics

In this test, we investigate how different heuristics (listed in Table III) may affect time to mission completion, and liveness. Towards this aim, 10 robots are required to perform 10 forward and 10 backward missions along paths with the same shape. Paths have been computed off-line (see Fig. 5(a)) and simulations were run 3 times, randomising the assignment of starts and goals among robots, while maintaining the same order of goal dispatching between the robots. Also, a constant channel delay is used. The choice of constant delay, similar paths, and uniform timing of goal posting for each robot is aimed at removing all factors from the simulation that could affect the results (other than the choice of heuristic). Blocking is avoided a priori since the infrastructure is well-formed. Also, deadlocks are prevented via local re-ordering (Algorithms 4 and 5) which, in this scenario, ensures they never happen since critical sections do not overlap. Details about the setup and results are reported in Table IV.

Results are summarised as follows: heuristics based on strict hierarchies (IDs and FCFS) may lead to "useless" waiting for robots with lower priorities, as highlighted by the maximum peak values in Table IV. Conversely, the distance heuristic keeps this peak low, providing more fair access to critical sections (as shown by the lower standard deviation).

B. Test 2: Experimental validation of Theorems 6 and 7

In this simulation, we give a further validation of Theorems 6 and 7. Two static totally ordering heuristics (IDs and FCFS) were tested while allowing goals to be asynchronously posted. Experiments were run in the simulated warehouse-like environment shown in Fig. 5(b) and satisfying Definition 2. Paths are computed online using the sampling-based motion planner RRTConnect [48] – so that the geometry of critical sections is unpredictable – and each robot is required to perform 10 forward and 10 backward missions between two preassigned locations. The scenario was specifically designed to increase the possibility of deadlocks, since both the map and the path planning induce complex, overlapping critical sections.

To confirm the generality of Theorem 6 with respect to transmission delays, both the cases of reliable communication ($\eta_i = 0$ and $\tau_i^{\text{ch}} = 0$ for every i) and of constant channel delay ($\eta_i = 0$ and $\tau_i^{\text{ch}} = 500$ ms for every i) are considered with 5 runs for each case. As expected, the occurrence of deadlocks in the 10 runs was equal to 0% (all the simulations ended without deadlocks) when FCFS was used; conversely, all simulations ended with deadlocks when using the IDs heuristic (see [44]).

C. Test 3: Complexity of computing cycles.

This test is designed to provide an empirical evaluation of the complexity required to compute cycles both in D_∞ ($O(2^{n \log n})$ in the worst case, see Section VI-B1), and in D_1 ($O(n^2)$ in the worst case, see Section VI-C1). The former is evaluated by measuring the time required by executing lines 21 and 27 in Algorithm 3 when incrementally adding edges to build the graph. We compare two realisations of the graph-building routine, one which computes cycles incrementally and one which does not. Results are shown in Table V and highlight the practical limitation of the approach for graphs with size of the largest connected component greater than 10 (all tests which may lead to $|V_1| > 10$ ran out of memory before achieving a graph density equal to 1). Note that the complexity of computing cycles is only partially related to the spatial distribution of the set $\mathcal{C}$ since $C \in \mathcal{C}_{ij} \implies e_{ij} \in D_\infty$. The analysis provides a practical strategy to design paths/assign missions to lower such complexity (see also Section VIII-D2).

Note that the measured time to compute cycles in D_1 when considering the worst case (i.e., all the robots paired) with $|V_1| = 1000$ was lower than 0.15 sec (avg. 0.137 sec, max. 0.142 sec, svd. 0.042 sec in 3 simulations).

D. Test 4: Performance with different deadlock prevention/repair strategies in benchmark and realistic scenarios

In this test we investigate the efficacy of the proposed strategies in a benchmark scenario (see Fig. 5(c)) as well as in a realistic application (see Fig 6). The comparison considers average time and rate of mission completion, and computational overhead. The latter is measured by the two metrics $T_{\text{rev}}/\mathcal{C}$ and T_{rev}/T_c with T_{rev} being the time required to perform line 2 in Algorithm 1. The first measures the cost of updating precedence constraints per critical section, while the second measures the proportion of T_c spent updating paths and revising precedence constraints.

1) Benchmark scenario: The setup is similar to the one described in Section VIII-B. Details and results are listed in Table VI. In particular, the proposed strategies for deadlock

(a) avoidance, that is, the original algorithm proposed in [4] with the FCFS heuristic (col. 2);
(b) global prevention, that is, Algorithm 3 (col. 3);
(c) local prevention/repair, that is, Algorithm 4 either when Algorithms 5 and 7 are used jointly (col. 4) or on their own (col. 5–7)

are compared in a test involving 10 robots. The distance heuristic is used in cases (a) and (b).

Heuristic	Description	Properties
First Come First Served (FCFS)	$i \prec_{\mathrm{h}} j$ if $\boldsymbol{p}_j$ is accepted before $\boldsymbol{p}_j$	dynamic, totally ordering
IDs	$i \prec_{\mathrm{h}} j$ if $i < j$	static, totally ordering
RANDOM	randomly choose either $i \prec_{\mathrm{h}} j$ or $j \prec_{\mathrm{h}} i$	dynamic, not totally ordering
DISTANCE	$i \prec_{\mathrm{h}} j$ if $\ell_i^C - \sigma_i > \ell_j^C - \sigma_j$ or $(\ell_i^C - \sigma_i = \ell_j^C - \sigma_j) \wedge i < j$	dynamic, not totally ordering

TABLE III: Heuristics used in the tests.

(a) (b) (c)

Fig. 5: **a.** A snapshot of Test 1 (arrows between robots indicate precedence constraints). **b.** Corridor environment used in Test 2. **c.** Random environment used in Test 4.1.

$N = 4$, $\max_{\mathrm{k}}(\lvert\mathcal{C}_i(\mathrm{k})\rvert) = 16$		FCFS	IDs	Random	Distance
Normalised time (sec) - single rob. avg. 17.9 sec	avg.	1.27	1.31	1.26	1.17
	max	2.35	3.03	2.70	2.25
	std	0.29	0.46	0.35	0.19
Avg. nonl. stat. & deadlocks		0, 0	0, 0	19, 0	0, 0
$N = 8$, $\max_{\mathrm{k}}(\lvert\mathcal{C}_i(\mathrm{k})\rvert) = 32$		FCFS	IDs	Random	Distance
Normalised time (sec) - single rob. avg. 31.0 sec	avg.	1.13	1.18	1.29	1.06
	max	1.59	3.68	1.84	1.59
	std	0.14	0.44	0.22	0.09
Avg. nonl. stat. & deadlocks		0, 0	0, 0	39, 0	0, 0

Parameters: $T_c = 2$s, $T_i = 0.03$s, $v_i^{\max} = \pm 6$m/s, $u_i^{\max} = \pm 4$m/s^2, $\eta = 0$, $\tau_i^{\max} = \tau_i^{\min} = 0.5$s, footprints: 0.5×0.5 m^2. N: spatial period for the sinusoidal paths.

Performance has been evaluated on an Intel Core i7-6700 CPU 3.40GHz $\times$ 8 processor, 15.6 GiB, and refers to the current implementation [47] with line 2 of 1 implemented according to Section VI-C1 (partial re-ordering).

TABLE IV: Test 1: performance with different heuristics (defined according to Table III).

Results are summarised as follows. (a) The FCFS heuristic ensures the absence of nonlive sets but has the worst performance in terms of time to completion. (b) Algorithm 3 ensures liveness. Also, in this scenario, its mean computational overhead is comparable with other strategies. However, $T_{\mathrm{rev}}/T_c > 1$ may prevent **P1** from holding (since the lookahead Δ_i^{stop} assumes that Algorithm 1 executes each cycle within T_c). A proper tuning of the coordination period T_c (while considering the maximum number of interacting envelopes — see Test 3) may overcome this issue. (c) Algorithm 4 using jointly partial re-ordering and re-planning provides the best trade-off between mission time and computational overhead, and shows the practical ability to maintain liveness. Conversely, both partial re-ordering and re-planning may not prevent deadlocks when used on their own.

2) Realistic scenario: The proposed algorithms are now compared in a 40-robot scenario elicited via our ongoing collaboration with industrial partners[10]. The environment is an underground mine, shown in Fig 6. The application relies on a roadmap with paths computed via Bézier curves between manually defined waypoints. 40 Load-Haul-Dump vehicles (LHDs) are required to perform 10 missions, each consisting of scooping up material at fixed draw points, and transporting the material through a system of tunnels to a specific dump point. In our setup, we consider 4 LHDs for each tunnel, which results in a maximum of 20 intersecting envelopes. The limited manoeuvring capabilities of LHDs in the narrow tunnels prevents an effective use of the re-planning strategy to deal with deadlocks. Therefore, Algorithm 4 was run while enabling only partial re-ordering. Results of the simulations are shown in Table VII and are in accordance with the ones of Tests 1, 2, 3 and 4.1. Specifically, (i) the theoretical claim in Theorem 5 is confirmed: the FCFS heuristic ensures the absence of nonlive sets without requiring algorithms for deadlock prevention/recovery. Also, the totally ordering heuristic IDs is not able to prevent nonlive sets (see results related to IDs + Algorithm 4 in the table). (ii) The distance heuristic results in less nonlive sets being detected (and hence in a higher number of heuristically decided orders), and in lower times for mission completion. (iii) Partial re-ordering, while less computationally demanding, forfeits completeness (13% mission completion with the IDs heuristic). (iv) The global re-ordering strategy ensures liveness. However, as in Test 4.1, $T_{\mathrm{rev}}/T_c > 1$ may prevent **P1** from holding; either fewer robots, or a greater T_c may prevent this issue.

IX. Conclusions

We have formalised a centralised algorithm for coordinating generic (possibly heterogeneous) multi-robot systems

[10]Newcrest mining (https://www.newcrest.com) and Epiroc (https://www.epiroc.com/).

Number of robots (i.e., $\lvert V_\infty \rvert$)	10		20		40	
Density* of $D_\infty = (V_\infty, E_\infty)$	90, 90		191, 380	209, 380	391, 1560	428, 1560
In-degree $v_i \in V_\infty$ (min, max)	9, 9		9, 10	10, 11	10, 10	10, 11
Out-degree $v_i \in V_\infty$ (min, max)	9, 9		0, 19		0, 39	
# cycles detected (max, avg.)	1.1e6, 7.5e4		2.9e6, 6.4e4	1.1e7, 3.5e5	2.0e6, 2.3e4	1.01e7, 1.68e5
Time to add a new $e_{ij} \in E_\infty$ (sec):	(incr.)	(non-incr.)	(incr.)	(non-incr.)	(incr.)	(non-incr.)
− Total (max, avg.)	5.6, 0.28	0.447, 0.35	11.8, 0.23	11.3, 0.32	13.9, 0.11	12.9, 0.16
− Detect cycles (max, avg.)	1.67, 0.05	0.447, 0.35	1.07, 0.02	11.3, 0.31	3.05, 0.02	12.9, 0.16
− Update $\mathcal{L}$ (max, avg.)	5.1, 0.23	-	10.7, 0.21	-	10.8, 0.09	-
Time to delete an $e_{ij} \in E_\infty$ (sec):	(incr.)	(non-incr.)	(incr.)	(non-incr.)	(incr.)	(non-incr.)
− Total (max, avg.)	1.76, 0.08	0.01, 5e-4	12.8, 0.11	8e-3, 2e-4	9.42, 0.05	0.01, 1e-4
− Update $\mathcal{L}$ (max, avg.)	1.76, 0.08	-	12.8, 0.11	-	9.42, 0.05	-

*: achievable before ran out of memory. Performance has been evaluated on an Intel Core i7-5500U CPU @ 2.40GHz × 4 processors, 7.7 GiB and refers to the current implementation [47] of lines 21 and 27 of Algorithm 3.

TABLE V: Complexity of computing cycles in D_∞ while increasing the graph density.

Metric		**FCFS**	**Algorithm 3**	**Algorithm 4**	**Part. re-ordering**	**Stat. re-plan**	**Dyn. re-plan**
Time to mission completion (sec)	avg.	51.0	37.7	39.1	35.0	39.5	44.7
	std.dev.	11.9	8.3	9.9	6.7	10.8	14.6
Rate of mission completion*	avg.	100%	100%	100%	56%	100%	89%
	min	100%	100%	100%	37%	100%	68.9%
$T_{\text{rev}}/\mathcal{C}$ (ms)	avg., max	0.8, 49	2.6, 336	2.0, 47	3.1, 80	5.3, 57	4.4, 30
T_{rev}/T_c	avg., max	8e-3, 0.05	0.01, 4.9	0.01, 0.05	0.02, 0.06	0.01, 0.04	0.02, 0.08
Critical sections analysed	avg. each T_c, tot all tests	26, 7455	22, 5886	31, 6423	30, 3103	28, 6267	26, 5840

Strategy	**Nonlive sets in $\mathcal{T}_1$**
FCFS	Detected: 0.
Global re-ordering	Detected: 0. $\langle m_i, u_j^C \rangle$ updated as $\prec_{\text{h}}$: avg. 18%, std.dev. 14%.
Partial re-ordering + dyn. re-plan	Detected: avg. 25, max. 49. Solved: avg. via re-ordering 73%$^{(b)}$, via re-planning 27%$^{(b)}$ (succ. rate 71%).
Partial re-ordering	Detected: avg. 6.3$^{(c)}$, max. 11$^{(c)}$. Solved: avg. 56%$^{(c)}$, min. 20%$^{(c)}$.
Static re-plan	Detected: avg. 21, max. 23. Solved: 46% (succ. rate 73%).
Dynamic re-plan	Detected: avg. 44, max. 68. Solved: avg. 99%$^{(c)}$ (succ.rate 88%$^{(c)}$).

*while filtering out the ones which were not completed due to the blocking phenomenon.
Parameters: $T_c = 3\text{s}, T_i = 0.03\text{s}, v_i^{\max} = \pm 4\text{m/s}, u_i^{\max} = \pm 3\text{m/s}^2$, $\eta = 0$, $\tau_i^{\text{ch}} \in [0.01, 0.5]\text{s}$, footprints: 0.5×0.5 m^2.

TABLE VI: Test 4.1, performance with different deadlock prevention/recovery strategies in a benchmark environment. Algorithm 3 and all the different versions of Algorithm 4 were run using the distance heuristic. All tests were repeated 3 times to increase statistical validity. **b:** with respect to the total nonlive sets detected in $\mathcal{T}_1$. **c:** data are biased by deadlocks which prevent some missions to be completed.

Heuristic			**FCFS**	**IDs**		**Distance**	
Algorithm for deadlock prev.			none	Alg. 3	Alg. 4	Alg. 3	Alg. 4
Norm. time to mission compl.$^{(a)}$ (sec)		avg., std.dev.	3.96, 1.24	5, 4.64	1.29*, 0.7*	2.20, 2.0	2.28, 1.69
Rate of mission completion		tot	100%	100%	13%	100%	100%
$T_{\text{rev}}/\mathcal{C}$ (ms)		avg., max	12.6, 40	16.2, 89	12.7, 22.9*	24.0, 101	25.6, 90
T_{rev}/T_c		avg., max	42%, 73%	49%, 102%	51%, 62%	25%, 74%	26%, 63%
Critical sections analysed		avg., tot	201, 4399	177, 4073	246*, 306*	80, 2791	82, 2970
Nonlive sets in $\mathcal{T}_1$	Detected	avg.	0	0	29*	0	2
	Solved	avg.	-	-	20*	-	2
	$\langle m_i, u_j^C \rangle$ as $\prec_{\text{h}}$	avg.	100%	11%	-	63%	-

*: simulation ended without completing all the missions. a: with respect to avg. time w/o coordination (avg. 55.8 sec, std.dev. 103.2 sec).
Parameters: $T_c = 6\text{s}, T_i = 0.03\text{s}, v_i^{\max} = \pm 30\text{m/s}, u_i^{\max} = \pm 6\text{m/s}^2$, $\eta = 0$, $\tau_i^{\text{ch}} \in [0.01, 0.5]\text{s}$, footprints: 11.4×3.1 m^2.

A maximum of 3 new missions, with exception of the first T_c, was assigned at each time. Performance has been evaluated on an Intel Core i7-6700 CPU 3.40GHz × 8 processor, 15.6 GiB, and refers to the current implementation [47].

TABLE VII: Test 4.2, performance with different deadlock prevention/recovery strategies in a realistic environment. Notes: Algorithm 4 was run while enabling only partial re-order strategy. No collision was observed.

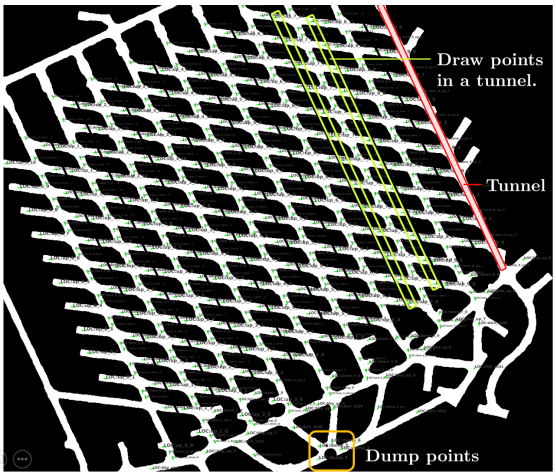


Fig. 6: Test 4: the realistic scenario.

subject to online asynchronous goal assignment and (possibly dynamic) precedences. The approach decouples motion planning from coordination, which is achieved by regulating access to shared parts of the workspace using precedence constraints, and revising these online. Precedences account for user-definable heuristic function(s) as well as kinodynamic constraints on robot motions, thus ensuring safety no matter how speeds are chosen by the robot controllers. While safety was already investigated in [4], in this paper we have focused on liveness. We have identified four factors which may prevent robots from reaching their destinations: blocking, deadlocks, livelocks, and unpredictable disturbances. Among these, only the latter cannot be prevented with an appropriate coordination mechanism.

To overcome the problem of blocking, (i) we extend the concept of well-formed infrastructure given in [6] to generic robots (not necessarily disc-shaped); (ii) we formally prove that blocking never happens if the infrastructure is well-formed; (iii) since this assumption imposes a tight constraint on the set of goals that can be posted, we propose and formally validate two theorems to prevent blocking also when the assumption is relaxed.

While blocking is a function of the set of goals to be assigned, deadlocks are caused by inadequate priority assignments, which impose circular waits among robots. Hence, extending the characterisation of deadlocks given in [3], we formally prove that: (iv) if goals are posted asynchronously, then static hierarchical priorities are not able to ensure liveness; (v) the heuristic which sorts robot priorities according to mission assignment time (First Come, First Served) ensures that deadlocks never happen, both with synchronous and asynchronous goal posting. We then generalise this result to any user-defined heuristic function, by proposing (vi) a global and two local algorithms for deadlock prevention and recovery. We formally prove that the global algorithm ensures deadlock-free motion at the cost of exponential computation in the worst case. Our experimental analysis has shown how all these strategies are feasible in realistically-sized fleets. Receding horizon techniques may be explored in the future to further improve the efficiency of the global approach in large fleets.

The two local algorithms for deadlock prevention and recovery are based on re-ordering precedences and re-planning paths, respectively. These are aimed at further reducing computational complexity, at the price of losing completeness. The algorithms were tested and compared with the global one, both in isolation and in combination. Tests show the practical ability of these algorithms to prevent and recover from deadlocks in the tested scenarios.

Finally, we characterise livelocks, formally proving that if robots are not allowed to drive back along their paths and re-planning is not used (e.g., as with Algorithm 3), then livelocks will never happen.

Further investigations will be devoted to improving the efficacy of re-planning strategies, exploiting global cost functions to compute/select new paths, or partially coupled methods [26].

References

[1] P. R. Wurman, R. D'Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[2] *Semantic Robots*, 2019. http://semanticrobots.oru.se.

[3] F. Pecora, H. Andreasson, M. Mansouri, and V. Petkov. A loosely-coupled approach for multi-robot coordination, motion planning and control. In *Proc. 28th Int. Conf. Autom. Planning & Scheduling*, 2018.

[4] A. Mannucci, L. Pallottino, and F. Pecora. Provably safe multi-robot coordination with unreliable communication. *IEEE Robotics and Automation Letters*, 4(4):3232–3239, 2019.

[5] T. Lozano-Perez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

[6] M. Čáp, P. Novák, A. Kleiner, and M. Selecký. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Trans. Autom. Sci. Eng*, 12(3):835–849, 2015.

[7] D. Bareiss and J. Van den Berg. Generalized reciprocal collision avoidance. *Int. J. Robot. Research*, 34(12):1501–1514, 2015.

[8] P. Spirakis and C. K Yap. Strong np-hardness of moving many discs. *Infor. Process. Lett.*, 19(1):55–59, 1984.

[9] M. Čáp, J. Vokřínek, and A. Kleiner. Complete decentralized method for on-line multi-robot trajectory planning in well-formed infrastructures. In *Proc. 25th Int. Conf. Autom. Planning & Scheduling*, 2015.

[10] S. Akella and S. Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proc. 2002 IEEE International Conf. Robotics & Aut. (Cat. No. 02CH37292)*, volume 1, pages 624–631. IEEE, 2002.

[11] F. Pecora, M. Cirillo, and D. Dimitrov. On mission-dependent coordination of multiple vehicles under spatial and temporal constraints. In *2012 IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pages 5262–5269. IEEE, 2012.

[12] M. Čáp, J. Gregoire, and E. Frazzoli. Provably safe and deadlock-free execution of multi-robot plans under delaying disturbances. In *2016 IEEE/RSJ Int. Conf. Intell. Robots & Syst. (IROS)*, pages 5113–5118. IEEE, 2016.

[13] A. Coskun and J. M O'Kane. Online plan repair in multi-robot coordination with disturbances. In *2019 International Conf. Robotics & Autom. (ICRA)*, pages 3333–3339. IEEE, 2019.

[14] Zhi Yan, Nicolas Jouandeau, and Arab Ali Cherif. A survey and analysis of multi-robot coordination. *International J. Advanced Robotic Syst.*, 10(12):399, 2013.

[15] P. A O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *ICRA*, volume 89, pages 484–489, 1989.

[16] C. Tomlin, I. Mitchell, and R. Ghosh. Safety verification of conflict resolution manoeuvres. *IEEE Trans. Intell. Transp. Syst.*, 2(2):110–120, 2001.

[17] L. Pallottino, E. M Feron, and A. Bicchi. Conflict resolution problems for air traffic management systems solved with mixed integer programming. *IEEE Trans. Intell. Transp. Syst.*, 3(1):3–11, 2002.

[18] L. Pallottino, V. G Scordio, A. Bicchi, and E. Frazzoli. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Trans. Robotics*, 23(6):1170–1183, 2007.

[19] L. Chen and C. Englund. Cooperative intersection management: A survey. *IEEE Trans. Intell. Transp. Syst.*, 17(2):570–586, 2015.

[20] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljević, and S. Bogdan. Decentralized control of multi-agv systems in autonomous warehousing applications. *IEEE Trans. Autom. Sci. Eng.*, 13(4):1433–1447, 2016.

[21] V. Digani, L. Sabattini, C. Secchi, and C Fantuzzi. Ensemble coordination approach in multi-agv systems applied to industrial warehouses. *IEEE Trans. Autom. Sci. Eng.*, 12(3):922–934, 2015.

[22] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robot. Autom. Mag.*, 4(1):23–33, 1997.

[23] J. Van Den Berg, S.J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

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

[25] J. E Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the" warehouseman's problem". *The International J. of Robotics Research*, 3(4):76–88, 1984.

[26] G. Wagner and H. Choset. M*: A complete multirobot path planning algorithm with performance bounds. In *2011 IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pages 3260–3267. IEEE, 2011.

[27] M. Erdmann and T. Lozano-Perez. On multiple moving objects. *Algorithmica*, 2(1-4):477, 1987.

[28] J. Van Den Berg and M. Overmars. Kinodynamic motion planning on roadmaps in dynamic environments. In *2007 IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pages 4253–4258. IEEE, 2007.

[29] J. P Van Den Berg and M. H Overmars. Prioritized motion planning for multiple robots. In *2005 IEEE/RSJ Int. Conf. Intell. Robots & Syst.*, pages 430–435. IEEE, 2005.

[30] M. Bennewitz, W. Burgard, and S. Thrun. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Rob. & Aut. Syst.*, 41(2-3):89–99, 2002.

[31] K. Kant and S. W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International J. of Robotics Research*, 5(3):72–89, 1986.

[32] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International J. of Robotics Research*, 24(4):295–310, 2005.

[33] H. Andreasson and other. Autonomous transport vehicles: where we are and what is missing. *IEEE Robot. Autom. Mag.*, 22(1):64–75, 2015.

[34] M. Mansouri, B. Lacerda, N. Hawes, and F. Pecora. Multi-robot planning under uncertain travel times and safety constraints. In *Proc. 28th Int. Conf. on Artificial Intell., IJCAI-19*, pages 478–484. International Joint Conf.s on Artificial Intell. Organization, 7 2019.

[35] G. R. de Campos, P. Falcone, R. Hult, H. Wymeersch, and J. Sjöberg. Traffic coordination at road intersections: Autonomous decision-making algorithms using model-based heuristics. *IEEE Intell. Trans. Syst. Mag.*, 9(1):8–21, 2017.

[36] J. Gregoire, S. Bonnabel, and A. De La Fortelle. Priority-based intersection management with kinodynamic constraints. In *2014 European Control Conf. (ECC)*, pages 2902–2907. IEEE, 2014.

[37] J. Gregoire. *Priority-based coordination of mobile robots*. PhD thesis, 2014. arXiv preprint arXiv:1410.0879.

[38] R. Ghrist, J. M O'Kane, and S. M LaValle. Computing pareto optimal coordinations on roadmaps. *The International J. of Robotics Research*, 24(11):997–1010, 2005.

[39] E. G Coffman, M. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys (CSUR)*, 3(2):67–78, 1971.

[40] H. Andreasson, J. Saarinen, M. Cirillo, T. Stoyanov, and A. J Lilienthal. Fast, continuous state path smoothing to improve navigation accuracy. In *2015 IEEE International Conf. Robotics & Autom. (ICRA)*, pages 662–669. IEEE, 2015.

[41] R. Ghrist and S. M Lavalle. Nonpositive curvature and pareto optimal coordination of robots. *SIAM J. Control & Optimization*, 45(5):1697–1713, 2006.

[42] J. Gregoire, S. Bonnabel, and A. de La Fortelle. Optimal cooperative motion planning for vehicles at intersections. *arXiv preprint arXiv:1310.7729*, 2013.

[43] M. P. Fanti and M. Zhou. Deadlock control methods in automated manufacturing systems. *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, 34(1):5–22, 2004.

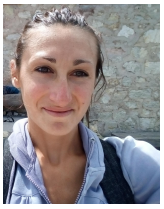
[44] Anna Mannucci, Lucia Pallottino, and Federico Pecora. *Provably Safe and Live Multi-Robot Coordination, Youtube*, 2020. https://youtu.be/dFwf8gkItYU.

[45] D. B Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Computing*, 4(1):77–84, 1975.

[46] Theodore P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[47] F. Pecora, A. Mannucci, and C.S. Swaminathan. *Robot- and Motion-planning agnostic online coordination for multiple robots*, 2019. https://github.com/FedericoPecora/coordination_oru, version udp-live-V1.5.

[48] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.



Anna Mannucci is currently Research Assistant at the Multi-Robot Planning and Control Laboratory at Örebro University. She earned her Master's degree in Robotics and Automation in 2016 (Bachelor in Electronic in 2013) and the Ph.D. in Robotics and Automation in 2020 at the University of Pisa. Together with Federico Pecora, she is one of the principal developers and maintainers of the coordination_oru library [47], a general-purpose tool for integrated motion planning, coordination, and control for fleets of autonomous robots. The library is used in the ILIAD European project and in industrial research and development by third parties such as Volvo, Scania, and Epiroc.

Lucia Pallottino is currently Associate Professor at the Centro di Ricerca "E. Piaggio" and the Dipartimento di Ingegneria dell'Informazione at the University of Pisa. She received the "Laurea" degree in Mathematics in 1998 and the Ph.D. in Robotics and Industrial Automation in 2002. She is Deputy Director of Centro di Ricerca "E. Piaggio". She is Associate Editor of the IEEE Robotics and Automation Letters (since 2017) and has been Associate Editor of the IEEE Transaction on Robotics (2014-2017). Her main research interests within Robotics are in motion planning and optimal control of multi robot systems and coordination of multi–robot vehicles.

Federico Pecora is currently Associate Professor in Computer Science at Örebro University, where he leads the Multi-Robot Planning and Control Laboratory. He is a graduate of Computer Science Engineering at the University of Rome "La Sapienza", where he earned his Ph.D. in 2007. His research interests lie at the intersection of Artificial Intelligence and Robotics, focusing specifically on constraint-based reasoning, automated planning, and search techniques for hybrid reasoning. Most of his recent work deals with the use of these methods for plan-based robot control, multi-robot coordination, and the integration of these with robot motion planning and control.

Appendix C Heuristic search methods for combined task assignment, motion planning and coordination in fleets of mobile robots (MSc thesis, 2020)

Master's Degree Course in Robotics and Automation Engineering
Department of Information Engineering
University of Pisa

Heuristic search methods for combined task assignment, motion planning and coordination in fleets of mobile robots

Supervisors:
Prof.ssa Pallottino, Lucia
Prof. Pecora, Federico
Dott.ssa Mannucci, Anna

Candidate:
Forte, Paolo

Academic Year 2019/2020

Contents

List of Figures

Abbreviations

AVGs Automated Guided Vehicles

BB Branch and Bound

BIP Binary integer programming

LP Linear Programming

MILP Mixed integer linear programming

MRTA Multi-Robot Task Allocation

NLP Nonlinear programming

QP Quadratic Programming

Abstract

This work [1] considers the Task Assignments Problem for fleets of Automated Guided Vehicles (AVGs) while considering the subsequent path planning problem and the coordination problem. Each assignment of a robot to a task has a certain cost; the estimated costs are interrelated, however, the overall cost incurred by several robots cannot be estimated simply as the sum of all costs, as it much account for the degree in which robots interfere with each other in carrying out their tasks. For instance additional penalization costs are incurred by overlapping paths. This work evidences that the general The Multi-Robot Task Allocation (MRTA) problem is NP-hard, and investigate specialized sub-instances with particular cost structures. Sequential application of task assignment and path planning often gives rise to pathological situations, such as deadlocks, in which AGVs block each other, thus preventing tasks completion. The MRTA problem can vary widely depending on the characteristics of the robots in the fleet and of the application context. Goals can be shared or individual. Robots may be aware of each other's tasks or not; tasks may or may not require interaction between robots; the fleet can be a centralized or distributed system; in this study, a centralized system is considered. A systematic algorithm is proposed, that finds a solution in a reasonable time both on small instances (if interference among robots is considered) and on big instances (if it is not considered). Aiming at larger problems, some changes are introduced that allow to find a sub-optimal assignment quickly even for problems of considerable size. This algorithm is capable of finding a solution that avoids the deadlock situation considering the interference function. Task deadlines are also considered during the allocation. Simulations are performed on maps of real industrial environments in order to compare the proposed method with traditional task assignment algorithms.

[1] carried out at *Örebro University*

Acknowledgements

Volevo ringraziare la professoressa Lucia Pallottino che ha reso possibile questa esperienza di tesi all'estero che ritengo molto utile per la mia formazione come ricercatore. Volevo inoltre ringraziare il professore Federico Pecora che mi ha seguito pazientemente durante tutto il mio lavoro di tesi, facendo accrescere la mia passione per la ricerca e permettondo inoltre di migliorare la mia formazione scolastica nel campo della robotica. Volevo ringraziare inoltre la dottoressa Anna Mannucci che mi ha supportato, insieme al professore Pecora, durante tutto il lavoro di tesi. Volevo inoltre ringraziare tutte le persone del MRPC lab che mi hanno aiutato durante il periodo trascorso all'estero. Inoltre volevo ringrazire la mia famiglia e i miei amici che mi hanno sempre sostenuto e supportato in tutte le decisioni che ho preso durante tutto il percorso di laurea, fino alla preparazione della tesi.

Chapter 1

Introduction

Consider a fleet of n^{idle} possibly heterogeneous idle robots moving in a shared environment and a set of m tasks to be accomplished. The Multi-Robot Task Allocation problem (MRTA) is the problem of deciding which robots in the team should perform which tasks in order to maximize/optimize the overall fleet performance according to a global cost function subject to application-specific constraints. The problem applies to both to heterogeneous and homogeneous fleets of robots. In the former, the different robot capabilities may be opportunely modelled into the problem by constraints to better exploit heterogeneity (i.e., the fact that a robot can perform a task or not should be considered in the MRTA problem). Thus, this problem can be seen as an optimal assignment problem, that can be a maximization or minimization problem, based on the desired performance.

Several works can be found in the literature addressing the MRTA problem, spanning from solving the problem while taking into account the path planning problem in order to minimize conflicts and deadlock [2] where, the vehicles are constrained to move along a roadmap and conflicts are considered utilizing a so called conflict graph. Other work has considered to the presence of shared resources (robots may choose one of the shared resourced to perform a task) [3], where the proposed optimization function is divided into two parts: the first one is interference-free while the second one takes into account interference among AGVs. Giordani and colleagues address the MRTA problem in case of a decentralized system [4], where the solution to the assignment problem is reached without any common coordinator or shared memory of the system. Sabattini and colleagues used an estimation of the state of the traffic for defining, dynamically, weights for the allocation [5].
Turpin and Michael dynamically assign priority based on possibly conflicting tasks [6] and [7]. In other words, for each AVG of the fleet the optimal path to reach a final goal position is computed with the Dijkstra's algorithm while tasks are assigned with the Hungarian method. Pallottino et all. address the problem as a distributed optimization problem and for solving it, they propose a use a sub gradient method[8]. An example of the MRTA problem is shown in Figure 1.1.

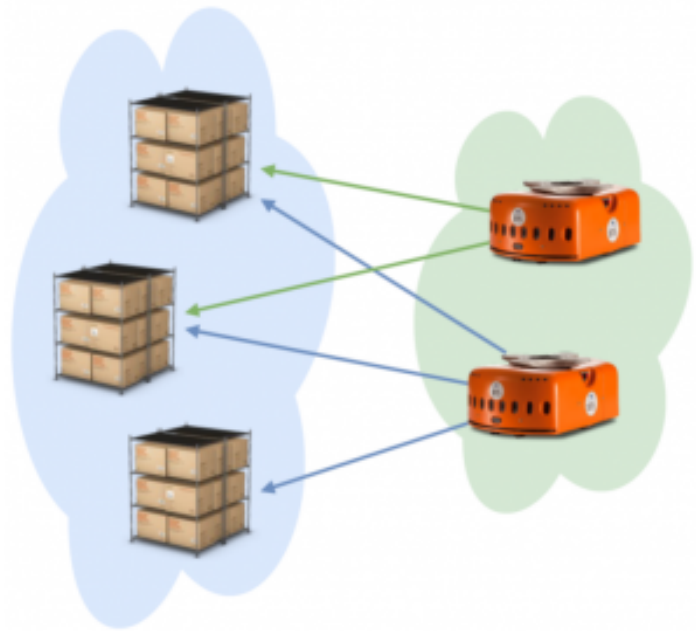


Figure 1.1: Task Allocation Problem

The contributors of this thesis can be summarized as follows:

- the combined task assignment, motion planning and coordination problem is formulated;
- systematic search algorithm to solve the problem is developed,
- evaluating the algorithm with a fleet of simulated robots;
- the developed systematic algorithm is compared with local search methods for solving the same problem.

This thesis is organized as follows: in Chapter 2 background on MRTA problems and on optimization problem is summarized. Chapter 3 provides the mathematical formulation of the problem. Chapters 4 and 5 describe the cost functions that are considered in the problem. The systematic algorithm developed during this work is described in Chapter 6 and results obtained comparing the developed systematic algorithm with local search methods for solving the same problem are reported in Chapter 7.

Chapter 2

State of the Art

The aim of this chapter is to provide a theoretical background on the problem addressed in this thesis. The chapter is organized as follows. In Section 2.1 a Taxonomy of MRTA problems is proposed. In Section 2.2 an introduction to unconstrained and constrained optimization is provided. In Section 2.3 some methods to solve an optimization problem are described and in Section 2.4 some techniques to relax constraints in optimization problems are reported. Beyond the common algorithms that can be used to solve optimization problem, a few alternative algorithms are described in Section 2.5. Finally, in Section 2.6 some available software for solving optimization problems are introduced and compared.

2.1 Taxonomy

In an industrial environment, autonomous vehicles or more in general all robots can perform a large variety of tasks depending on their characteristics. For instance, an autonomous vehicle can carry materials across a warehouse, put materials on conveyor belts, while a robotic arm may be able to accomplish tasks such as welding, screwing, assembling, etc . In a scenario with different task types, it is natural to try to assign a task to the most appropriate robot. Different MRTA problems can be defined according to the system type. Specifically, the system can be a centralized or distributed , where centralized means that there is a leader that has all information about the system (e.g., the locations of robots, robots goals) and plans the actions of the other robots. Conversely in a distributed system all robots have planning capabilities and relevant information is shared between all robots in the fleet. Also, the system may be different according to different robot types (robots can be heterogeneous or homogeneous), or for shared information between robots (such as own state). The MRTA problem can also differ for the target type where the target accounts for task characteristics (e.g., individual or shared, one or more task to each robot). Tasks can be discrete (e.g. deliver a package to a specific room) or continuous (e.g. monitor the building entrance for intruders) but they also may vary in several other ways, including timescale, complexity, and specificity [9].

To help organize MRTA problems, a taxonomy was firstly proposed in [10] and then extended in [11] to consider different types of the Task Assignment Problem. It shows how many such problems can be viewed as instances of other already studied optimization problem. MRTA problems can be classified considering a three axes criteria (see Figure 2.1).

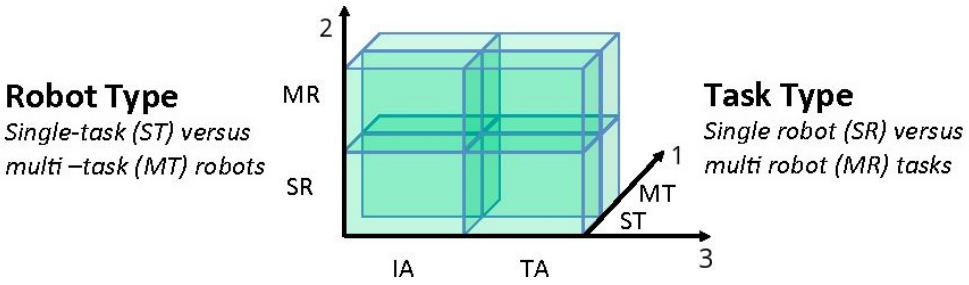


Figure 2.1: MRTA Taxonomy

In the first axis, Single Task (ST) vs. Multi Task (MT), MRTA problems are classified according to the number of tasks that can be accomplished at the same time by each robot. In ST problems each robot can execute at most one task at each time (e.g., a robot is asked to reach a specific position), while in MT problems with more than one task may be executed simultaneously by a robot (e.g., an anthropomorphic robot may be asked to open a door while maintaining its centre of mass in a specific position). Problems in which each task requires exactly one robot to accomplish it and problems in which some tasks may require multiple robots are distinguished along the second axis, single robot tasks (SR), for example lift light load, versus multi-robot tasks (MR), where more robots cooperate to lift a heavy load. Finally, in the third axis, instantaneous assignment (IA) versus time-extended assignment (TA), distinguishes between problems where tasks are allocated to robots instantaneously (i.e., the solution is computed without considering future allocations) and problems which account for both current and future allocations at the same time. In the first case, the available information on the robots, the tasks, and the environment permits only an instantaneous allocation of tasks to the robots, while in TA problems each robot is allocated several tasks which must be executed according to a given schedule (more information is available, such as the set of all tasks that will need to be assigned, or a model of how tasks are expected to arrive over time, how tasks must be executed).

Various types of tasks performed by agents can be distinguished depending, for example, on the number of robots required to accomplish a task [11]. Tasks that can be executed by a single robot are called *elemental* task. If a difficult task can be divided or decomposed into multiple easier sub-tasks, it is a *decomposable* task. If there is a single way to decompose the task into easier sub-tasks, this task is defined as *compound* task; conversely, if there are multiple ways to decompose the task, the task is defined a *complex* tasks. Different parts of a compound task may be allocated to different agents but also to the

same agents, in this case the task is defined as *decomposable simple* task. In Figure 2.2 different tasks types are illustrated.

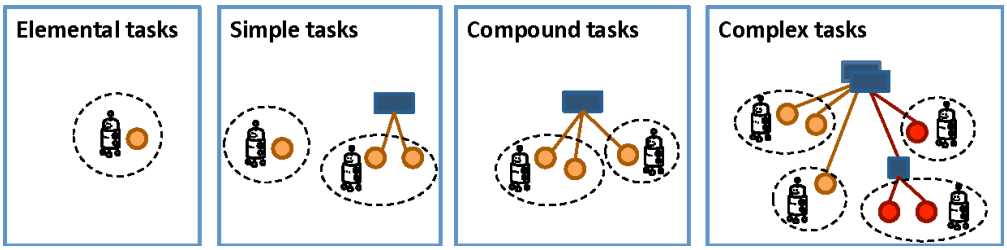


Figure 2.2: Possible Types of Tasks

Following this taxonomy, an MRTA problem is defined by a triple of two letter abbreviations. The simplest problem is Single-Task Robots, Single-Robot Tasks, Instantaneous Assignment (ST–SR–IA). The ST–SR–IA can be posed as an Optimal Assignment Problme (OAP) as follows: Given n^{idle} idle robot and m tasks, each task can be assigned only to one idle robot and each robot can perform only one task at a time. Furthermore, in order to consider the capability of robots to accomplish tasks, typically a binary variable that assumes values $\{0, 1\}$ is introduced, where 0 means that the robot is incapable of performing the task, while 1 means that it is capable.

2.2 Optimization Problem

Since each MRTA can be posed as an Optimal Assignment Problem (OAP), linear programming method (LP) can be used to find the best outcome whenever the objective function, as well as equality and inequality constraints, are all linear. An optimization problem consists in optimizing (maximizing or minimizing) a function subject to constraints and relative to some set, i.e.,

$$\text{optimize } f(x) \text{ with } x \in \mathcal{S} \tag{2.1}$$

where $x \in \mathcal{X}$ is the *optimization variable* with domain $\mathcal{X}$, $f : \mathcal{X} \to \mathbb{R}$ is the *objective function*, which is required to be at least continuous and differentiable, and $\mathcal{S} \subset \mathcal{X}$ is the *feasible set* containing all admissible choices for x. Considering the nature of the set $\mathcal{X}$, each optimization problem can be classified as

- **Continuous Optimization**: x can take continuous values in $\mathcal{X} \subseteq \mathbb{R}^n$; $\mathcal{S} \subset \mathcal{X}$ entails that the optimization problem is constrained, whereas $\mathcal{S} = \mathcal{X}$ entails that the problem is unconstrained;
- **Discrete Optimization**: the set $\mathcal{X}$ is countable; specifically, in *integer programming* problems $\mathcal{X} \subseteq \mathbb{Z}^n$, while in *boolean optimization* $\mathcal{X} \subseteq (0, 1)^n$;

- **Mixed Problems**: where some optimization variables are continuous and others are discrete.

Usually, a finite number of equality and inequality relations are used to define the feasible set $\mathcal{S}$. Given the functions $g_i : \mathcal{X} \to \mathbb{R}$, $i \in \{1, \ldots, m\}$, it is possible to define a constrained feasible set as follows:

$$\mathcal{S} = \{x \in \mathcal{X} \mid g_1(x) \leq 0, \ldots, g_m(x) \leq 0\} \tag{2.2}$$

where each $g_i(x) \leq 0$ is a **constraint**. Any constraint in the form $g_i(x) \geq 0$ can be also expressed as $-g_i(x) \leq 0$. Also, each equality constraint $g_i(x) = 0$ can be transformed a pair of inequality constraints $g_i(x) \leq 0$ and $-g_i(x) \leq 0$.

An optimization problem is defined as a **Linear Program (LP)** or **linear programming problem** if *all* the objective and constraint functions are linear. Conversely, if at least one of the functions is not linear, the optimization problem is a **Non-Linear Program (NLP)** or **nonlinear programming problem**.

2.2.1 Preliminary Definitions

According to the notation introduced, a minimization problem is defined as

$$\text{minimize } f(x) \text{ with } x \in \mathcal{S}, \tag{2.3}$$

and its optimal solution coincides with the optimal solutions of the maximization problem (that is the dual problem)

$$\text{maximize } -f(x) \text{ with } x \in \mathcal{S}$$

Henceforth, without loss of generality, let assume each optimization problem to be defined as in (2.3).

Definition 1 *A minimization problem is **unfeasible** if $\mathcal{S} = \emptyset$, that is, there are no admissible choices.*

Definition 2 *A minimization problem is **unbounded** if for any value $M > 0$ a point $x \in \mathcal{S}$ exists such that $f(x) < -M$.*

Definition 3 *A point x^* is an **optimal solution** of a minimization problem (or global minimizer) if $f(x^*) \leq f(x) \ \forall x \in \mathcal{S}$.*

If $\mathcal{S}$ is an open set, this point is called an *unconstrained minimizer*; otherwise, it is a *constrained minimizer*. A minimization problem admits a solution if there exists a global minimizer $x^* \in \mathcal{S}$ and the value $f(x^*)$ is called the *optimal value*. Since the global minimizer can be difficult to compute, most algorithms are able to find a local minimizer, that is the smallest value of f only in a neighbourhood. Given a distance ρ, a point $\bar{x} \in \mathcal{S}$ is a local minimizer if there exists a neighbourhood $\mathcal{N}(\bar{x}, \rho)$ of $\bar{x}$ such that

$$f(\bar{x}) \leq f(x) \ \forall x \in \mathcal{N} \cap \mathcal{S}.$$

Furthermore, a point $\bar{x} \in \mathcal{S}$ is defined as a *strict local minimizer* if there exists a neighbourhood $\mathcal{N}(\bar{x}, \rho)$ of $\bar{x}$ such that

$$f(\bar{x}) < f(x) \;\; \forall x \in \mathcal{N} \cap \mathcal{S}, \mathrm{x} \neq \bar{x} \tag{2.4}$$

2.2.2 Unconstrained Optimization

In this kind of problem, the aim is to minimize an objective function that depends on real variables and there are no restrictions on the values of these variables [12]. Many algorithms for unconstrained optimization exist. All algorithms for unconstrained minimization require the user to supply a starting point, which is usually denoted by x_0. The choice of x_0 affects the time for retrieving a good solution. Thus, whenever prior knowledge about the problem is available, the point x_0 should be chosen as a reasonable estimate of the solution. Otherwise, the starting points must be chosen by the algorithm either by a systematic approach or in an arbitrary way.

Starting from x_0, an optimization algorithm generates a sequence of iterates $\{x_k\}$ and terminates when it is not possible to make more progress, or a solution point has been approximated with enough accuracy. In deciding how to move from a point x_k to x_{k+1}, the algorithms use information about the function f at point x_k and also information from some previous iterates $x_0, \ldots, x_{k-1}$, so that $f(x_{k+1}) < f(x_k)$ at each k.

There are two fundamental strategies for moving from the current point x_k to x_{k+1}: the **Line Search** and the **Trust Region** methods.

In the **Line Search** strategy, at each step k the algorithm chooses a direction p_k and searches along this direction to move from the current x_k to the next iterate x_{k+1}. The distance α_k to move along p_k, namely the step length, can be found by approximately solving the following problem

$$\arg\min_{\alpha_k > 0} \;\; f(x_k + \alpha_k p_k) \tag{2.5}$$

By solving the problem (2.5) exactly, it is possible to derive the maximum benefit from the direction p_k, but it may be expensive and is usually unnecessary. Thus, a limited number of trial steps is usually considered, searching for the value of α_k that loosely approximates the minimum of (2.5).

Since at each step $x_{k+1} = x_k + \alpha_k p_k$, the effectiveness of each line search method strictly depends on how both the direction p_k and the step length α_k are computed. In particular, the choice of the step length influences the convergence velocity. Large steps may converge more quickly, but may also overstep the solution or, if the error surface is very eccentric, go off in the wrong direction. A classic example of this is when the algorithm progresses very slowly along a steep, narrow valley, bouncing from one side across to the other. On the other hand, very small steps may go in the correct direction but may require too many iterations [13]. For this purpose, an algorithm such as the Backtracking Line Search as well as convergence conditions have been proposed in the literature to compute α_k opportunely (see Chapter 3 of [12] for details).

The steepest descent direction $-\nabla f_k$ is the most intuitive choice for the search direction. Along with all possible directions from x_k to x_{k+1}, the one that is chosen is the one along which f decreases most rapidly, i.e., $p_k = -\nabla f_k$. This method is called the **steepest descent method**.

Another important search direction method is the **Newton direction** in which the direction is derived from the second order Taylor series approximation of $f(x_k + p)$, that is

$$f(x_k + p) \simeq f_k + p^T \nabla f_k + \frac{1}{2} p^T \nabla^2 f_k p \stackrel{\text{def}}{=} m_k(p)$$

Assuming $\nabla^2 f_k$ to be positive definite, the Newton direction is obtained by finding the vector p that minimizes $m_k(p)$. By setting the derivative of $m_k(p)$ equal to zero, the following equation can be explicitly derived

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k. \tag{2.6}$$

The Newton direction is reliable when the difference between the true function $f(x_k+p)$ and its quadratic model $m_k(p)$ is not too large. The Newton direction can be used in a line search method when $\nabla^2 f_k$ is positive definite; usually, the line search implementations of *Newton Direction* use the unit step ($\alpha = 1$) where possible, and adjust the value of α only if the method does not produce a satisfactory reduction in the value of f. When $\nabla^2 f_k$ is not positive definite, this method cannot be used since both the inverse of $\nabla^2 f_k$ may not exist or $\nabla f_k^T p_k^N$ may not be a descendent direction. Methods that use the Newton direction have typically a fast rate of local convergence, that is typically quadratic; indeed, when the method reaches a neighbourhood of the solution, convergence to high accuracy occurs in few iterations. The main drawback of the Newton direction is the need to calculate the Hessian $\nabla^2 f_k$ that in some cases can be difficult to determine: since the gradient and Newton methods require the inversion of Hessian matrix, if this matrix is semi-definite positive, it cannot be inverted so the problem is badly posed. Another problem can be verifying if the Hessian matrix is definitely positive, but the conditioning number is high (no numerical inversion is possible). That is why typically a **Quasi Newton** search direction method is adopted. This last method does not require computing the Hessian and still attains a superlinear rate of convergence. Instead of calculating the Hessian, an approximation B_k is used; this is updated after each step in order to consider the additional knowledge gained during the step. The quasi Newton search direction is obtained by using the approximation B_k instead of the true Hessian in the Equation 2.6.

Another search method is the **nonlinear conjugate gradient method**. In this method the step p_k is calculated as follows:

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1}$$

where β_k is a scalar that ensures that p_k and p_{k-1} are conjugate. These methods do not attain the fast convergence rates of Newton or quasi-Newton methods, but they have the advantage of not requiring storage of matrices. In general, nonlinear conjugate gradient directions are more effective than the steepest descent direction and are almost as simple to compute. All the search methods discussed so far can be

used directly in a line search framework. They give rise to the steepest descent, Newton, quasi-Newton, and conjugate gradient line search methods. All except conjugate gradients have an analogue in the trust region framework.

In the **Trust Region** strategy, the information about the objective function is used to construct a *model function* m_k whose behaviour near the current point x_k is similar to that of the current objective function. In order to consider that the model m_k of function f is not a good approximation when x is far from x_k, the search for a minimizer is made on a region around x_k. The step p is computed as

$$\arg\min_p m_k(x_k + p), \quad \text{where } x_k + p \text{ lies inside the trust region.} \tag{2.7}$$

Usually, the trust region is a ball defined by $\|p\|_2 \leq \Delta$, where $\Delta \in \mathbb{R}^+$ is the trust region radius. The model m_k in Equations 2.7 is usually defined as a quadratic function of the form

$$m_k(x_k + p) = f_k + p^T \nabla f_k + \frac{1}{2} p^T B_k p$$

where f_k, ∇f_k are chosen to be the function and gradient values at the point x_k, and the matrix B_k is either the Hessian $\nabla^2 f_k$ or an approximation of that. Set $B_k = 0$ and consider the trust region with the Euclidean norm, the trust region subproblem becomes

$$\begin{aligned} &\text{minimize } f_k + p^T \nabla f_k \\ &\text{subject to } \|p\|_2 \leq \Delta_k \end{aligned}$$

It is possible to determinate the step p_k as:

$$p_k = -\frac{\Delta_k \nabla f_k}{\|\nabla f_k\|}$$

This is simply the steepest descent step in which the step length is determined by the trust region radius. Another choice can be performed choosing B_k as the Hessian. Because of the restriction $\|P\|_2 \leq \Delta_k$ the subproblem expressed in Equation 2.7 is guaranteed to have a solution even when the Hessian is not positive definite.

A very important issue in optimization is **scaling** operation. Considering an unconstrained optimization problem, it is defined **poorly scaled** if changes to x in a certain direction produce much larger variations in the values of f than do changes to x in another direction. Some optimization algorithms previously mentioned, such as steepest descent is sensitive to poor scaling; whereas Newton methods are unaffected by it. Usually it is preferable to choose algorithms that are not sensitive to scaling, because they can deal problem in a more robust way.

2.2.3 Constrained Optimization

The general formulation of a constrained optimization problem is

$$\text{minimize}_{x\in\mathbb{R}} \quad f(x) \tag{2.8}$$
$$\text{subject to} \begin{cases} c_i(x) = 0 \quad i \in \mathcal{E}, \\ d_i(x) \geq 0 \quad i \in \mathcal{I}, \end{cases}$$

where f and c_i are smooth, real valued function defined on a subset of $\mathbb{R}^n$ and $\mathcal{E}$ and $\mathcal{I}$ are two set of indices referred to constraints of the problem. The function f is called **objective function**, as before, and the c_i are called **equality constrains** while d_i are the **inequality constraints**. Furthermore, let Ω be the set of points x satisfying the constraints, i.e.,

$$\Omega = \{x | c_i(x), i \in \mathcal{E}; d_i(x) \geq 0, i \in \mathcal{I}\},$$

so that it is possible to rewrite Equation 2.8 as

$$\text{minimize}_{x\in\Omega} \quad f(x) \tag{2.9}$$

As in the previous subsection, two cases can be distinguished: *necessary* and *sufficient* conditions. The first are conditions that must be satisfied by any solution point (under certain assumptions); the second are those that, if satisfied at a certain point x^*, guarantee that x^* is in fact a solution.

Definitions of the different types of local solutions are simple extensions of the corresponding definitions for the unconstrained case, but now in addition some consideration to the feasible points in the neighbourhood of x^* are necessary. In the following some definitions are provided.

Definition 4 *A vector x^* is a **local solution** of the problem defined in 2.9 if $x^* \in \Omega$ and if there is a neighbourhood $\mathcal{N}$ of x^* such that $f(x) \geq f(x^*)$ $\forall x \in \mathcal{N} \cap \Omega$.*

Definition 5 *A vector x^* is a **strict local solution** (**strong local solution**) of the problem defined in 2.9 if $x^* \in \Omega$ and if there is a neighbourhood $\mathcal{N}$ of x^* such that $f(x) > f(x^*)$ $\forall x \in \mathcal{N} \cap \Omega$ with $x \neq x^*$.*

Definition 6 *A point x^* is an **isolated local solution** if $x^* \in \Omega$ and there is a neighbourhood $\mathcal{N}$ of x^* such that x^* is the only local solution in $\mathcal{N} \cap \Omega$. An isolated local solution is also a strict solution, but the reverse is not true.*

Definition 7 *An active set $\mathcal{A}(x)$ at any feasible x consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints i for which $c_i(x) = 0$; this set is defined as follows*

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} | c_i(x) = 0\}$$

*At a feasible point x, the inequality constraint $i \in \mathcal{I}$ is defined as **active** if $c_i(x) = 0$; while is defined **inactive** if the strict inequality $c_i(x) > 0$ is satisfied.*

Smoothness of objective functions and constraints is an important issue in characterizing solutions. In fact, it ensures that the objective function and the constraints all behave in a reasonably predictable way and therefore allows algorithms to make good choices for search directions.

Typically, a First Order Taylor Series expansion of the objective function is used in order to form an approximate problem in which both objective and constraints are linear. It is necessary to notice that this approach makes sense only when the linearized approximation captures the essential geometric features of the feasible set near the point x considered. If the linearization is different from the feasible set near x the linear approximation cannot be used because it could lose information about the original problem. Hence, some assumptions about the nature of the active constraints c_i are required in order to ensure that the linearized approximation is similar to the feasible set. So, the *constrained qualifications* are assumptions that ensure similarity of the constraint set Ω and its linearized approximation in a neighbourhood of x^*. Given a feasible point x, define $\{z_k\}$ as a **feasible sequence approaching** x if $z_k \in \Omega \ \forall k$ that are sufficiently large and $z_k \to x$. Moreover, a local solution of the optimization problem defined in 2.8 is characterized as a point x at which all feasible sequences approaching x have the property that $f(z_k) \geq f(x) \forall k$ that are sufficiently large.

Definition 8 *A vector v is said to be a **tangent vector** to Ω (a tangent is a limiting direction of a feasible sequence) at a point x if there are a feasible sequence $\{z_k\}$ approaching x and a sequence of positive scalars $\{t_k\}$ with $t_k \to 0$ such that the following equations is verified*

$$\lim_{k \to \infty} \frac{z_k - x}{t_k} = v \tag{2.10}$$

The set of all tangents to Ω at x^* is called the tangent cone and is denoted by $T_\Omega(x^*)$.

Definition 9 *Given a feasible point x and the active constraint st $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(x)$ is defined as*

$$\mathcal{F}(x) = v \mid \begin{cases} v^T \nabla c_i(x) = 0, \ \ \forall i \in \mathcal{E} \\ v^T \nabla d_i(x) \geq 0, \ \ \forall i \in \mathcal{A}(x) \cap \mathcal{I} \end{cases} \tag{2.11}$$

Constraint qualifications are conditions under which the linearized feasible set $\mathcal{F}(x)$ is similar to the tangent cone $T_\Omega(x)$. In fact, the two sets are identical thanks to the constraint qualifications. Therefore, set these conditions allow $\mathcal{F}(x)$, that is defined as a linearization of the algebraic description of the set Ω at x, to take into account the geometric features of the set Ω.

Definition 10 *Given point x and the active set $\mathcal{A}(x)$, the linear independence constraints qualification (LICQ) holds if the set of active constraint gradients $\{\nabla c_i(x), i \in \mathcal{A}(x)\}$ is linearly independent.*

Let the Lagrange function of the problem be defined as 2.8

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{2.12}$$

where λ_i is called *Lagrange Multiplier* and c_i considered both equality and inequality constraints.
Suppose that x^* is a local solution of the optimization problem 2.8 and that the objective function f_i and constraints c_i are continuously differentiable and that the LICQ holds at x^*. Then there is a Lagrange multiplier vector λ^*, with components λ_i^* with $i \in \mathcal{E} \cup \mathcal{I}$ such that the following conditions are satisfied at (x^*, λ^*)

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \\ c_i(x^*) = 0 \quad \forall i \in \mathcal{E}, \\ c_i(x^*) \geq 0 \quad \forall i \in \mathcal{I}, \\ \lambda_i^* \geq 0 \quad \forall i \in \mathcal{I}, \\ \lambda_i^* c_i(x^*) = 0 \quad \forall i \in \mathcal{E} \cup \mathcal{I}. \end{cases}$$

These conditions are known as the First order necessary conditions or Karush-Kuhn-Tucker and are complementarity condition.
Given a local solution x^* of optimization problem 2.8 and a vector λ^* satisfying the previous mentioned conditions, it is possible to say that the strict complementarity conditions hold if exactly one of λ_i and $c_i(x^*)$ is zero for each index $i \in \mathcal{I}$.
If this strict complementarity property usually makes it easier for algorithms to determine the active set $\mathcal{A}(x^*)$ and converge rapidly to the solution x^*. When an optimization problem as 2.8 is posted and a solution x^* is determined, it is possible to find many vectors λ^* that satisfied the aforementioned conditions, that when the LICQ holds the optimal solution that can be found is unique.

2.3 Solving Technique

Typically, an optimization problem has the following standard form defined in Equation 2.8. In the case of an LP problem, where the cost function is linear the problem assumes the following standard form

$$\begin{aligned} &\text{minimize } c^T x \\ &\text{subject to } \begin{cases} Ax = b; \\ x \geq 0. \end{cases} \end{aligned} \tag{2.13}$$

where c and x are vector in $\mathbb{R}^n$, b is a vector in $\mathbb{R}^m$ and A is a $m \times n$ matrix and usually is assumed that it has full row rank. In this section different techniques to solve optimization problem are summarized; these algorithms can differs depending on the class of optimization problem (LP, NLP, ...)

2.3.1 Linear Programming: The simplex method

The **Simplex method** is linear programming method for solving optimization problem. This method solves linear programs by moving along the boundaries from one vertex (extreme point) to the next [1]. It tests adjacent vertices of the feasible set that is a polytope in sequence so that at each new vertex the objective function improves (in minimization problem a lower value) or is unchanged. Simplex Method is still one of the most used method for solving optimization problem due to its efficient in practise (in fact it is usually taking 2m or 3m iteration, where m is the number of equality constraints) and converging in a polynomial time for certain distributions of random inputs.
In order to visualize how the *Simplex Algorithms* solves an optimization problem in standard form, as expressed in 2.8 consider the Figure 2.3

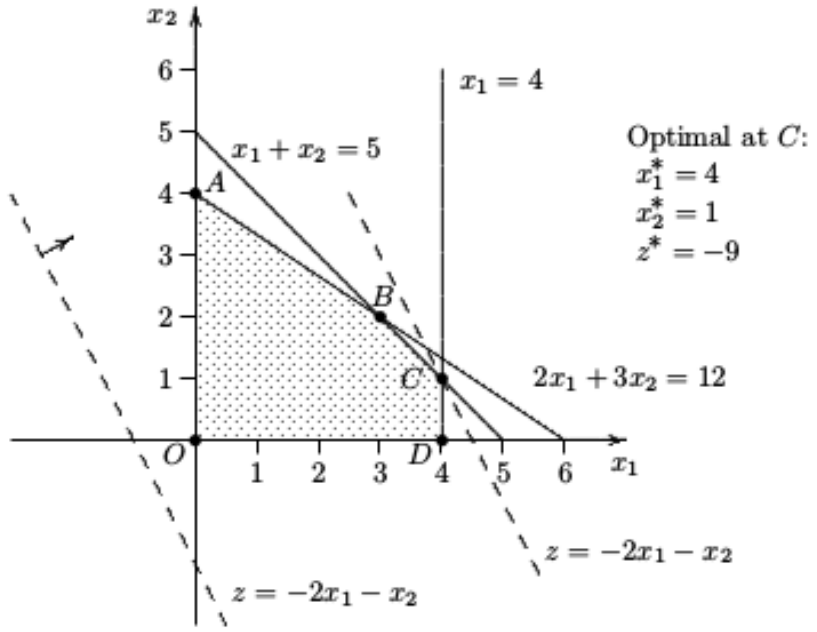


Figure 2.3: Graphical Solution of a Two-Variable LP [[1]]

The points labelled O, A, B, C and D are the vertices or extreme points, C is the optimal vertex. Segments OA, AB, BC, CD and DO are the edges or the boundaries of the feasible region. The simplex algorithm moves from O to A, then from A to B and last from B to C, or it can move from O to D to C, depending on the criteria adopted. The Simplex Algorithm described in this section assumes that an initial feasible solution (an external point) is provided. If no extreme point is given, a variant of the simplex method, called Phase I, is used to find one or to determine that there are no feasible solutions.

Each iteration of the simplex method is called a *basic feasible point* of 2.13. A vector x can be defined as a basic feasible point if x is feasible and if there exists a subset $\mathcal{B}$ of the index set $1, \ldots, n$ such that the following conditions are satisfied:

- $\mathcal{B}$ contains exactly m indices;
- $i \notin \mathcal{B} \implies x_i = 0$;
- Matrix B that has $m \times m$ dimension is defined as $B = [A_i]_{i \in \mathcal{B}}$ (is a partition of A) and it is a non-singular matrix, where A_i is the i-th column of matrix A defined in 2.13.

A set $\mathcal{B}$ that satisfies the aforementioned conditions is called a **basis** of the optimization problem 2.13 while the matrix B is called the **basis matrix**. In the theory of linear programming, a basic feasible solution (BFS) is, intuitively, a solution with a minimal number of non-zero variables. Geometrically, each BFS corresponds to a corner of the polyhedron of feasible solutions. If there exists an optimal solution, then there exists an optimal BFS.
Now the Simplex Algorithm consists in the following steps:

Algorithm 1: Simplex Algorithm

1 Put the linear program into standard form;
2 Find a first basic feasible solution;
3 Calculate the Reduced costs;
4 Test for optimality;
5 Choose the entering variable;
6 Calculate the search Direction;
7 Test for unboundedness;
8 Choose the leaving variable by the Min Ratio Test;
9 Update the solution;
10 Change the basis;
11 Go to step 3.

2.3.2 Active set Methods

Given a constraint $g_i(x) \geq 0$, this can be defined **active** at point x if $g_i(x) = 0$ meanwhile if $g_i(x) > 0$ at x the constraint is defined **inactive**. The active set at x is composed of constraints $g_i(x)$ that in point x are active. Typically, an active set algorithm has this structure: Some Active set methods are given below:

- Successive linear programming;

Algorithm 2: Standard structure of an active set algorithm

1 Find a feasible starting point;
2 Repeat until reach optimal enough;
3 Solve the equality problem defined by the active set ;
4 Compute the Lagrange multipliers of the active set;
5 Remove a subset of the constraints with negative Lagrange multipliers;
6 Search for infeasible constraints;
7 End Repeat.

- Sequential quadratic programming;
- Sequential linear-quadratic programming;
- Reduced gradient method;
- Generalized reduced gradient method;

2.3.3 Successive Linear Programming

Successive Linear Programming (SLP), which is also known as the Method of Approximation Programming, solves nonlinear optimization problems via a sequence of linear programs. This method starts from an estimate of the optimal solution and then solving a sequence of first-order approximations of the model (for example a linearization).

2.3.4 Newton's method

The Newton method is a root finding algorithm that at each iteration provides a better approximation to the roots of a real valued function. This method can be used to find a minimum or maximum of a function $f(x)$. Since the derivate is zero at a minimum or maximum, the local minima or maxima (due to optimization problem) can be found by applying this method to the derivate f' of the objective function f. The Newton method attempts to solve this problem by constructing a sequence $\{x_k\}$ from an initial guess (a starting point) x_0 that converges towards a minimizer x_* of f by using a sequence of second order Taylor approximation of the objective function f around the iterates. The second order Taylor expansion of f around x_k is defined as follows

$$f(x_k + t) \simeq f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2 \tag{2.14}$$

The successive iteration (step x_{k+1}) is defined to minimize this quadratic approximation in t and setting $x_{k+1} = x_k + t$. If the second derivative is positive, the quadratic approximation is a convex function of t, and its minimum can be found by setting the derivative to zero.

2.3.5 Sequential quadratic programming

The **Sequential quadratic programming** (SQP) is an iterative method for constrained nonlinear optimization. This method is used in optimization problem for which bot the objective function and the constraints are twice continuously differentiable. SQP methods solve a sequence of optimization subproblems, for each subproblem the goal is optimizes a quadratic model of the objective function, subject to a linearization of the constraints. If the problem is unconstrained, this method coincides with the Newton method, applies in a problem for finding a point where the gradient of the objective vanishes. While if the problem has only equality constraints, then the method is equivalent to applying Newton's method to the first-order optimality conditions.

2.4 Relation Techniques

Under opportune hypotheses, an optimization problem can be relaxed. This allows to solve an approximation of a difficult problem by nearby problem that are easier to be solved. In mathematical optimization this modelling strategy is called *relaxation*. For example, a linear programming relaxation of integer programming problem removes the integrality constraint and so allows non integer rational solutions. It is possible to use linear over and underestimations for each nonlinear term in a non-convex NLP in order to obtain a linear relaxation of the problem. Because a linear problem is always convex, the convexity properties guarantee the validity of a lower bound is still true. The advantage of a linear relaxation with respect to a convex (possibly nonlinear) relaxation is that linear optimization software can be employed to solve the relaxed underestimating problem. Linear optimization solvers are much more efficient than nonlinear optimization ones; hence the overall run of a Branch-and-Bound algorithm might be faster, and this can be seen in the following chapters. Two example of relation technique are the *Fractional Relaxation* and the *Lagrange Decomposition Method*

2.4.1 Fractional Relaxation

Usually solving integer linear programs is NP-hard, however writing a problem as an integer linear program is often very useful. In fact, it is possible to relax the integrality conditions to obtain a linear program, called *Fractional Relaxation* of the problem.
Since a feasible solution of an integer linear program is always a feasible solution of its fractional relaxation, the optimal solution of the linear program is at least as good as that of the integer linear program. In a minimization problem the relaxed program has a value smaller or equal to that the original program. Given a minimization problem as in Equation 3.1a and defined is fractional realization,

2.4.2 Lagrange Decomposition Method

The Lagrange relaxation (LR) decomposition is a technique used for solving non-linear constrained optimization problems. The purpose of this technique is to convert a difficult optimization problem into an easier problem and yields a solution that is an approximate solution of the difficult problem. With this method the problem can be relaxed removing the complicating constraints and transferring them to the objective function along with an assigned weight called *Lagrange multiplier*. The drawback of this method is that, if the overall problem is non-convex, the solution of the dual problem obtained in the LR decomposition method cannot be the same as for the initial problem. In this way a Lagrange multiplier penalizes the violation of constraints the objective function. Once the constraints are identified, the optimization problem is relaxed by moving them from the set of constraints to the objective function. In this way is possible to define the Lagrange function as:

$$\mathcal{L} = \mathcal{O} + \lambda\gamma \tag{2.15}$$

where $\mathcal{O}$ represents the objective function and λ and γ are the Lagrange multiplier and constraints, respectively.

2.5 Other Algorithms

Several algorithms for solving MRTA problems can be found in the literature. For example, in *Distributed Systems* the $\mathbf{A}^*$ **Algorithm** is often adopted.
This algorithm constructs the problem as a search tree and then it searches the node of the tree starting from the *starting node* (that represent a null solution) called *root*. *Intermediate nodes* represent the partial solutions while the *leaf* nodes represent complete solutions or goals. To compare solutions, a cost is associated to each node and it is computed by a cost function f as follows:

$$f(n) = g(n) + h(n) \tag{2.16}$$

where $g(n)$ represents the cost of the search from the starting node to the current node n; meanwhile $h(n)$ is a lower bound estimation of the cost from node n to a goal node.
These nodes are order for cost according to f, the algorithm first selects the node with the minimum expansion cost.
The algorithm maintains a sorted list of nodes, according to f, and always selects a node tithe the best cost of expansion. The expansion of a node is to generate all its successor or children. The algorithm guarantees to find an optimal solution as long as the heuristic $h(n)$ is admissible, that is, it is a lower-bound estimate of the cost from n to a goal state.
In the context of MRTA this technique is called **Optimal Assignment with Sequential Search (OASS)**; for this kind of problem some considerations are necessary;

- The search space is a tree;
- The initial node is a null assignment node;
- Intermediate nodes are partial assignment nodes;
- A goal node (solution) is a complete assignment node.

An example of application of this kind of algorithm in a task assignment problem is proposed in [14] and [15].

Also, **Genetic Algorithms** can be used to solve the MRTA. Genetic Algorithms work with a population of individuals. Each potential solution is represented by a set of parameters known as *genes*; these parameters are then combined to form a *chromosome*, which represents a possible solution. A genetic algorithm works as follows:

First a random population of candidate solutions is generated; then a *fitness function* is used in order to evaluate the fitness of each solution (the goodness of a solution); the next step is applied a selection criteria in order to choose some of the best individuals (those have a higher fitness value) that are used for generate new candidate solutions. Once some candidate are selected, various operations, such as *mutation* or *crossover*, are applied to them in order to produce new candidate solutions. Due to crossover operations, a new candidate solution inherits partial characteristics from its parent solution. The mutation operator instead prevents the loss of diversity and is helpful to traverse different regions of the search space and to escape from local minima or maxima. The step involved in Genetic Algorithm for task assignment are described in [16].

For binary integer programming problem, the **Branch and Bound (BB)** algorithm can be used to find an optimal solution. This algorithm is quite different respect to a *Brute Force* Algorithm, where all possible solutions are enumerated; in fact the *Branch and Bound* algorithm finds the bound of the objective function given certain subset of the feasible set. At each iteration the algorithm choose the branch with less cost and uses bounds to prune the others, more expensive, bounds. The algorithm starts from a root node and computes the cost associate to each node depending on the objective function. Selection of the node to be expanded is performed by alternating branching and bounding operation using *Jumptracking* or *Backtracking*. *Jumptracking* implements a frontier search where a node with a minimal lower bound is selected for examination, while *Backtracking* implements a depth first search where the descendant nodes of a parent node are examined either in an arbitrary order or in order of non decreasing lower bounds. Thus, in *Jumptracking* strategy the branching process jumps from one branch of the tree to another one, while in *Backtracking* the process first proceeds directly to the bottom along some path to find a trial solution and then retraces that path upward up to the firs level with active nodes [17]. Each edge in the tree represent an assignment of a task to a robot, while each node corresponds to a partial or complete allocation.

The algorithm is dived into two parts: *Branching* and *Bounding*.

Branching is the procedure of partitioning a large problem into two or more sub-problem that are usually mutual exclusive and then the procedure continues dividing the subproblem similarly. *Bounding* calculates a lower bound on the optimal solution values for each subproblem generated in branching process. A node is *Unfeasible* if it does not satisfy constraints and in this case no subtrees are generated starting from it. Since this is a brute force algorithm, the computation time is impractical and the memory required to save all node is very large. Thus, this can be used, such as the exact algorithm, only for small value of task and robot; a modified BB algorithm is proposed in [18], where an improvement both in Computation time and number of generated nodes is obtained. Another algorithm used to solve MRTA problem is the **Particle Swarm Optimization (PSO)**.

The *Particle Swarm Optimization* is a population based optimization algorithm, where each particle is associated a position that represent the quality of the solution and a velocity that determines the next iteration direction of the particle. The updating formulae for velocities and positions are as follows:

$$v_{ik}^{t+1} = v_{ik}^{t} + c_1 r_1 (x_{pbest_{ik}}^{t} - x_{ik}^{t}) + c_2 r_2 (x_{gbest_{ik}}^{t} - x_{ik}^{t}) \tag{2.17}$$

$$x_{ik}^{t+1} = x_{ik}^{t} + v_{ik}^{t} \tag{2.18}$$

where x_{ik}^{t+1} and x_{ik}^{t} represent the current and previous position of particle i in dimension k, respectively; v_{ik}^{t+1} and v_{ik}^{t} represent the current and previous velocity of particle i in dimension k, respectively; the weights c_1 and c_2 are acceleration constants which are generally viewed as the cognitive acceleration coefficient and social acceleration coefficient, respectively; r_1 and r_2 are random number in [0,1]; $x_{pbest_{ik}}^{t}$ is the best position of particle i in dimension k up to iteration t; and $x_{gbest_{ik}}^{t}$ is the best position of the whole swarm in dimension k up to iteration t.

The Algorithm described in [19] works as follows:

The Algorithm starts by generating an initial solution; then an iteration counter is initialized and the setup it reruns until this counter reaches a given value. At every iteration, the algorithm selects the next neighbourhood operator from an operator sequence that is randomly generated. For each operator all possible moves are examined and if an improved solution is found, this solution become the new best solution; while if no improvement can be obtained by any of the operators, a locally optimal solution is found. In order to run away from a local optimum, the search process is typically repeated. Before skipping to next iteration, the best solution is perturbed in order to obtain a new solution that can be potentially improved using operators. In [19] also a new encoding and decoding method to improve solution quality is proposed.

2.6 Software

Much software for solving optimization problem are developed. They are solver or linear programming (LP), quadratic programming (QP), mixed integer linear programming (MILP) and few of them for nonlinear programming (NLP) . Some of them are reported in this subsection:

1. **Gurobi Optimization**;
2. **CPLEX**;
3. **SCIP**;
4. **GLPK**.
5. **MOSEK**
6. **OR-TOOLS**

In Table 2.1 a summary about these solvers is proposed, describing what kind of problem each solver can resolve, what language can be used, and if it is possible to call external functions to perform some operation (e.g. to evaluate costs of objective function)

Software	LP	LNP	Functions	Language	License
Mosek	✓	✓	✓	C/C++, Java, Python, MATLAB	Free trial License
Gurobi	✓	✗	✓	C/C++, Java, Python, MATLAB	Free trial License
Cplex	✓	✗	✓	Cplex	Free trial License
Glpk	✓	✗	✗	Glpk	Free
Or-Tools	✓	✗	✓	C/C++, Java, Python	Free
SCIP	✓	✓	✓	C/C++, Java, Python	Free trial License

Table 2.1: Optimization Software

Chapter 3

Problem Formulation

In this chapter the MRTA problem is formulated. Firstly, Sections 3.1 and 3.2 introduces the notation used and recalls preliminary concepts about optimization problems; then, in Section 3.3 the description of the mathematical formulation is reported, defining the objective function and the constraints of the problem. The objective function is divided into two other functions that are called $\mathcal{B}$ and $\mathcal{F}$ function respectively. The former considers all interference-free costs, meanwhile the latter accounts for cost related to the interference between robots.

3.1 Binary Programming Problem

The task assignment problem can be formulated as a binary integer programming problem (BIP), i.e., a problem where each decision variable can be either 0 or 1 and all the coefficient of the objective function are non-negative (in order to ensure a physical representation of the costs). Since BIP are a particular subset of integer programming problems, they are NP-Complete that is when the objective function and the constraints are all linear; under these conditions is possible to relax the constraints and the problem becomes a mixed-integer linear program (MILP), as described in Section 2.4. Once the problem is relaxed it is easier to solve, since it is possible to use an algorithm that can solve the problem in polynomial time. Several techniques can be used to resolve MILP such as Local Search Algorithm or other algorithm that use the sub-gradient of the objective function. Finding the optimal solution is equivalent to determine an optimal task assignment of AVGs fleet [20].

3.2 Notation and preliminaries

First, symbols and variables are defined as follows:

- Let $\{r_i, \ldots, r_n\}$, $n \in \mathbb{N}$, be the set of (possibly heterogeneous) robots composing the fleet. Hence-

forth, label i is used to referring to robot r_i, where each robot r_i is identified by a unique identifier (ID). Let also $\mathcal{R}$ be the set of robot indices.

- Let $\{g_1, \ldots, g_m\}$, $m \in \mathbb{N}$, be a set of m non-cooperative tasks; each g_j is characterized by a couple of pose in $SE(3)$, $g_j^{start} = (x_j^s, y_j^s, \theta_j^s)$ and $g_j^{final} = (x_j^f, y_j^f, \theta_j^f)$, where (x_j, y_j) is a Cartesian position expressed in an inertial reference frame and $\theta_j \in (-\pi, \pi]$ is the goal orientation. Each task involves moving from a starting configuration g^{start}, which may differ from the starting position of a robot, to a final configuration g^{final} in which some operations (e.g., picking or placing an object) may be eventually performed. Note, in fact, that this definition may easily extend to include other noncooperative operations. Specifically, it is assumed that this latter may be performed only in the starting or in the final configuration and each of them to be characterized by an operational time (i.e., the time to perform an operation). Another assumption regards an estimation of each operation time that is then included in the definition of a task. Let also $\mathcal{G}$ be the set of tasks' indices.

- Let $\mathcal{P}_{ij} = \bigcup_{s=1}^{p_{ij}} \boldsymbol{p}_{ijs}$ be the set of p_{ij} alternative paths to reach a task j for a robot i, with $\mathcal{P}_{ij} = \emptyset$ when robot i cannot execute task j. Notably, p_{ij} may differ for different pairs (i, j). Specifically, each path $\boldsymbol{p}_{ijs}$ is a continuous function $\boldsymbol{p}_{ijs} : [0, 1] \to \mathcal{Q}_i^{\text{free}}$, parameterized using arch length σ, where $\mathcal{Q}_i$ is robot i's configuration space. Given a starting configuration $q_i^0 \in \mathcal{Q}_i^{\text{free}}$ and a goal configuration $q_i^1 \in \mathcal{Q}_i^{\text{free}}$, the path planning problem consists in finding a kinematically feasible obstacle-free path $\boldsymbol{p}_{ijs}$ such that $\boldsymbol{p}_{ijs}(0) = q_i^0$ and $\boldsymbol{p}_{ijs}(1) = q_i^1$ and $\boldsymbol{p}_{ijs}(\sigma) \in \mathcal{Q}_i^{\text{free}}$ for all the $\sigma \in [0, 1]$. In particular, paths are assumed to be known at each instance of the task allocation algorithm.

- Let $\mathcal{P}^{\text{all}} = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m} \mathcal{P}_{ij}$ be the set of all paths.

- Let $\mathcal{P}$ be the current set of assigned paths (path of busy robots); these robots must be considered into the problem due to the presence of the interference.

- Let n^{idle} be the number of idle Robots at time t; tasks can be assigned only to idle robots; let R^{idle} be the set of idle robots' indices.

3.3 Mathematical Formulation

In this study, a possibly different number of robots and tasks are considered, and the robots should be allocated to tasks in order to optimize the overall performance of the fleet. Associate a task to a robot i's implies that a robot will reach the start position, perform the first operation (e.g. takes some instruments that must be used to perform the second operation), reaching the final position of the task and then performing the final operation. As common in literature [3], at each instance of the task

allocation problem, dummy robots or tasks will be included in the problem whenever n^{idle} differs from m, the number of tasks. In this manner the problem is square and it is simpler to solve. Moreover, each task is assigned only to one robot, so cooperative tasks are not considered during this work and robot can execute only one task each time (i.e. an ST-SR-IA). This implies that, if a robot is already executing a mission, it will be not available to be assigned to a new task. Therefore, the task allocation problem falls into the ST-SR-IA, and hence can be formulated as the following OAP: the aim is finding x_{ijs} such that

$$\underset{x=\{x_{ijs}\}\in\mathcal{X}}{\text{minimize}} \;\; \alpha\mathcal{B}(p,t,q,\dot{q},\ddot{q}) + (1-\alpha)\mathcal{F}(\mathcal{P}^{\text{all}},t,Q,\dot{Q},\ddot{Q}) \tag{3.1a}$$

subject to:

$$x_{ijs} \in \{0;1\}, \forall (i,j,s) \in \{1,\dots,n\}\times\{1,\dots,m\}\times\{1,\dots,p_{ij}\} \tag{3.1b}$$

$$\sum_{i=1}^{n}\sum_{s=1}^{p_{ij}} x_{ijs} = 1 \;\; \forall i \tag{3.1c}$$

$$\sum_{j=1}^{m}\sum_{s=1}^{p_{ij}} x_{ijs} = 1 \;\; \forall j \tag{3.1d}$$

$$x_{ijs} = 0 \;\; \forall s \text{ if robot } i \text{ cannot perform task } j \tag{3.1e}$$

where:

- Let α be a user-defined parameter which weights the two components of the objective function to be optimized;
- To account for different optimization criteria (e.g., efficiency, energy consumption, etc.), the interference-free cost function $\mathcal{B}$ is expressed as a linear combination of different cost functions $w_{ijs}^{k} \in [0,1]$ are combined (with weights $\beta_k \in [0,1]$ such that $\sum_{k=1}^{\eta}\beta_k = 1$), that is, $\mathcal{B} = \sum_{k=1}^{\eta}\beta_k \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{s=1}^{p_{ij}} w_{ijs}^{k}\, x_{ijs}$. Each $w_{ijs}^{k} \in [0,1]$ accounts for individual robot information and indicates the cost for robot i to perform the task j through the path s according to the k-th interference-free cost function.
- Each $x_{ijs} \in \{0,1\}$ is a binary variable such that $x_{ijs} = 1$ when the robot i is assigned to the path $\boldsymbol{p}_{ijs}$ to execute the task j; otherwise, $x_{ijs} = 0$.
- $\mathcal{S}$ is a set of pairs (i,j), $i \in \{1,\dots,n\}$, $j \in \{1,\dots,m\}$ used to model both capability and capacity constraints. Specifically, $(i,j) \in \mathcal{S}$ whenever the robot i is not capable of performing for the task j or when it is already performing a task allocated at a previous time (capacity constraint);
- The first part of Equation 3.1a ($\mathcal{B}$) takes into account the interference-free assignment cost through the linear sum of costs, while the second part ($\mathcal{F}$) considers the additional cost associated with the interference.

Constrain 3.1b states that x_{ijs} is a binary decision variable. Constrains 3.1c and 3.1d ensure that: 1. only one path to each goal is selected for each robot, 2. each task can be assigned only to a robot, and 3. each robot can execute one task at a time. Constraint 3.1e ensures that task can be assigned only to robot that can perform it.

Chapter 4

Interference-free Function

In order to determine the best matches between a robot and several task instances generated from the same task, in this chapter some way to calculate the costs associated to allocate a task to a robot are defined. Equation 3.1a quantifies the total cost associated with a given assignment. As said before, in this study the costs are divide into two categories: the interference-free assignment cost and the additional cost associated with the interference. The first is individual to each robot and may depend on its state and dynamics. The second considers the coordination between AVG fleet, which includes the additional cost of the effects of precedence on overlapping regions of the paths. The second accounts for how the other robots in the fleet affect each robot motion due to shared regions of the workspace to be traversed. It is possible to consider more costs in the optimization problem, then the simplest way to resolve OAP is a weighted linear combination of all costs. Through weights is possible to choose how much important give to each cost. This chapter focus on the first part of Equation 2.3, the one related to the interference-free costs. In the following sections, different possible interference-free costs are described. All these costs are considered during this work, and the importance to give to each cost can be weighed by a linear parameter β as described in Chapter 3, while the second part of Equation 2.3 is addressed in Chapter 5.

4.1 Total Path Length

In this case, assignment costs will depend on the robot path length to reach the task. As said before, a path is a continuous function $\boldsymbol{p}_i : [0,1] \rightarrow \mathcal{Q}_i^{\text{free}}$. Given a starting and a goal configuration, the path planning problem consists of finding a kinematically feasible obstacle-free path $\boldsymbol{p}_i$ from the starting position of the robot to the goal position of a task.
It might be thought, once a path for robot i is computed, to resolve the MRTA problem, assigning to a robot i a task in order to minimize the total distance traversed by the robots in the fleet. In this manner, each robot is associated with a task that minimizes robot path p_{ij} that can be computed as $\int_0^1 \boldsymbol{p}_{ijs} d\sigma$.
In order to evaluate the cost w_{ijs} in the Equation 3.1a, since $w_{ijs} \in [0,1]$, this length is normalized and

so w_{ijs} is defined as $w_{ijs} = \frac{\int_0^1 \boldsymbol{p}_{ijs} d\sigma}{p_{max}}$ where $p_{max} = \max_{i \in \mathcal{R}} \boldsymbol{p}_{ijs}$ is the length of the longest path in the given roadmap $\mathcal{P}^{all}$. In this work, the complete set $\mathcal{P}^{all}$ was assumed to be computed at each instance of the task allocation algorithm.

4.2 Arrival Time

In this case, assignment costs will depend on the time needed by the robots to physically reach the task. In literature [21] is possible to find several ways to generate a robot trajectory from an initial point to goal point. Typically, two different velocity profiles are considered in the trajectory generation. One of them is the trapezoidal profile, which is illustrated in Figure 4.1.

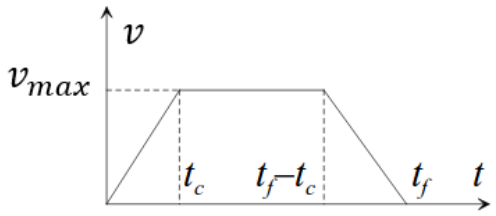


Figure 4.1: Trapezoidal velocity profile

The trajectory can be divided into three parts:

- Constant Acceleration, linear velocity and parabolic position;
- Null acceleration, constant velocity, linear position;
- Negative acceleration, linear velocity, parabolic position.

Let $v_i^{\max}$ and $a_i^{\max}$ be the max velocity and acceleration, respectively, for robot i. Also, let $t_c = \frac{v_i^{\max}}{a_i^{\max}}$ be the acceleration time and $\Delta_i = v_i^{\max}(t_f - t_c)$ the travelled distance. The time for each robot i to achieve the final configuration, namely, the **arrival time**, can be then computed as follows:

$$t_f = \frac{\Delta_i}{v_i^{\max}} + \frac{v_i^{\max}}{a_i^{\max}} \tag{4.1}$$

The other profile is the triangular profile, that is shown in Figure 4.2

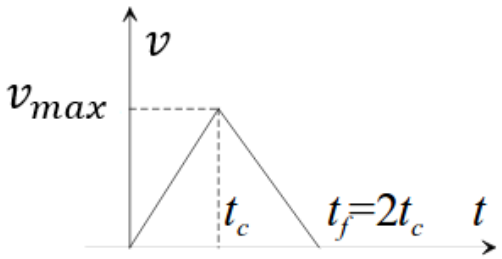


Figure 4.2: Triangular velocity profile

In this case once the system reach the max velocity, it starts decelerating in order to stop in the final point.

It's possible to evaluate the travelled distance as $\Delta = a_i^{\max} t_c^2$ and the **arrival time** as:

$$t_f = 2\sqrt{\frac{\Delta}{a_i^{\max}}} \tag{4.2}$$

where: $t_c = \sqrt{\frac{\Delta}{a_i^{\max}}}$ and $v_i^{\max} = a_i^{\max} t_c = \frac{\Delta}{t_c} = 2\frac{\Delta}{t_f}$

In this case, it might be thought to resolve the MRTA problem minimizing the total traveling time of the robot. In this way, each task is associated with the robot that will take less time to arrive at the goal point. The cost w_{ijs} in the Equation 3.1a is defined as $\frac{t_{f_i}}{t_{f_{max}}}$ where $t_{f_{max}}$ is the arrival time needed by the slowest robot of the fleet to reach the further task. It is important to notice that this time is evaluated considering the robot alone on the map.

4.3 Robot Competence

Let $\rho_{ij} \in [0, 1]$ be the capacity of robot i to perform task j, where $\rho_{ij} > \rho_{kj}$ implies that robot i is *more capable* than robot k to perform the task j. The dependency from the subscript s is omitted since ρ_{ij} is constant for all the paths $\boldsymbol{p}_{ijs} \in \mathcal{P}_{ij}$. The OAP now will associate each task to robot that are more capable on executing it. If $\rho_{ij} = 0$ a constraint $x_{ij} = 0$ is added (Equation (3.1e)); if $\rho_{ij} \in (0, 1]$, the competence is used to define the interference-free cost $w_{ijs} = 1 - \rho_{ij}$ in the $\mathcal{B}$ function in Equation 3.1a.

4.4 Tardiness

Another problem, especially on industrial application, is guaranteed that tasks are performed until a defined deadline [22] and [23]; these problems are called the problem of tardiness. Once a time window (in which a task must be completed) is provided, the tardiness is defined as the further time required to

complete the task after the end time of the time window. For example, if the task of the robot is to feed multiple machines, the late arrival of the mobile robot at a feeder causes a delay in production.
In this study, the problem of scheduling m delay constrained tasks on n mobile robot is considered as follows. Let T_j denote the operation time of task $\mathcal{J}_j$, this time can be computed as:

$$C_j = \mu_j + \omega_j \tag{4.3}$$

where μ_j is the arrival time to reach task final location from robot starting position and ω_j is the workload (both in seconds). Each task has a due time D_j. Although the task is expected to be accomplished by the given due time, any tardiness is subject to a cost penalty. Given a feasible schedule, the tardiness of task $\mathcal{J}_j$ can be computed as

$$max(0, (C_j - D_j)) \tag{4.4}$$

where C_j denotes the completion time of task $\mathcal{J}_j$.
The goal is to assign tasks to the robots such as the delay of a scheduled task does not exceed a given delay threshold. The tardiness for all tasks can be computed as

$$\Theta = \sum_{\{j=1....m\}} max(0, (C_j - D_j)) \tag{4.5}$$

In this case, the related cost w^k_{ijs} in the $\mathcal{B}$ function on Equation 3.1a can be computed as follows:

$$w_{ijs} = \Theta_{ijs} \tag{4.6}$$

An OAP minimizing the tardiness assigns the tasks to penalize the robot that will employ more time over the deadline to complete a task. Another idea can be performing a task assignment in which each task is performed by the robot that will employ less time to complete the task. In this case, it is defined as *earliness*. The earliness of task $\mathcal{J}_j$ can be computed as

$$min(0, (C_j - D_j)) \tag{4.7}$$

Chapter 5

Penalization Function

This chapter focus on the second part of Equation 2.3, the one related to the interference between robots. The penalization function maps a particular assignment to the additional cost associated with the interference.

In Section 5.1 a preview on Critical Section is provided, describing how to compute and manage a critical section; then in Section 5.2 method to evaluate the time delay due to precedence constraint is described.

5.1 Critical Section

Given a robot i, let $\mathcal{Q}_i$ be the robot's configuration space and let $R_i(q_i) \subset \mathbb{R}$ be the collision space when the robot is in a configuration $q_i \in \mathcal{Q}_i$. For each path $\boldsymbol{p}_i$, the *spatial envelope* $\mathcal{E}_i$ is defined as a set of constraints such that $\bigcup_{\sigma\in[0,1]} R_i(\boldsymbol{p}_i(\sigma)) \subseteq \mathcal{E}_i$. If the equality holds (this assumption is considered true from now on) a spatial envelope is the sweep of the robot's footprint along its path. Henceforth, define the spatial envelope as $\mathcal{E}_i^{\{\sigma',\sigma''\}} = \bigcup_{\sigma\in[\sigma',\sigma'']} R_i(\boldsymbol{p}_i(\sigma))$.
It is possible to note that the intersection between the spatial envelope and the set of obstacles is an empty set by construction, in fact, the collision between robot and obstacle are avoided via the motion planner.
Given a pair of paths $\boldsymbol{p}_{i_1}$ and $\boldsymbol{p}_{i_2}$, a collision between the robot i_1 and i_2 may happen only in the set $\{q_{i_1} \in \mathcal{Q}_{i_1}, q_{i_2} \in \mathcal{Q}_{i_2} \mid R_{i_1}(q_{i_1}) \cap \mathcal{E}_{i_2} \neq 0 \ \vee \ R_{i_2}(q_{i_2}) \cap \mathcal{E}_{i_1} \neq 0\}$. Let $\mathcal{C}_{i_1 i_2}$ be the decomposition of this set into its largest contiguous subsets, each of which is called a **critical section** [24]. An example of an envelope and critical section is shown in Figure 5.1.

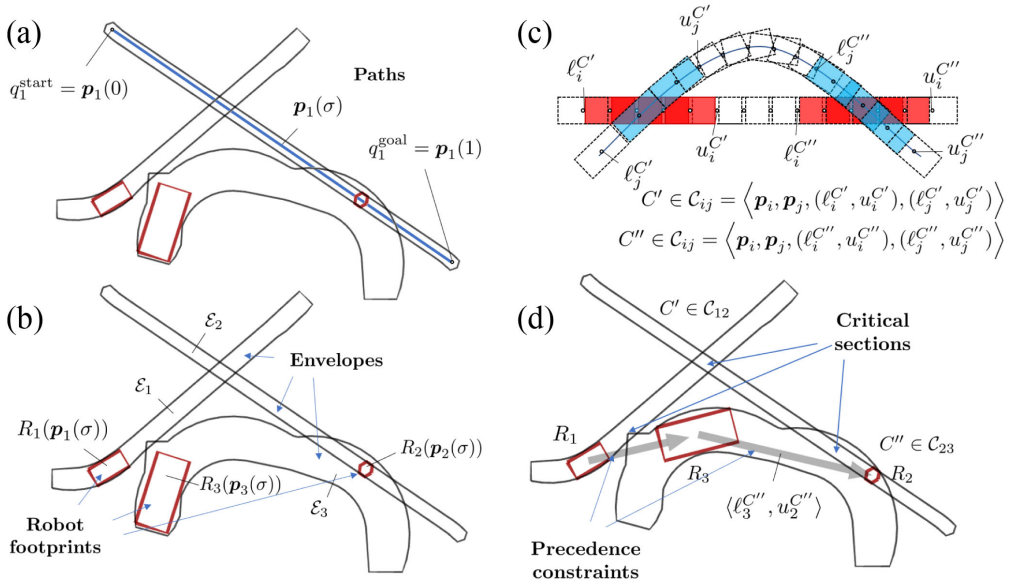


Figure 5.1: Envelope and Critical Section

For each critical section $C \in \mathcal{C}_{i_1 i_2}$, let $l_{i_1}^C \in [0,1]$ be the highest value of σ_{i_1} before robot i_1 enters the critical section; similarly, let $u_{i_1}^C \in [\ell_i^C, 1]$ be the lowest value of σ_{i_1} after robot i_1 exits the critical section. Considering the two robots temporal profile $\sigma_{i_1}(t)$ and $\sigma_{i_2}(t)$, if there exists a time t' such that $R_{i_1}(\boldsymbol{p}_{i_1}(\sigma_{i_1}(t'))) \cap R_{i_2}(\boldsymbol{p}_{i_2}(\sigma_{i_2}(t'))) \neq 0$ (this means that the robot collide while laying in their envelopes), then $l_{i_1}^C < \sigma_{i_1} < u_{i_1}^C$ and $l_{i_2}^C < \sigma_{i_2} < u_{i_2}^C$. Therefore, given a set of paths $\mathcal{P}^{\text{all}}$, the *coordination problem* is the problem of synthesizing, for each couple of robot $(i_1, i_2 \neq i_1)$ such that $\mathcal{E}_{i_1} \cap \mathcal{E}_{i_2} \neq 0$, a constraint on temporal profiles σ_{i_1} and σ_{i_2} such that $R_{i_1}(\boldsymbol{p}_{i_1}(\sigma_{i_1}(t'))) \cap R_{i_2}(\boldsymbol{p}_{i_2}(\sigma_{i_2}(t'))) = 0 \ \forall t'$.
The coordination problem, described in [25], provides some *precedence constraints* in order to choose for each critical section which robot has the priority to cross first it, meanwhile the second robot waits for that the first one exits from the critical section.

5.2 Evaluation of Time Delay

Given a set of tasks $\mathcal{G}$, let $\mathrm{T}_{arr_{ij}}$ be the arrival time of i-th robot to j-th task in nominal condition (i.e. following perfectly the velocity profile and no stop to avoid collision with other robots). The real-time to reach a goal may be higher than $\mathrm{T}_{arr_{ij}}$ due to the presence of precedence constraint; in fact, the robot that not have the precedence at a critical section, must stop in the critical point before entering the critical section and wait that the other robot exits from the critical section. Define a *delay* δ as the

time require to solve a critical section (the time that a robot waits before entering the critical section). Let the Time To Complete(TTC) be the total time required by the robot to complete a task; it can be computed as follows

$$\text{TTC} \;=\; \text{T}_{arr} + \delta \tag{5.1}$$

where δ is the total delay due to the interference between two paths of two robots.
The delay can be computed with the *EstimateTimeDelay* function described in Algorithm 3

Algorithm 3: The estimateTimeDelay Function

Input : $\mathcal{R} = \{1, \ldots, n\}$ set of robots, $\mathcal{G} = \{1, \ldots, m\}$ set of goals, the ID of a robot it

Output: The total time delay δ

```
1 δ ← 0;
2 forall i ∈ R do
3     if i ≠ it then
4         δ= δ + Compute the Time Delay
5 return δ;
```

Algorithm 3 is based on these principles : if exists an intersection between the current robot path (the one with it as ID) and the path of another robot in the fleet, the *ComputeTimeDelay* function evaluates the time delay of both paths due to interference (lines 2-4); a variable is introduced to evaluate the time difference between the start of the two robots (necessary to correctly evaluate the delay); furthermore a boolean variable is used to impose safety or priority constraints; if the overlap between two paths not exist, this function return a negative time delay, that indicates the safety margin between the two robots. Note that a target has an initial configuration and a final configuration. The initial position of the robot may not coincide with the target starts and in this case, the time to complete a task must be considered in addition also the further time to reach the target start from current position. Furthermore, if a robot is already executing a task, this time is computed as the sum of the time to reaches the current target final configuration from current position and the time required to go from current target final position to start position of next target.
The TTC is computed with the function *estimateTimeDelayToExecuteMission*, described in Algorithm 4:

Algorithm 4: The estimateTimeDelayToExecuteMission Function

Input : $\mathcal{G} = \{1, \ldots, m\}$ set of goals, a robotID it

Output: The time to complete a Mission TTC (including waiting time and time to reach the target start) and the set of pathIds which build up the mission path_vec

1 . Initialize an offset variable with the time to reach the target start position from the current position;
2 Evaluate time to reach the next target start and the path;
3 **if** *path to reach the target not exist* **then**
4 return false;
5 Evaluate time to reach final positon of current task ;
6 Evaluate total time to reach next target start;
7 Estimate the path id that gives the smallest time to completion;
8 Increase this time with offset and compute path IDs;
9 **return** *TTC,path IDs*;

Algorithm 4 works as follows: given a robot ID and a set of targets, the time to reach the next target start from the final position of the current task for the current robot and the set of pathIds, that will bring the robot to the start, are initially computed (line 2) ; if path to reach the target start exists, the time for the current robot to reach the final position of the current target from the current position is evaluated, and increased with the previous one; the result of this sum is the total time to reach the target start (lines 5-6); then a check on if a path for the target (that connect initial position and final position) exists is performed , if the path does not exits the value ∞ is assigned to the time to complete variable *ttc*; if instead, the path exists, the target that requires the smallest time to completion and the set of paths Ids that builds up the mission are computed and finally, this time is increased with the time offset(lines 7-8);

For each possible assignment the sum of all delay due to intersection between a couple of paths can be computed and it can be used as a cost for $\mathcal{F}$ function in Equation 3.1a.

The penalization cost is considered as the additional delay time on arrival time due to the precedence constrains for every couple of the fleet. A discrete spatio-temporal representation of each robot trajectory is stored. This representation is used to evaluate spatial and temporal overlap. For each pixel of the critical section, the time delay that should be introduced to avoid a collision is evaluated (delay returned is the max).

Through this function is possible to evaluate the delay time on the arrival time for each robot, caused by the presence of both spatial and temporal overlap with other robots and so the necessity to introduce a precedence constraint. It is important to specify that this function provided a ∞ delay if the goal of a robot is on a critical section of another robot, and so considering the $\mathcal{F}$ Function is possible to avoid all assignment that gives rise to pathological situations, such as deadlocks.

Chapter 6

Algorithms for Task Assignment

In this chapter, some different algorithms to solve the optimal assignment problem are provided. The aim of the algorithm is to find an optimal assignment in order to optimize the overall performance of the fleet according to a certain criterion. The formulation of this problem is presented in Chapter 3. During this work the algorithms that are developed and analyzed are:

- Exact Algorithm;
- Modified Systematic Algorithm;
- Greedy Algorithm;
- Gradient Descent Algorithm;
- Simulated Annealing Algorithm.

All these algorithm are based on a structure described in Algorithm 5

Algorithm 5: The general structure of a SR-ST-IA Task Assignment Algorithm

Input: $\{s_i\}_{i=1}^n$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks.

Output: An optimal assignment of paths to robots and its cost (x^*, c^*).

```
Compute dummy robots and tasks;
Initialize the optimal solution and its cost x* ← ∅, c* ← ∞;
Initialize the set of visited solutions X^{f,v} ← ∅;
while (X^f \ X^{f,v}) ≠ ∅ do
    Compute a feasible solution x ∈ (X^f \ X^{f,v});
    X^{f,v} ← X^{f,v} ∪ {x};
    forall idle robots and tasks do
        if robot i is assigned to a task j then
            Compute the interference-free cost B(x);
            Compute the penalization cost F(x);
            Compute the total cost c(x) = αB(x) + (1 − α)F(x);
            if c(x) < c* then
                Update the optimal solution with (x, c(x));
return (x*, c*);
```

Each of the aforementioned algorithms is different from the evaluation of the next solution. To evaluate their performance, these algorithms have been developed in *Java* and are available at [26].

The software used to build the optimization problem is **OR-TOOLS** [27], an open-source software suite for optimization, tuned for tackling the world's toughest problems in vehicle routing, flows, integer and linear programming, and constraint programming. OR-TOOLS supports different programming languages to build the optimization problem and then, once the problem is modelled, is possible to use any of a half dozen commercial solvers to solve it: commercial solvers such as Gurobi or CPLEX, or open-source solvers such as SCIP, GLPK, or Google's GLOP and award-winning CP-SAT [1].

This software was chosen due to his simplicity and does not require an external application to write the optimization problem, but it is possible to write directly the problem in one of the main programming languages.

Now some useful variables are introduced.

Let w_B be the cost of the current assignment considering the interference-free function $\mathcal{B}$. Similarly, let w_F be the one related to the penalization function $\mathcal{F}$. The penalization function maps a particular assignment to the additional cost associated with the interference. Moreover, let $\mathcal{X}^{\mathrm{f}} \subseteq \mathcal{X}$ be the set of all feasible assignment according to the constraints (3.1b), (3.1c), and (3.1d). Also, let $\mathcal{P}^f \subseteq \mathcal{P}^{\mathrm{all}}$ be the

[1] To install *OR-TOOLS* follow the guidelines described in [28].

set of selected paths considering the current assignment, where each element p_{ij}, of this set, is the vector that contains all possible paths for the robot i-th to reach task j-th.
Several methods can be used to compute $\mathcal{X}^{\mathrm{f}}$: one of them is considering every possible combination of the vector p_{ij}; Specifically the assignment (i, j) is *feasible* if and only if $p_{ij} \neq \emptyset$
The classical method to compute $\mathcal{X}^{\mathrm{f}}$ is computing the set of all possible sorting of assignment vector (those one verify the constraints 3.1c and 3.1d) and then consider only those where the path to reach the target exists and that are kinematically permissible.
Optimization Problem Formulation is build following several steps:

1. Search for Idle Robots;
2. Evaluation of Dummy Robot and Tasks; these are added in order to have the square case $n^{idle} = m$, where a dummy task (if $n^{idle} > m$) is considered as a virtual task in which the robot remains in his starting position, meanwhile a dummy robot (if $m > n^{idle}$) is a virtual robot only; in this case, the worst targets are assigned to virtual robots. add to a queue and are considered as priority tasks when the Task Assignment function is recalled.
3. Build the optimization problem with the augmented number of robot and tasks (starting values of n^{idle} and m + presence of dummy robot or tasks);
4. Evaluate costs associated with the $\mathcal{B}$ and $\mathcal{F}$ function respectively;
5. Search for optimal solution (this can be a global or a local optimum depending on the algorithm)

Regarding $\mathcal{B}$, the interference-free function described in Chapter 4, costs considered were :

- Path Length;
- Arrival Time;
- Tardiness.

Meanwhile, the $\mathcal{F}$ function, described in Chapter 5, consider only the delay time to complete a task j-th for the robot i-th due to interference with other robots is considered; using the current assignment the time delay is computed by considering the intersection between the path s of robot i with the path of other robots. This delay is evaluated with a heuristic estimator as described in Chapter 5.

6.1 Exact Algorithm

Algorithm 6 builds upon Algorithm 5: once the number of dummy robot and tasks is computed and the optimization problem is built, for each feasible assignment, this algorithm computes the costs w_B and w_F associated with function $\mathcal{B}$ and $\mathcal{F}$ respectively and then the total cost associated to current assignment x (lines 8-15). Once the total cost of the current assignment is computed, it is compared with the optimal cost found so far and if the current cost is lower than the optimal cost, the current assignment is considered as new optimal assignment (lines 16-17); finally a constraint on the current solution is imposed in order to exclude this assignment and a new feasible assignment is computed (lines 18-19). The algorithm stops when no more feasible assignments can be computed.

Algorithm 6: Task Assignment Exact Algorithm

Input : $\{s_i\}_{i=1}^{n}$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks.

Output: An optimal assignment x^* and its cost c^*.

1 Initialize the optimal solution and its cost $x^* \leftarrow \emptyset$, $c^* \leftarrow \infty$;
2 Get the set of idle robots $\mathcal{R}^{idle}$ and number of idle robots n^{idle};
3 Get numbers of dummy robot n^d and dummy task m^d;
4 $n^a = n^{idle} + n^d$;
5 $m^a = m + m^d$;
6 Create the optimization problem in (2.3) considering n^a robots and m^a tasks;
7 Initialize the set of visited solutions $\mathcal{X}^{\mathrm{f,v}} \leftarrow \emptyset$;
8 **while** $(\mathcal{X}^{\mathrm{f}} \setminus \mathcal{X}^{\mathrm{f,v}}) \neq \emptyset$ $c \leftarrow 0$;
9 **forall** $i \in \mathcal{R}^{idle}$ **do**
10 **forall** $j \in \mathcal{G}$ **do**
11 **forall** $s \in \mathcal{P}_{ij}$ **do**
12 **if** $x_{ijs} = 1$ **then**
13 $w_B \leftarrow$ *evaluate*$\mathcal{B}(x)$;
14 $w_F \leftarrow$ *evaluate*$\mathcal{F}(x)$;
15 $c \leftarrow w_B + w_F + c$;
16 **if** $c \leq c^*$ **then**
17 *Update the optimal solution with* (x, c);
18 *Set a constraint on current solution;*
19 *Evaluate a new feasible solution;* **do**
20 **return**
21 (x^*, c^*)

The main drawback of this approach is that the evaluation of the lookup table for $\mathcal{X}^{all}$ may have potentially impractical running time in the worst case and may occupy a very large amount of memory. Assuming $n = m = M$ and $p_{ij} = p_{ij}^{max} = p \; \forall(i, j)$, so the dimension of x $\in \mathcal{X}^{\text{f}}$ is $M!p_{ij}^{M}$ and the lookup table has $M!$ possible combination, so the time complexity is exponential (i.e. in fact this problem is NP-hard) and the complexity is $O(M!)$. For example, in the case of $M = 50$, this means that possible combinations are 3.04×10^{64} and so 1.52×10^{44} MB of space, which is an impractical quantity of memory. Therefore, this algorithm finds an optimal assignment in a reasonable time on small instances, so the maximum acceptable value of M is 25. Other algorithms can be used to find a quick solution even if M is large. For example, a *Randomized Algorithm* can be used; in this kind of algorithm, a random permutation is generated by shuffling the array. Doing this operation for several iterations (a fixed number k that can be defined by an opportune choice), allows generating a sample of permutation that is a subset of the aforementioned lookup table. Notice that the same permutation can be created more than once, but this change decrease with increasing M. Therefore with this method only k possible random assignment are considered, this on one side allow to consider the case with a large value of M (that with the aforementioned algorithm is impractical) but on the other side the best solution found by the algorithm may not be the global optimum of the objective function but only a local one. The complexity now becomes O(k), with k that is the number of considered permutation; this value is chosen to consider the initial number of possible permutation (M!) and in order to have a polynomial complexity time.

6.2 Systematic Algorithm Modified

In this section, a modified version of the Systematic Algorithm to effectively prune the search space (reducing the average number of iterations required to find the optimal solution) while ensuring optimality. The approach is systematic (i.e., in the worst case, it requires the same number of iterations of the exact algorithm), however simulations demonstrate its practical effectiveness in most of the cases (see Chapter 7). The pseudo-code is listed in Algorithm 7. Let x^{-} be an optimal assignment considering only the cost of $\mathcal{B}$ function.

Algorithm 7 is based on the principle of Algorithm 5: firstly the number and the IDs of Idle Robots are computed and if the number of Idle Robots $n^{idle} \neq m$, the possibilities of adding dummy robot or dummy are considered in order to have the square case $n^{idle} =$ m (lines 1-2); the optimization problem is build considering also dummy robot and task (lines 3-5) through the *OR-TOOLS* software; the Optimal Assignment and the cost associated with it are initialized with 0 and ∞ (line 6); then an Optimal Assignment that minimizes the objective function considering only the $\mathcal{B}$ function is computed (line 7), wherein the B function only the interference-free cost are considered, so each cost w_{ijs}^{k} represent the k-th interference-free cost of assign robot i-th to task j-th through path s-th; the cost of current Assignment considering only $\mathcal{B}$ is evaluated and if it is lesser than the minimum of total cost from

previous Assignments, the cost of $\mathcal{F}$ Function for current Assignment is computed (lines 9-17); in this manner, some solutions are eliminated and this may reduce the computational cost of the algorithm (this is modelled as a constraint on cost); then the total cost of the current assignment is computed and compared with optimal cost find so far and if the current cost is lower than the optimal cost, the current assignment is considered as a new optimal assignment (lines 19-21); finally, a constraint on the current solution is imposed in order to not more consider current assignment as a value one and a new feasible assignment is computed (lines 21:23). The algorithm stops when no more feasible assignments can be computed or when the time is higher to a timeout than can be imposed to the problem in order to stop the search for the optimal solution after a certain amount of time.

Algorithm 7: Modified Task Assignment Algorithm

Input : $\{s_i\}_{i=1}^{n}$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks.

Output: An optimal assignment x^* and its cost c^*.

$R^{idle} \leftarrow$ get Idle Robots;
$[n^d, m^d] \leftarrow$ Get Numbers of Dummy Robots or Tasks;
$n^a = R^{idle} + n^d$
$m^a = m + m^d$
$oP \leftarrow$ build the Optimization Problem with n^a and m^a
$x^* \leftarrow \emptyset$, $c^* \leftarrow \infty$;
$x^- \leftarrow$ evaluate an Optimal Solution; /* Considering only the function B */
Initialize the set of visited solutions $\mathcal{X}^{\mathrm{f,v}} \leftarrow \emptyset$;
while $(\mathcal{X}^{\mathrm{f}} \setminus \mathcal{X}^{\mathrm{f,v}}) \neq \emptyset$ **do**
    $w_B \leftarrow evaluate\mathcal{B}(x^-)$;
    $c \leftarrow 0$
    $w_F \leftarrow 0$
    if $w_B < c^*$ **then**
        forall $i \in \mathcal{R}^{idle}$ **do**
            forall $j \in \mathcal{G}$ **do**
                forall $s \in \mathcal{P}_{ij}$ **do**
                    $w_F \leftarrow w_F + evaluate\mathcal{F}(i, j, s, x^-)$;
        $c \leftarrow w_B + w_F$
    if $c \leq c^*$ **then**
        $x^* \leftarrow x^-$;
        $c^* \leftarrow c$;
    Set constraint on current solution(x^-)
    $x^- \leftarrow$ evaluate a new Optimal Solution; /* Considering only the function B */
return (x^*, c^*)

Constraint on cost allows, in some cases, to eliminate some unfavorable solutions that have a cost considering only Function $\mathcal{B}$ that is higher than the minimum cost find so far. This may reduce the computation cost of the Algorithm. Figure 6.1 shows the number of solutions evaluated during the search for the optimal solution if the constraint is imposed and if not; it is important to remember that eliminate some solution is not guaranteed, in fact in the worst case all possible solutions are evaluated.

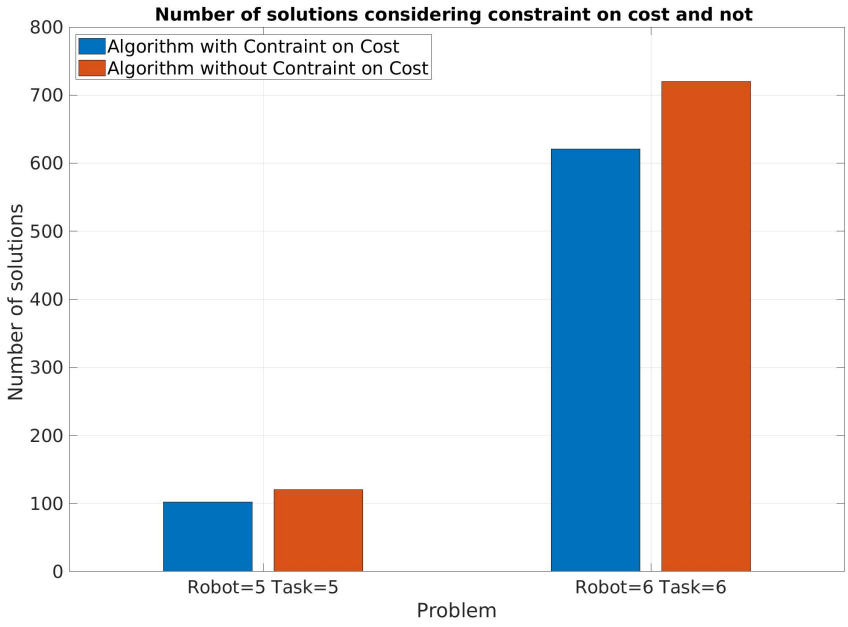


Figure 6.1: Number of Solution considering constraint on cost and not

6.3 Greedy Algorithm

Another Algorithm developed is the *Greedy Algorithm*. The greedy algorithm is a simple, intuitive algorithm usually used to solve the optimization problem. This algorithm makes the optimal choice at each step as it attempts to find the overall optimal solution. This kind of algorithms are quite successful in some problems as Dijkstra's algorithm, which is used to find the shortest path through a graph; but generally, a greedy strategy does not produce an optimal solution.

The idea of Algorithm 8 is to assign to each robot the best target in the list to drive, where the best target is considered the one that has minimized the $\mathcal{B}$ function. This Algorithm is obtained from 5: at each T_c a check if new targets to assign are available is performed (lines 1-2), if no new targets are available false value is returned; instead, if there are new target to assign, a check if there are some available robots is performed(3-7); for each robot, the cost associated with the $\mathcal{B}$ function is computed, and the task that minimizes this cost is selected (lines 8-15); once the best task is assigned to the current robot, the task is removed from the set $\mathcal{J}$ (line 18).

Algorithm 8: Greedy Algorithm for task assignment

Input : $\{s_i\}_{i=1}^{n}$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks.

Output: An optimal assignment x^* and its cost c^*.

```
1  if J = ∅ then
2      return false;
3  R^idle ← Get Idle Robots and number of Idle Robot n^idle;
4  [n^d, m^d] ← Get Numbers of Dummy Robots or Tasks;
5  R_a^idle ← Add dummy robot IDs to R^idle;
6  if R_a^idle is empty then
7      return false;
8  forall i ∈ R_a^idle do
9      w* = ∞;
10     j* = 0;
11     forall j ∈ J do
12         w_B ← evaluateB(i, j);
13         if w_B < w* then
14             w* = w_B;
15             j* = j;
16     x_ij ← Assign Robot i to task j;
17     x* ← Add(x_ij);
18     J.RemoveTarget(j);
19     if J = ∅ then
20         return x*;
21 return x*, c*;
```

This Algorithm obtains an optimal solution in a reasonable time even for large values of M and with less computation respect to Algorithm 7, but the optimal solution may not coincide with the optimal assignment. Algorithm 8 performs a local optimization for each robot, and the sum each local optimal

solution does not coincide with the globally optimal solution (i.e. the resultant assignment may be suboptimal). In the Greedy Algorithm each robot is assigned to the best task for him, so makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution. The main problem of the **Greedy Algorithm** is that it is a '*blind*' algorithm because it does not consider the interference between robots. In fact, the optimal solution is obtained considering the best allocation for each robot that considers only itself. The $\mathcal{F}$ function that can consider the interference between robots cannot be considered appropriately; this function can be considered in two ways:

- each robot consider only the interference with robots that are already allocated to a task;
- each robot considers all possible cases of all allocations for other robots of the fleet

In both cases Function $\mathcal{F}$ is evaluated differently compared to other algorithm developed during this work.

6.4 Local Search Algorithms

Finally, a couple of Local Search Algorithms are proposed, in order to have a better comparison with the Modified Systematic algorithm described in Section 6.2. Two different Local Search algorithms are developed during this work: A **Gradient Descent Algorithm** and a **Simulated Annealing Algorithm**.

6.4.1 Gradient Descent Algorithm

Gradient Descent is the most common optimization used algorithm due to its simplicity and computational cost. It is used a lot also in machine learning and deep learning. It is a first-order optimization algorithm; this means it only considers the first derivative when performing the updates on the parameters. On each iteration, the parameters are updated in the opposite direction of the gradient of the objective function. The size of the step on each iteration to reach the local minimum is determined by the learning rate α. Therefore, the direction of the slope downhill is followed until a local minimum is reached, this means that it is not guaranteed the global minimum (if exists) is found, but the optimal solution can be also a local minimum. There are three variants of gradient descent, which differ in how much data are used to compute the gradient of the objective function. Depending on the amount of data, it is possible to make a trade-off between the accuracy of the parameter update and the time it takes to perform an update. These variants are:

- Batch gradient descent;
- Stochastic gradient descent;
- Mini-batch gradient descent.

The pseudo-code of Gradient Descent Algorithm developed during this work is reported below.

Algorithm 9: Gradient Descent Task Assignment Algorithm

Input : $\{s_i\}_{i=1}^{n}$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks; K number of desired iterations

Output: An optimal assignment x^* and its cost c^*.

1 $R^{idle} \leftarrow$ get Idle Robots and number of Idle Robot n^{idle} ;
2 $[n^d, m^d] \leftarrow$ Get Numbers of Dummy Robots or Tasks;
3 $n^a = R^{idle} + n^d$;
4 $m^a = m + m^d$
5 $oP \leftarrow$ build the Optimization Problem with n^a and m^a
6 $x^* \leftarrow \emptyset$, $c^* \leftarrow \infty$
7 $k \leftarrow 1$,
8 $x^{start} \leftarrow$ evaluate a Feasible Solution;
9 $X^{prev} \leftarrow \emptyset$;
10 **while** $k < K$ **do**
11 $x \leftarrow$ Evaluate a Neighbour of x^{start};
12 **if** $x \notin X^{prev}$ **then**
13 $w_B \leftarrow evaluate\mathcal{B}(x)$;
14 $c \leftarrow 0$
15 $w_F \leftarrow 0$
16 **forall** $i \in \mathcal{R}^{idle}$ **do**
17 **forall** $j \in \mathcal{G}$ **do**
18 **forall** $s \in \mathcal{P}_{ij}$ **do**
19 $w_F \leftarrow w_F + evaluate\mathcal{F}(i, j, s, x)$;
20 $c \leftarrow w_B + w_f$
21 **if** $c \leq c^*$ **then**
22 $x^* \leftarrow x$;
23 $c^* \leftarrow c$;
24 $x \leftarrow x \cup \mathcal{X}^{prev}$
25 $k = k + 1$
26 **return** (x^*, c^*)

Once numbers of robot and task are defined, an evaluation on dummy robot and task is performed (lines 1-2) and some useful variables are initialized; A first feasible solution is computed and evaluated; each new solution is evaluated starting from the initial solution and considering its neighbours 11; the

neighbours are evaluated performing a permutation between a random couple of robots; Then the cost associated with this solution is evaluated and the current cost is compared to the optimal cost finds so far and in case the optimal solution is updated. Once all neighbours of a solution are considered, the algorithm re-start from the optimal solution and continue until the number of iterations indicated is reached (this number cannot be higher than the number of all feasible solutions).

6.4.2 Simulated Annealing Algorithm

The simulated annealing algorithm is an optimization method that mimics the slow cooling of metals, which is characterized by a progressive reduction in the atomic movements that reduce the density of lattice defects until a lowest-energy state is reached. Similarly, at each virtual annealing temperature, the simulated annealing algorithm generates a new potential solution (or neighbour of the current state) to the problem considered by altering the current state, according to a predefined criterion. The acceptance of the new state is then based on the satisfaction of the Metropolis criterion, and this procedure is iterated until convergence. During the annealing process, each a neighbour is accepted as a new solution with a time-dependent probability

$$P_T = \begin{cases} 1 & \text{if } f(x+1) < f(x) \\ e^{\frac{f(x)-f(x+1)}{T}} & \text{if } f(x+1) > f(x) \end{cases} \tag{6.1}$$

where f(x) is the current solution, f(x+1) is the next solution and T is the time. At the beginning of the process, which started from a randomly generated feasible solution, the higher probability of acceptance of new solutions allowed the algorithm to explore a wide region of the search space, thereby escaping from local minimum (i.e. when the algorithm starts the probability to accept a new solution is very high); however, as time was increased, the probability of acceptance of unfavorable solutions was reduced. The Pseudo-Code of the Simulated Annealing Algorithm developed during this work is reported in Algorithm 10.

Algorithm 10: Simulated Annealing Algorithm

Input : $\{s_i\}_{i=1}^{n}$ status of all the robots, with each s_i containing robot i's current position and mission; $\mathcal{P}$ current set of assigned paths; $\mathcal{G}$ (possibly empty) current set of tasks; K number of desired iterations

Output: An optimal assignment x^* and its cost c^*.

$R^{idle} \leftarrow$ get Idle Robot IDs and number of idle robots n^{idle};
$[n^d, m^d] \leftarrow$ get numbers of dummy robots or tasks;
$n^a = n^{idle} + n^d$;
$m^a = m + m^d$
$oP \leftarrow$ build the optimization oroblem with n^a and m^a
$x^* \leftarrow \emptyset$, $c^* \leftarrow \infty$
$k^* \leftarrow 1$,
$x^{start} \leftarrow$ evaluate a feasible solution;
$x^{sol} \leftarrow x^{start}$;
$X^{prev} \leftarrow \emptyset$;
while $k < K$ **do**
    $x \leftarrow$ evaluate a Neighbour of x^{sol};
    if $x \notin X^{prev}$ **then**
        $w_B \leftarrow evaluate\mathcal{B}(x)$;
        $c \leftarrow 0$
        $w_F \leftarrow 0$
        forall $i \in \mathcal{R}^{idle}$ **do**
            forall $j \in \mathcal{G}$ **do**
                forall $s \in \mathcal{P}_{ij}$ **do**
                    $w_F \leftarrow w_F + evaluate\mathcal{F}(i, j, s, x)$;
        $c \leftarrow w_B + w_F$
    $x^{sol} \leftarrow$ accept the neighbour as new solution with a certain probability
    if $c^k \leq c^*$ **then**
        $x^* \leftarrow x$;
        $c^* \leftarrow c$;
    $x \cup X^{prev}$
    $k = k + 1$
return (x^*, c^*)

This algorithm is similar to the Gradient Descent Algorithm described in Algorithm 9, but the evaluation of the next solution is different. In the Simulated Annealing Algorithm, a neighbour is accepted

as a new potential solution with a probability as shown in Equation 6.1.

6.5 NP Hardiness of MRTA problem with penalization

In general, there are several ways to estimate penalization costs in MRTA problems with penalization. When the evaluation of the interference is computable in polynomial-time the problem is called *Multiple-choice Assignment Problem with polynomial-time computable function (mApwPP)* [3]. Since the penalization function cannot be computed cheaply, the problem is still NP-hard. Another two cases can be investigated, and they differ for the form of penalization function: linear (mAPwLP) and general convex function (mAPwCP), which are in P and NP-hard respectively. Mathematically, the mAPwLP can be cast as an integer linear programming problem whose constraint matrix satisfies the property of totally unimodularity and thence the problem is in P. The mAp problem with a convex quadratic penalization function (mAPwCQP), is a subset of mAPwCP; this problem has the form

$$\min\{x^T Hx + cx\} \text{ subject to } Ax \leq b, x \in \{0,1\} \tag{6.2}$$

and is similar to the binary quadratic programming problem that is NP-hard. It is possible to prove that the mAPwCQP is NP-hard. Since mAPwCQP is NP-hard and mAPwCQP $\subseteq$ mAPwCP, also mAPwCP is NP-hard.

Chapter 7

Experimental Validation

In this chapter, the **Systematic Algorithm** developed is tested and compared with other classic algorithms proposed in Chapter 6. Some simulations were performed in order to make a comparison, in terms of the quality of the solution, between different algorithms. Properties that are analyzed for each algorithm are:

- Scalability;
- Time required for finding a solution;
- The goodness of the solution.

First, a scalability analysis is performed, in order to understand the computation cost and the potentiality of each algorithm when the numbers of robots and tasks increase. The second analysis regards the time required by the algorithm to find a solution: the systematic algorithm is able to find the optimal solution but it required more time to find it; meanwhile Local Search Algorithms are faster, but they do not guarantee to find the optimal solution. Another parameter analysed is the quality of the solution: the parameter used to evaluate the goodness of a solution is the sum of all difference between the nominal and the real arrival time. Since some algorithms find a solution that is a local minimum, this parameter allows understanding how much this solution is different respect to the best one. Clearly, there is not a best algorithm in all cases, but the best algorithm change depending on the application and desired performances.
Simulations are performed in a **Java** Environment, considering a different kind of Task Assignment Problem by changing scenario, the number of robots and tasks, maps, goals, number of paths for each task.
For each problem, different results are obtained changing the value of α in order to visualize how the solution differs , considering the same scenario, when important given to each part of Equation 3.1a change, and if exist a trend on the optimal value of α that gives the best solution. To compare different

solution, the criteria used is the arrival time, hence the sum of all differences between the real and the nominal arrival time for each robot in the fleet, where the real arrival time is the time required by the robot for going from its starting position to task final position allocate to it considering all AVG's robots, meanwhile the nominal arrival time is the arrival time if the robot was alone on the map. To build the optimization problem, OR-TOOLS software was used, while to solve it the aforementioned systematic algorithm described in Chapter 6, is used. All algorithms were written using **Java** as a programming language. In each test Tasks are shown with two hexagons, the blue one indicates the **Start Position** of the Task, meanwhile the red one indicates the **Goal Position**. Robots are represented through red rectangles, where the small arrow indicates the orientation of the robot, meanwhile, the big arrow between two robots is a precedence constraint managed by the coordination system. An example of how Task Allocation is different changing the value of α is reported in Figure 7.1 and Figure 7.2, where the number of Robots is equal to 5 meanwhile the number of tasks is 3.

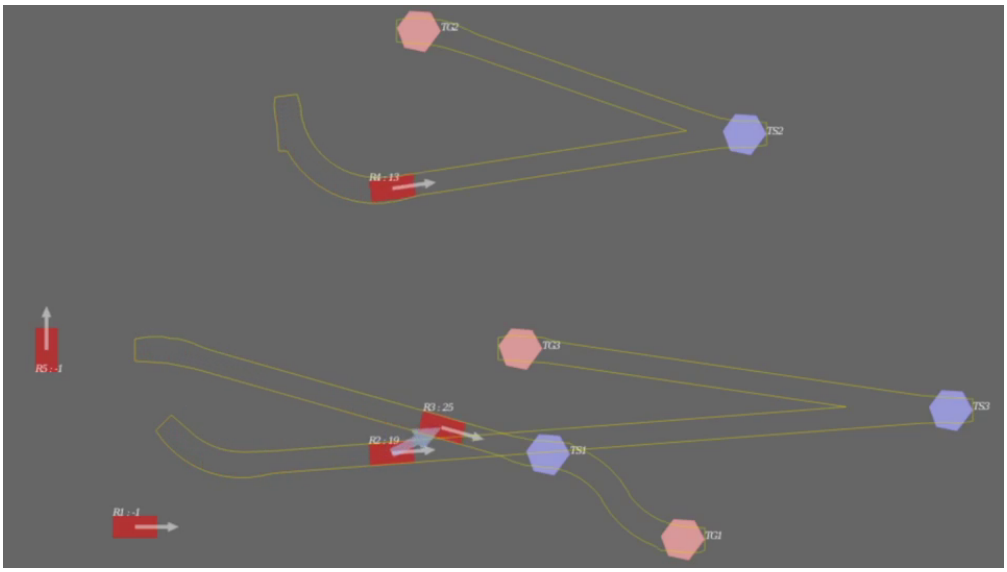


Figure 7.1: Task Allocation with $\alpha = 1$

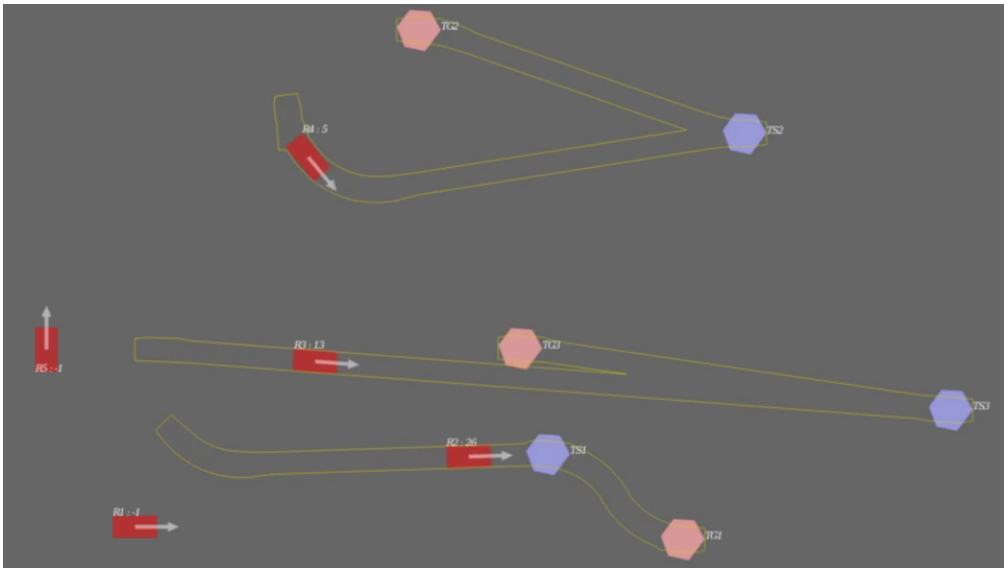


Figure 7.2: Task Allocation with $\alpha = 0.8$

The big arrows display the precedence constraints, where a critical section is found, where the robot with the arrow must wait for the robot that the arrow point to; the small ones instead display robot's heading.

7.1 Performance Indices

Before introducing results obtained through simulations, in this section the parameter used to evaluate the goodness of a solution is reported. Several metrics can be used to compare different solution, for this work, metric chosen is the arrival time; so for each allocation of the found solution, the different between the nominal arrival time (i.e. the time required by the robot to complete the task if it is alone on the map) and the real arrival time (i.e. the time required by the robot to complete the task considering other robots) is evaluated; all differences are added up and for each simulation, the best solution is the one with the minimum sum.

Another parameter that can be used to evaluate the goodness of a solution may be the delay time; so, in this case the best solution is the one that provided a minimum delay time, due to precedence constraint, for each robot. Clearly there is not the best parameter to use for evaluating the goodness of a solution for all scenario, but best criteria change due to application and desired performance of the fleet.

7.2 Scalability and Complexity

Scalability is the ability to handle increased workload without adding resources to a system. In other words, the scalability is how much a variable of the problem (e.g. number of robots) can be increased while the algorithm still can solve the problem in a reasonable time. In this section, the performance of the systematic algorithm was provided when numbers of robots and tasks are increased considering an empty scenario. In terms of complexity (i.e. the computational cost to find a solution) and scalability, the systematic algorithm developed during this work (Algorithm 7) is similar to an exact algorithm when $\alpha \neq 1$; the main difference is the additional constraint on cost imposed in the modified systematic algorithm that allows to eliminate some possible solutions of the optimization problem. However, this is not guaranteed, since, in the worst case, all the feasible solutions are evaluated. Since a systematic algorithm is NP-hard (as described in Section 6.5), complexity grows in an exponential way with the number of robot, task, and paths, so this algorithm takes a lot of time to find a solution when the number of robot and task is high (e.g. $n^{idle} = 25$). Differently, when $\alpha = 1$ the algorithm finds the optimal solution very fast and the scalability is very good (e.g. for $n^{idle} = 100$ the algorithm takes 1 second to find the optimal solution). This is possible because since the first solution that is found by the algorithm is the one that minimizes the $\mathcal{B}$ Function (i.e. the best solution), with the constraint on cost solution the next solution is surely associated with a cost that is higher than the first one, and so the algorithm is able to find the best solution at first iteration; Therefore, when the number of robot and task is very high and this algorithm is applied, consider to use a value of $\alpha = 1$ or use a timeout for the solution can be the best choice. To analyse the scalability property of the systematic algorithm, a simulation where the number of robots and tasks varied considering the same scenario is performed, as shown in Figure 7.3

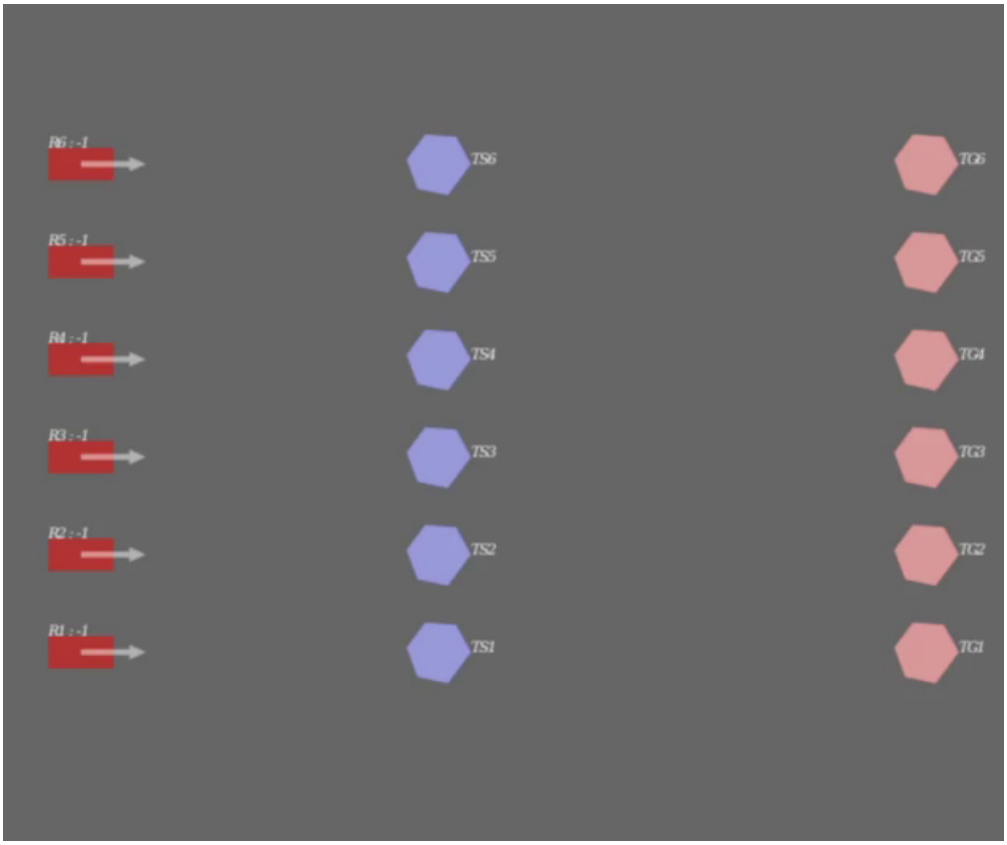


Figure 7.3: Scalability Analysis: Problem Configuration

In this case, tasks were simple a translation of robots' position on the x-axis. Figure 7.4 shows the total time required by the algorithm to solve the problem and to find the optimal solution in case of considering the presence of the $\mathcal{F}$ Function (i.e. $\alpha \neq 1$) and not considering the constraint on the cost of previous solutions.

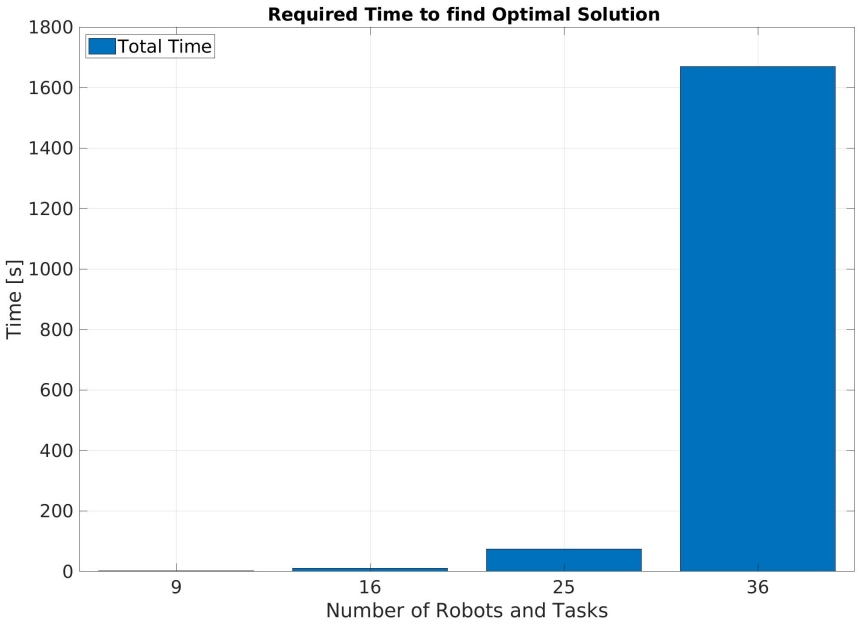


Figure 7.4: Time Required to find a solution considering the $\mathcal{F}$ Function

If instead the linear parameter $\alpha = 1$ or if the constraint on cost is considered, the algorithm can find a solution in a very small amount of time, as shown in Figure 7.5

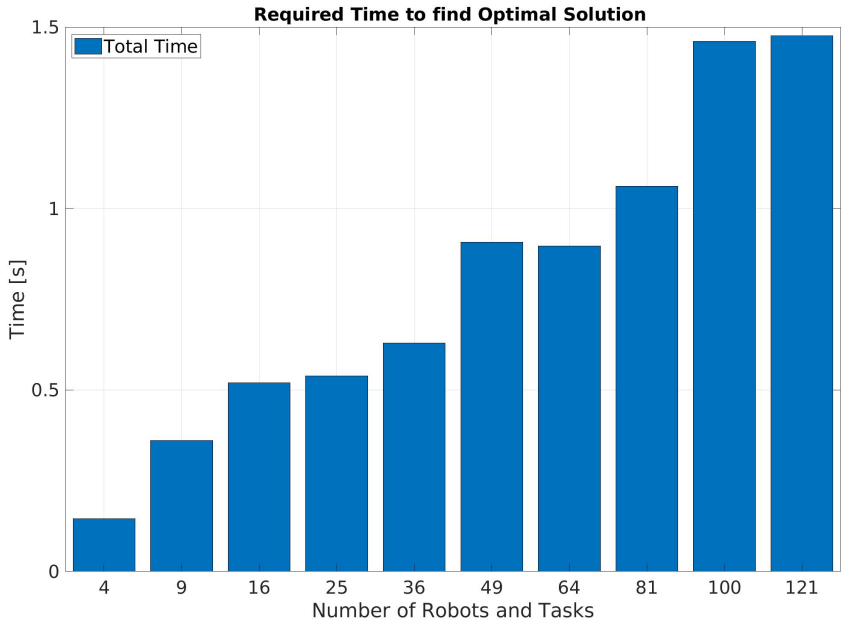


Figure 7.5: Time Required to find a solution without considering the $\mathcal{F}$ Function

In fact in this case, every new solution that is different from the one provided in Figure 7.6 , has a cost that is higher than this one, and so it is automatically eliminated by this constraint.

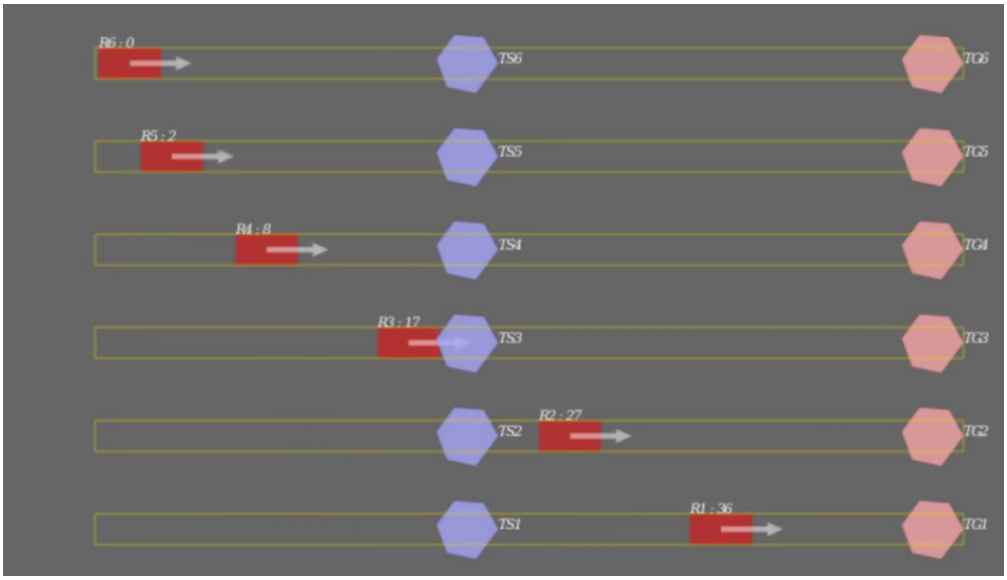


Figure 7.6: Scalability: Optimal Solution

7.3 Test 1: performance evaluation in empty space

The first test is performed in an empty environment where there are no obstacles except for the presence of another robot of the fleet.
Three different MRTA problems have been considered in this scenario, which differs for some aspects (i.e. numbers of robot e task, deadline, etc . . .).
Tasks and robots are of different types.

7.3.1 Varying Parameter α

Firstly results obtained varying the linear parameters α are provided in Figures 7.7 and 7.8.
The linear parameter α weights the important to give to $\mathcal{B}$ and $\mathcal{F}$ Function of Equation 3.1a.
Figure 7.7 shows the max arrival time for each simulation when the linear parameter changes.

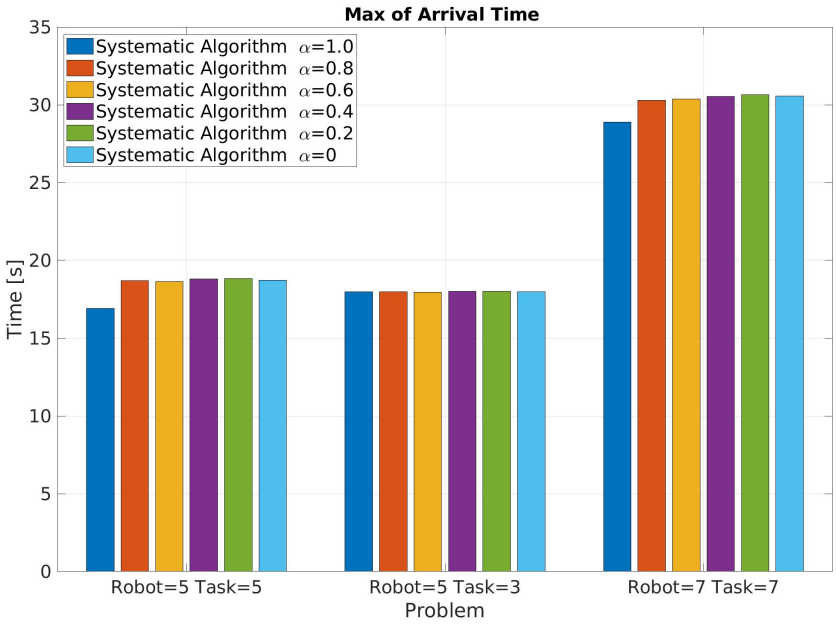


Figure 7.7: Test1: Max Arrival Time

Figure 7.8 shows the sum of all differences between the nominal and the real arrival time.

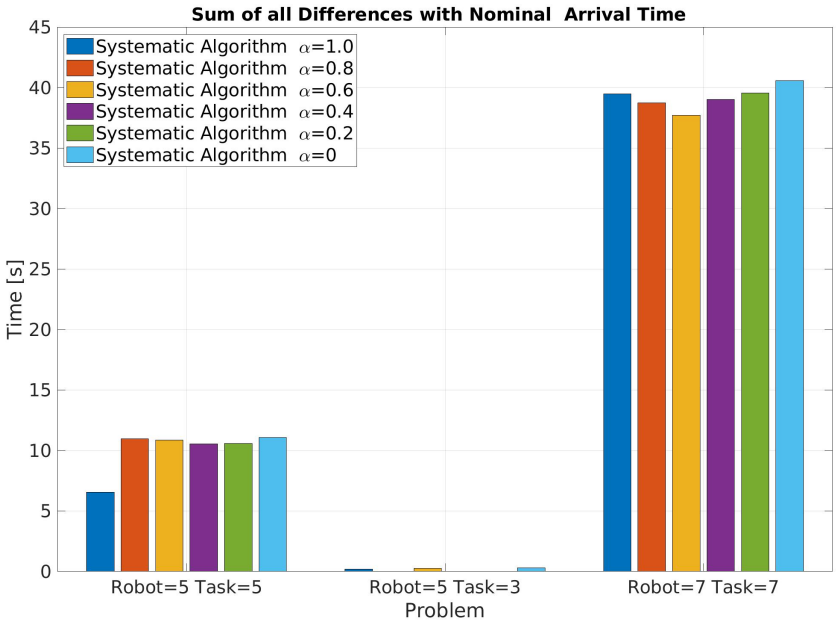


Figure 7.8: Difference between real and nominal arrival Time

7.3.2 Comparison with other Algorithms

Figure 7.9 shows the best solution find with Systematic Algorithm for each problem compared with the solution obtained, for the same problem, with both Local Search Algorithms described in Chapter 6; the quality of the solution is evaluated considering the arrival time of the entire fleet.

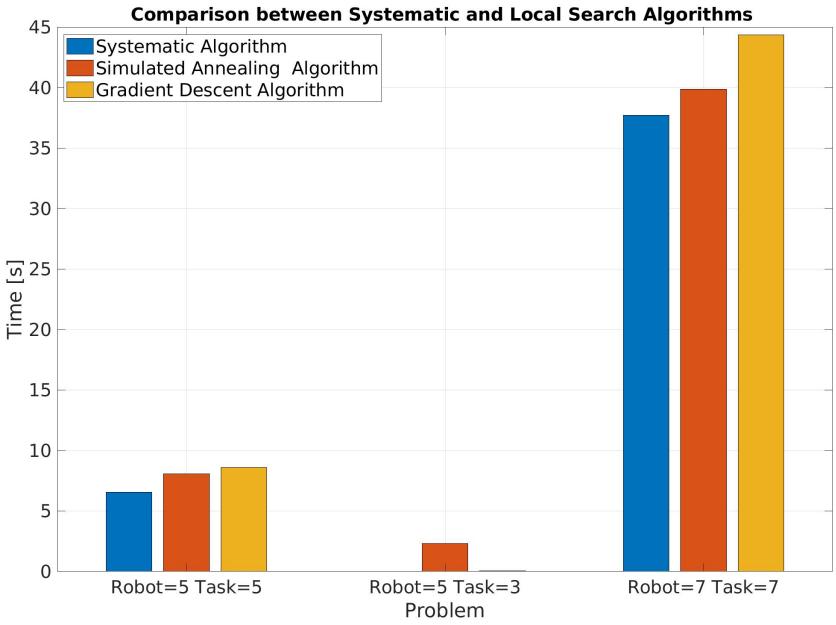


Figure 7.9: Comparison between Systematic and Local Search Algorithms

Figure 7.10 compares the best solution obtained through Systematic Algorithm with the solution obtained with the Greedy Algorithm.

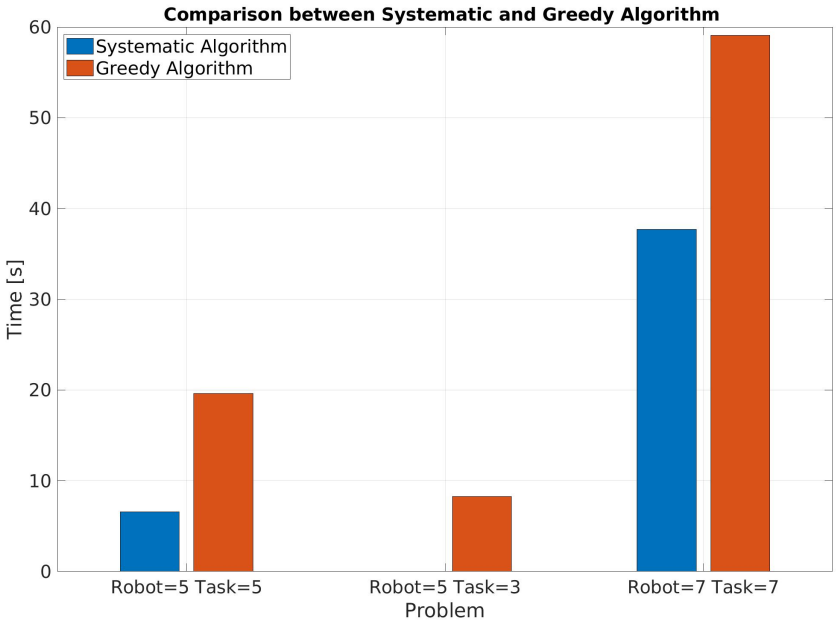


Figure 7.10: Comparison between Systematic and Greedy Algorithm

Considering the time to solve the optimization problem, Figure 7.11 shows the time required (in seconds) to find a solution for each Algorithm;

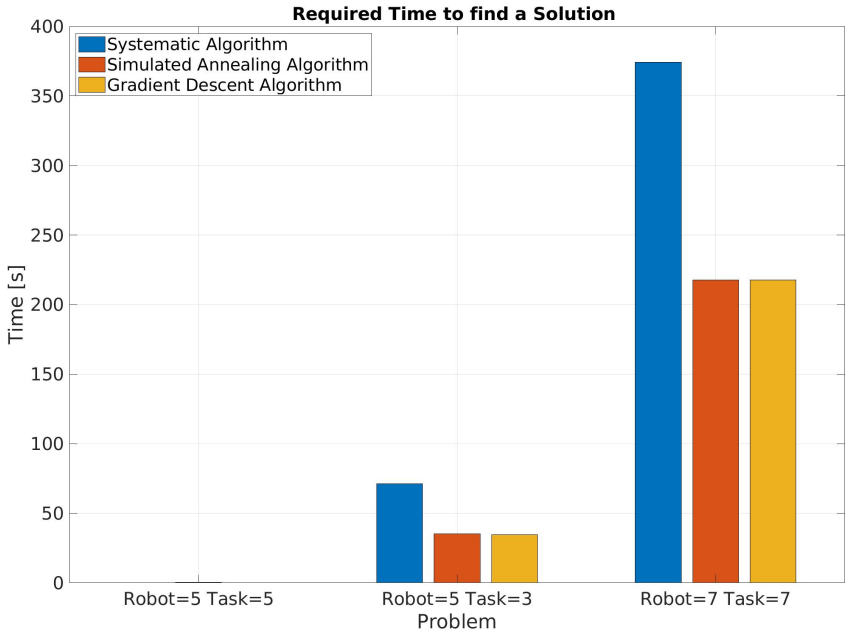


Figure 7.11: Required Time to Find a Solution

7.4 Map 1: Corridors

The map used for the second test is a Warehouse Map in Örebro; in this case, big obstacles and corridors are present. The map is shown in Figure 7.12

Figure 7.12: Map of a warehouse situate in Örebro

Four different problems of Task Allocation with these Map, where each problem is different from other for some aspects (e.g. numbers of robot e task, deadline, etc ...).

7.4.1 Varying parameters α

Firstly results obtained varying the linear parameters α are provided in Figures 7.13 and 7.14.

Figure 7.13 shows the max arrival time for each simulation when the linear parameter changes.

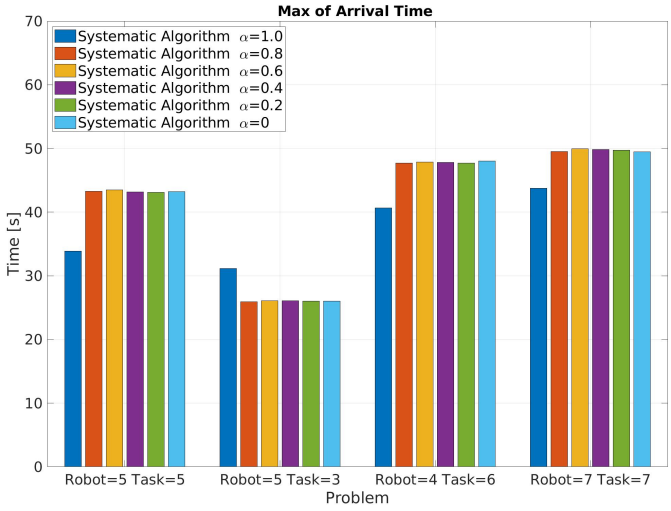


Figure 7.13: Test2: Max arrival time

Figure 7.14 shows sum of all difference between the nominal and the real arrival time

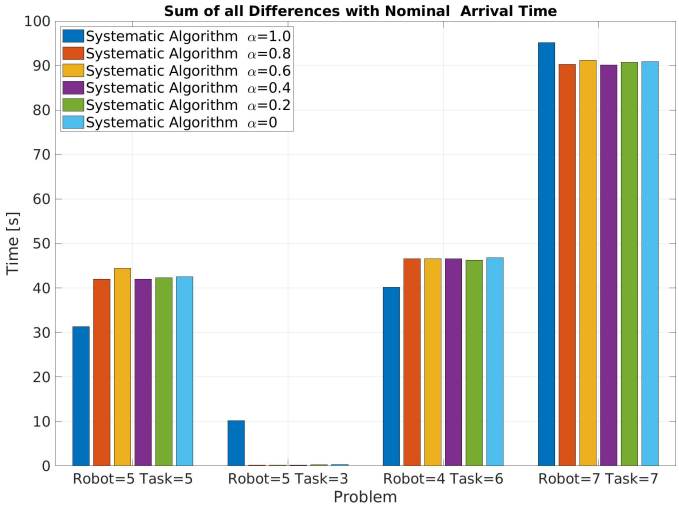


Figure 7.14: Difference between real and nominal arrival Time

7.4.2 Comparison with other Algorithms

Figure 7.15 shows the best solution find with Systematic Algorithm for each problem compared with the solution obtained, for the same problem, with both Local Search Algorithms described in Chapter 6; the quality of the solution is evaluated considering the arrival time of the entire fleet.

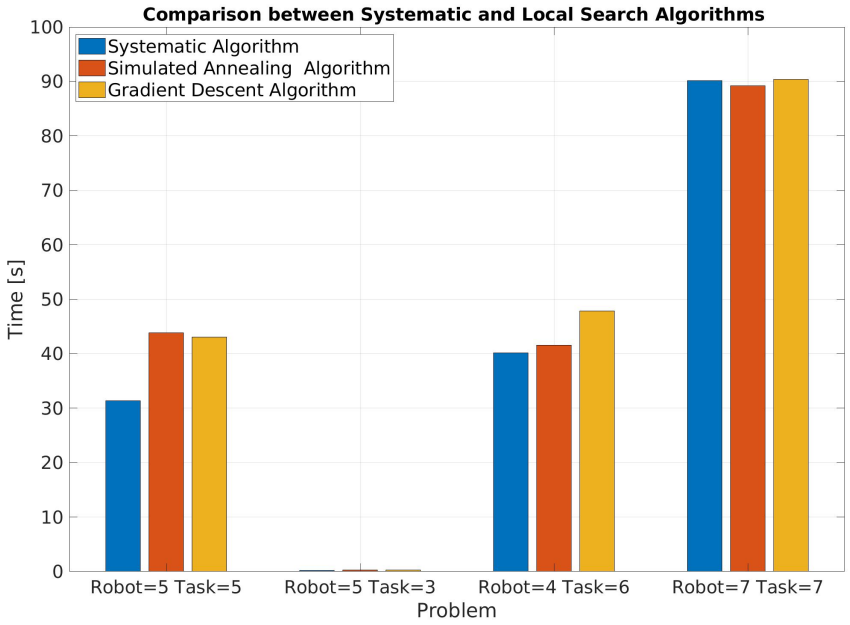


Figure 7.15: Comparison between Systematic and Local Search Algorithms

Figure 7.16 compares the best solution obtained through Systematic Algorithm with the solution obtained with the Greedy Algorithm.

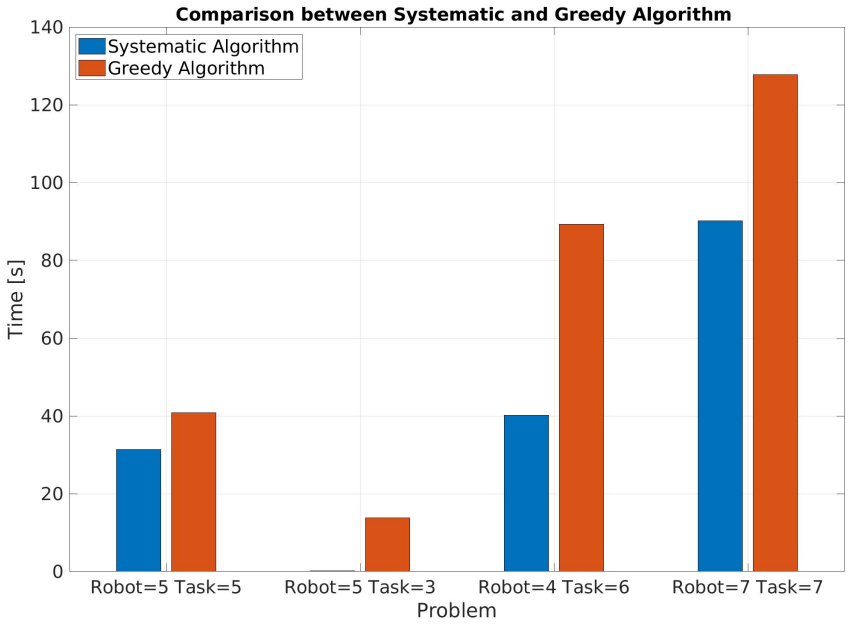


Figure 7.16: Comparison between Systematic and Greedy Algorithm

Considering the time to solve the optimization problem, Figure 7.17 shown the time required (in seconds) to find a solution for each Algorithm;

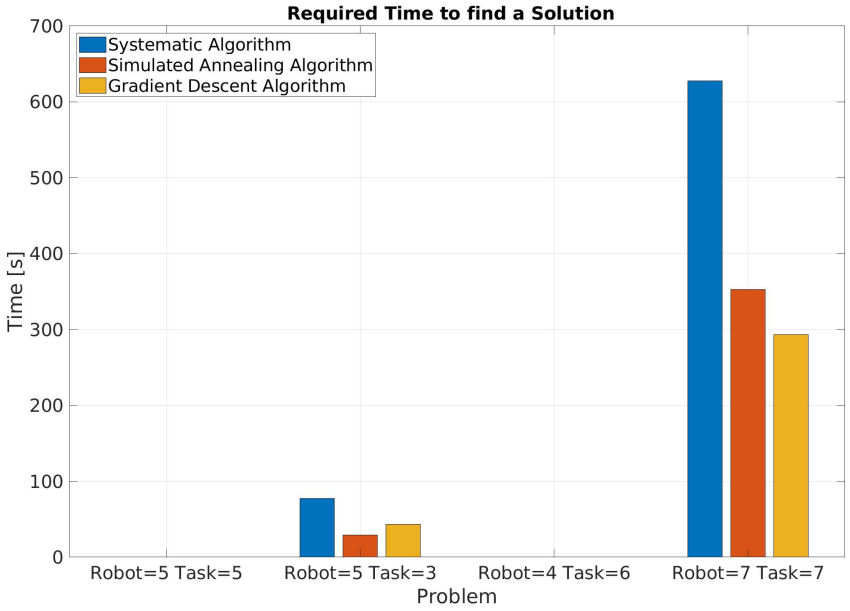


Figure 7.17: Required Time to Find a Solution

7.5 Map 2: Partial

The map used for the third test is a complex Map with more obstacles. The map is shown in Figure 7.18

Four different problems of Task Allocation with these Map, where each problem is different from other for some aspects (e.g. numbers of robot e task, deadline, etc . . .).

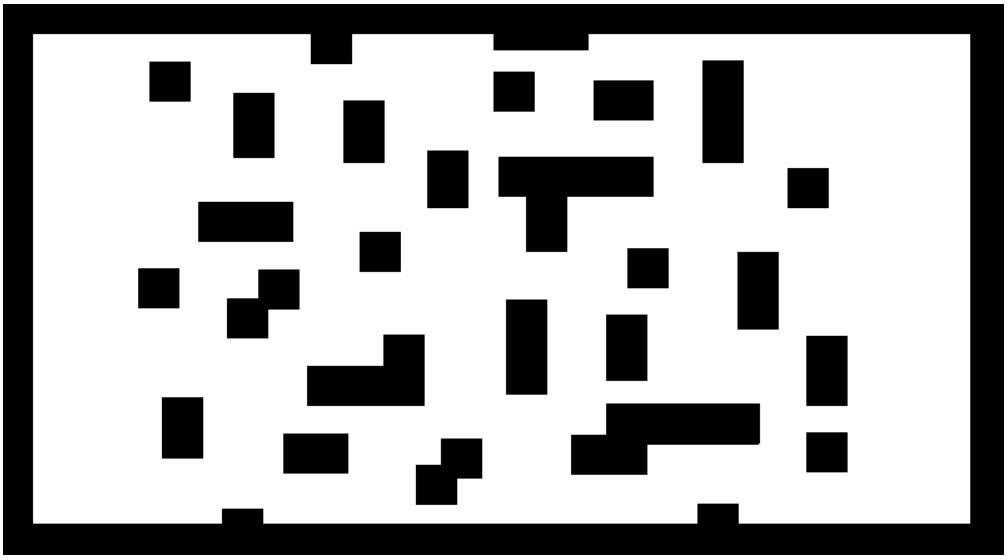


Figure 7.18: Map of the third Scenario

7.5.1 Varying parameters α

Firstly results obtained varying the linear parameters α are provided in Figures 7.19 and 7.20.

Figure 7.19 shows the max arrival time for each simulation when the linear parameter changes.

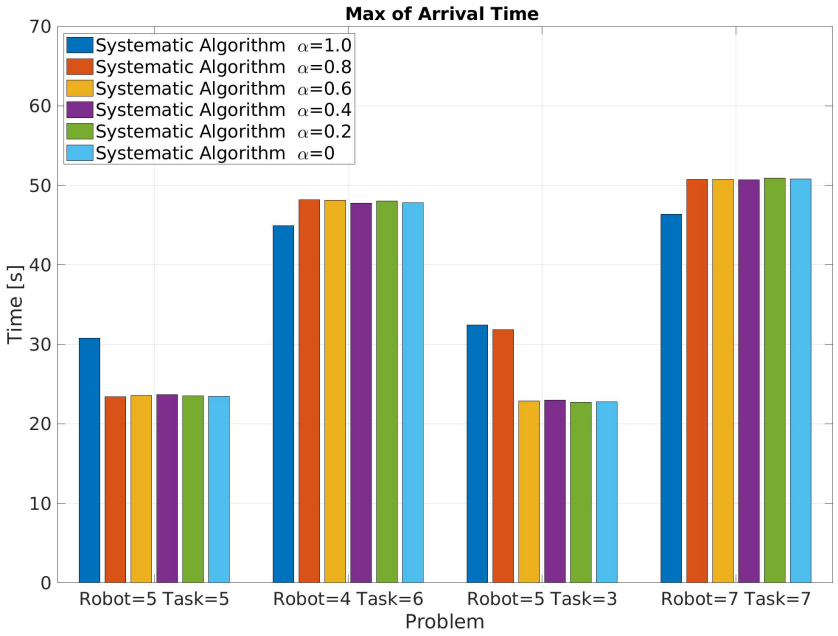


Figure 7.19: Test3: Max arrival time

Results reported in Figure 7.20 shown sum of all difference between the nominal and the real arrival time

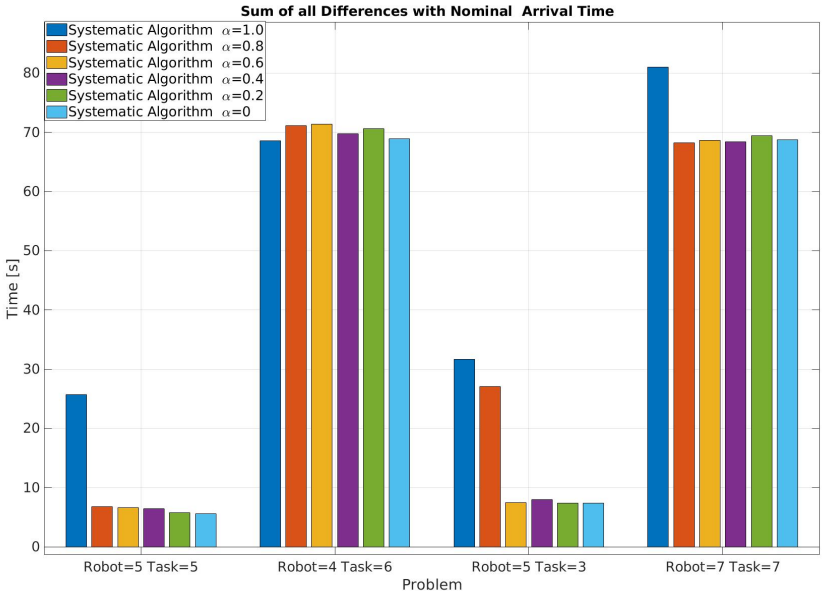


Figure 7.20: Difference between real and nominal arrival Time

7.5.2 Comparison with other algorithms

Figure 7.21 shows the best solution find with Systematic Algorithm for each problem compared with the solution obtained, for the same problem, with both Local Search Algorithms described in Chapter 6; the quality of the solution is evaluated considering the arrival time of the entire fleet.

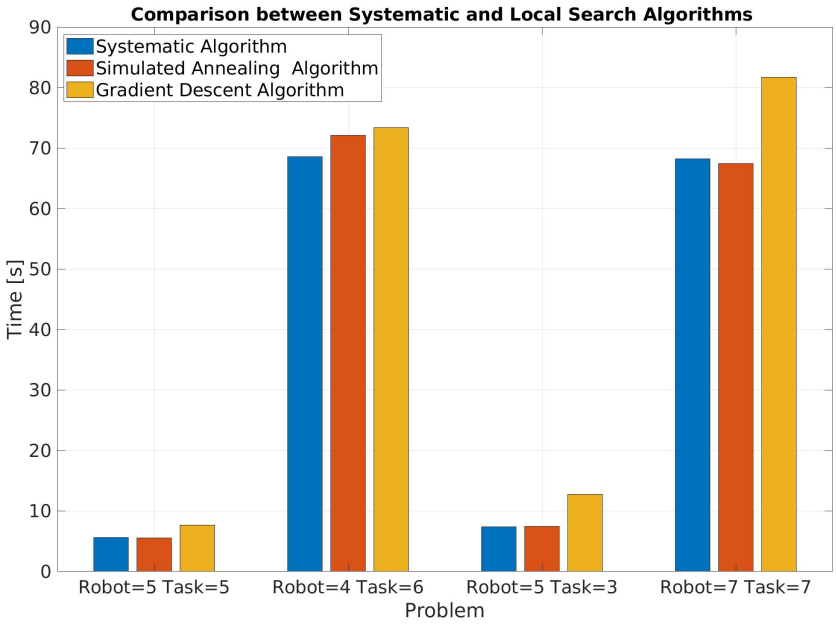


Figure 7.21: Comparison between Systematic and Local Search Algorithms

Figure 7.22 compares the best solution obtained through Systematic Algorithm with the solution obtained with the Greedy Algorithm.

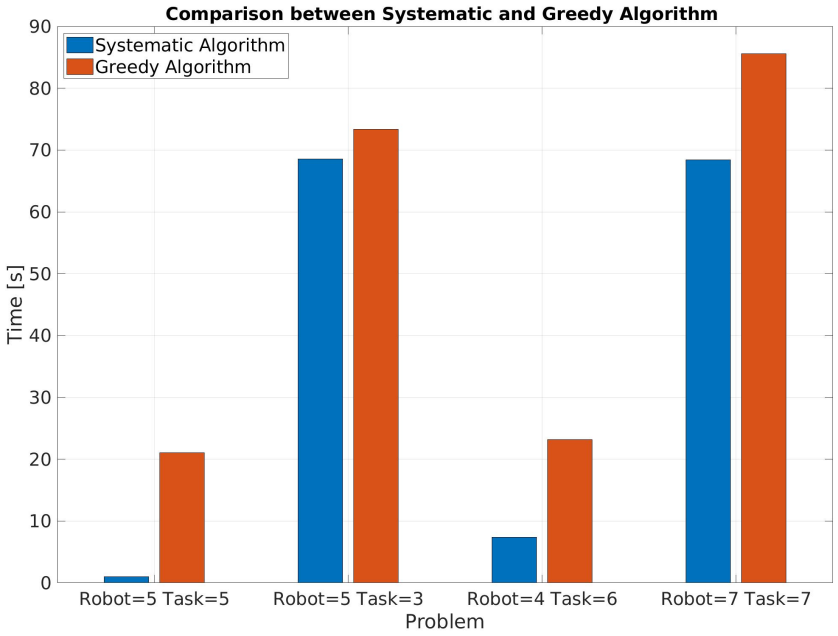


Figure 7.22: Comparison between Systematic and Greedy Algorithm

Considering the time to solve the optimization problem, Figure 7.23 shown the time required (in seconds) to find a solution for each Algorithm;

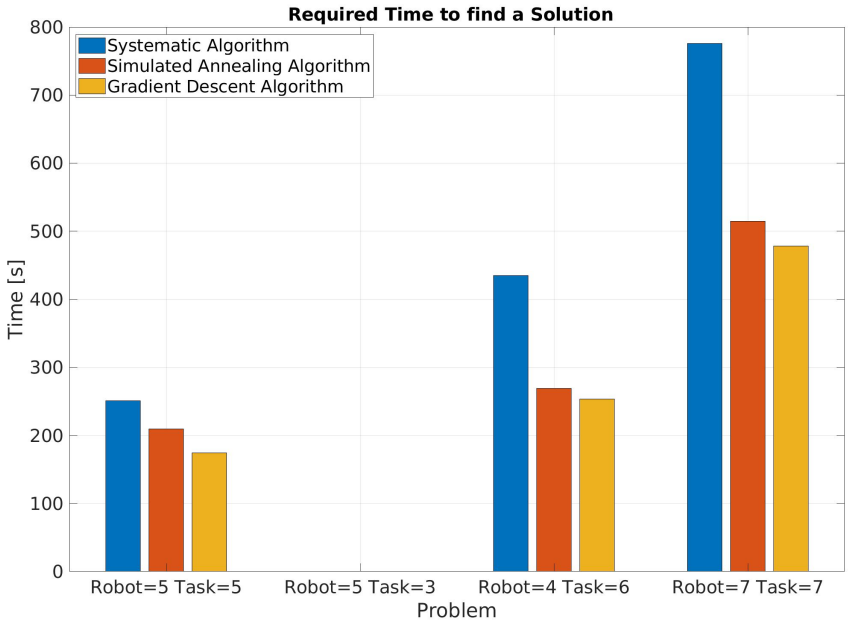


Figure 7.23: Required Time to Find a Solution

7.6 Map 3: Centro Piaggio

The map used for the fourth test is the map of Centro Piaggio in Pisa; this map is characterized by big presence of corridors and rooms. The map is shown in Figure 7.24

Three different problems of Task Allocation with these Map, where each problem is different from other for some aspects (e.g. numbers of robot e task, deadline, etc . . .).

Map is shown in Figure 7.24

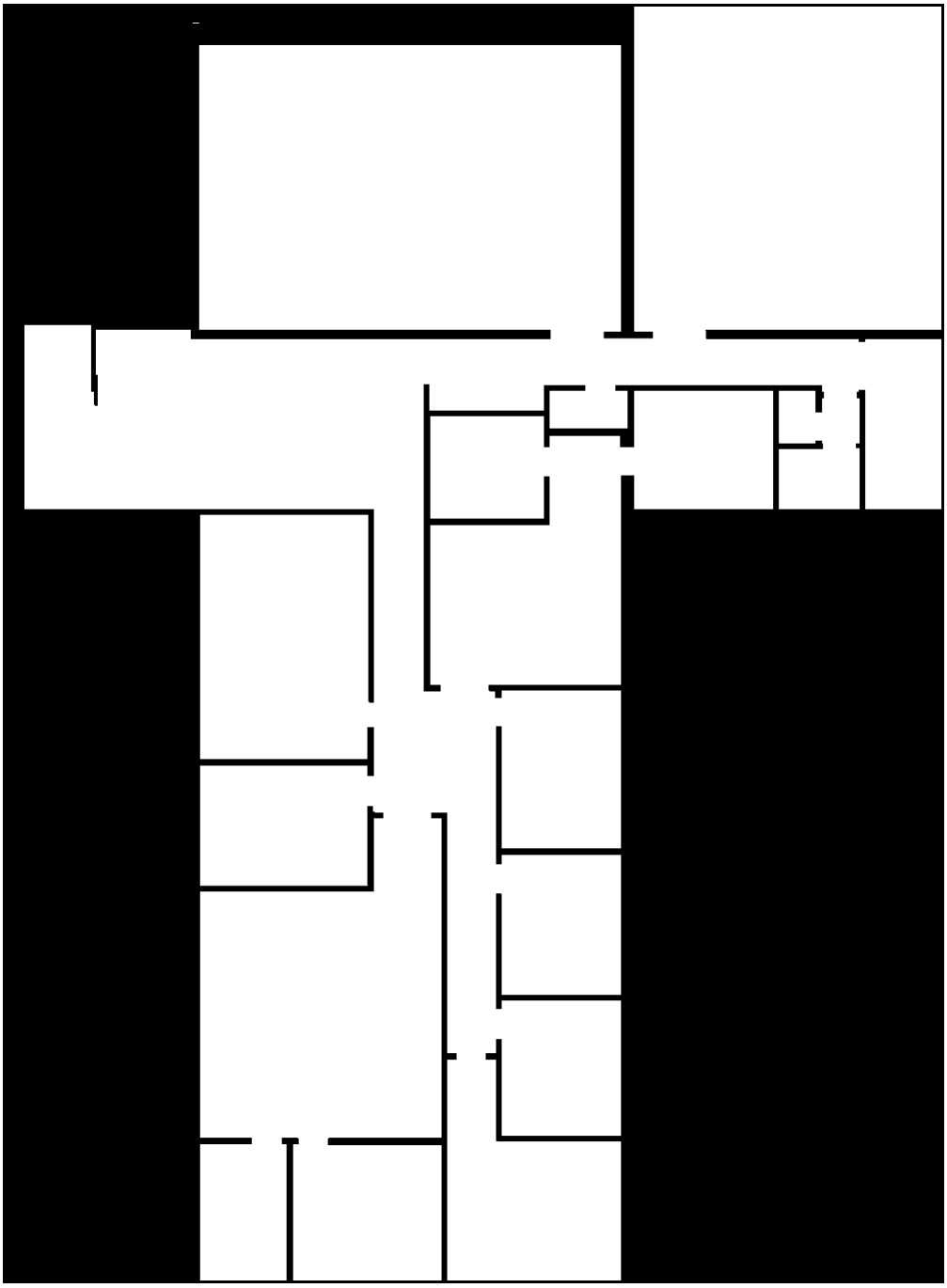


Figure 7.24: Map of Centro Piaggio situate in Pisa

7.6.1 Varying the parameter α

Firstly results obtained varying the linear parameters α are provided in Figures 7.25 and 7.26.

Figure 7.25 shows the max arrival time for each simulation when the linear parameter changes.

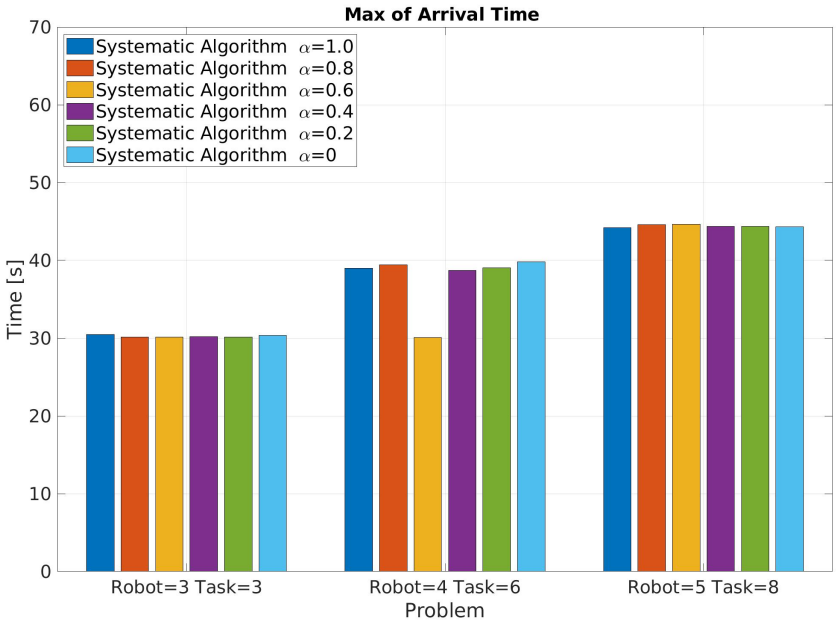


Figure 7.25: Test4: Max Arrival Time

Results reported in Figure 7.26 show sum of all difference between the nominal and the real arrival time

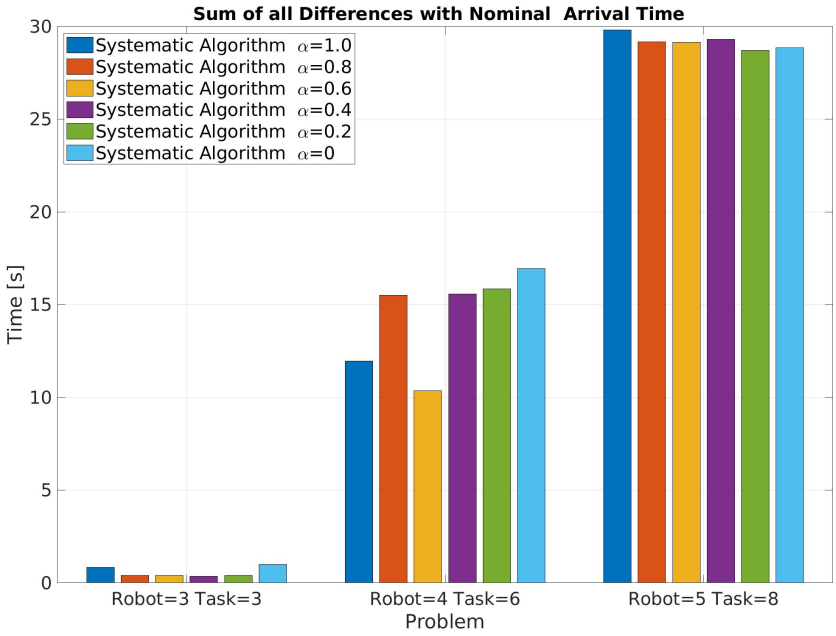


Figure 7.26: Difference between real and nominal arrival Time

7.6.2 Comparison with other algorithms

Figure 7.27 shows the best solution find with Systematic Algorithm for each problem compared with the solution obtained, for the same problem, with both Local Search Algorithms described in Chapter 6; the quality of the solution is evaluated considering the arrival time of the entire fleet.

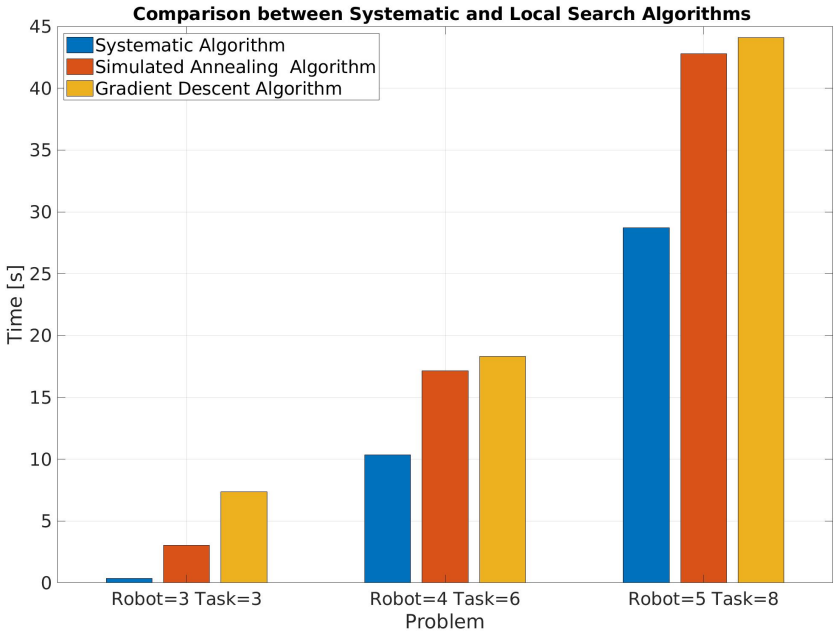


Figure 7.27: Comparison between Systematic and Local Search Algorithms

7.7 Discussion

Results show that there is not the best value of parameter α for all cases, but the best value change due to the problem. Nevertheless, in some test is possible to notice a trend on middle value ($0.4 < \alpha < 0.6$), so this can be a good starting value. Moreover, results show also that solutions do not change a lot varying the value of α; this is related to how the Function $\mathcal{F}$ is computed but also to the criteria to evaluate the goodness of a solution (a possible different criteria is the sum of all delay for each robot may furnish different results).

Regarding the comparison between algorithms, results show the fact that *Systematic Algorithm* furnish a better solution respect to *Local Search* Algorithms, since it guaranteed to find the optimal solution (if it exists) meanwhile, the solution of Local Search Algorithms can be a local minimum. It is possible to notice also a trend on the probability that Local Search and Systematic Algorithm give the same solution; this probability is higher for small instances, meanwhile, this probability decreases when the problem become bigger.

On one hand, the systematic algorithm guaranteed to find the optimal solution; on the other hand, the algorithm required a lot of time to find it when the $\mathcal{F}$ Function is considered, where the time increases in

an exponential way; Local Search Algorithm find a solution in less time but, again, this is no guaranteed that is the global optimal solution.
If, instead, the interference is not considered into the problem, the algorithm requires a very small amount of time to solve the problem since it can find the optimal solution at first iteration.

7.8 Blocking

Due to the presence of Function $\mathcal{F}$ every solution that ends with a blocking condition is avoided. Indeed, the delay returned by the Function $\mathcal{F}$ is infinity if the goal point of one of the robots ends in the critical section of the other.
An example of that can be seen in a simple example of some robots moving in line: Suppose that the robots start in a configuration as shown in Figure 7.28

Figure 7.28: Three Robots in Line: Starting Configuration

Suppose that tasks are simple a translation of robots along x-axis as reported in Figure 7.29

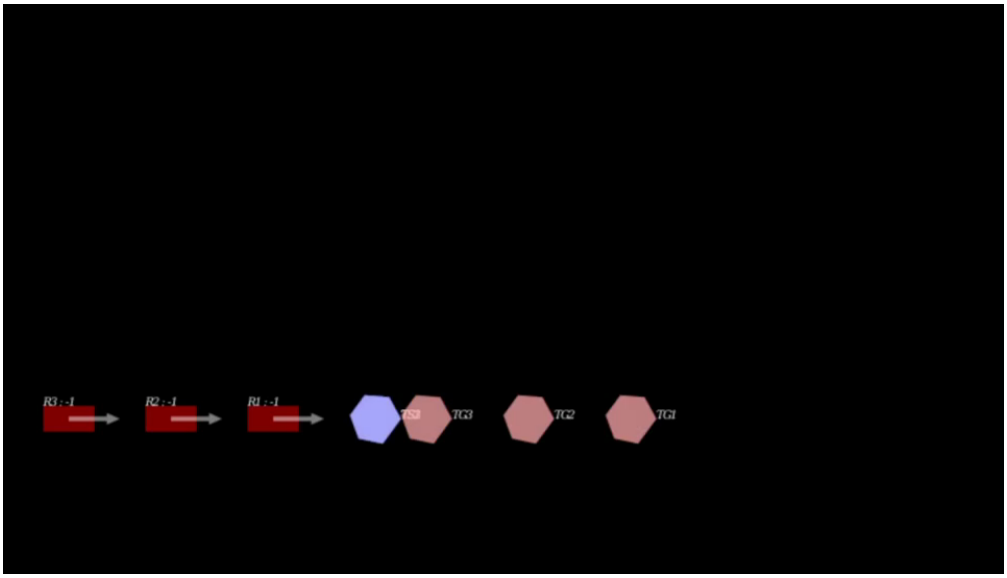


Figure 7.29: Three Robots in Line: Goal Positions

An optimal solution is a solution where the robots simply translate maintaining the same position as in starting configuration in the fleet. Applying the Systematic Algorithm, the solution is reported in Figure 7.30, where it is possible to notice that the optimal solution is found and that the allocation of tasks allow avoiding the blocking phenomena.

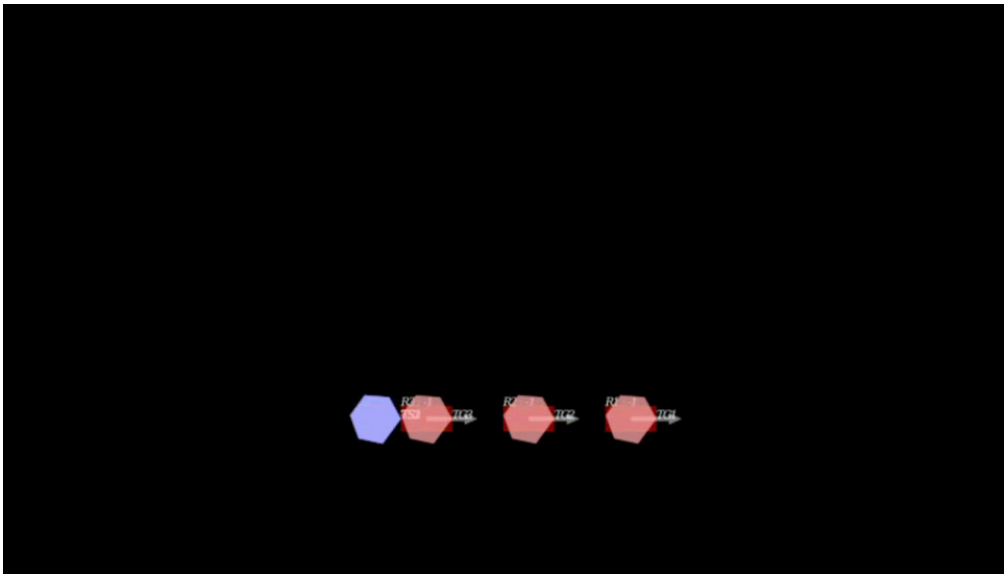


Figure 7.30: Three Robots in Line: Application of Systematic Algorithm

Considering instead the application of the Greedy Algorithm, the result is reported in Figure 7.31

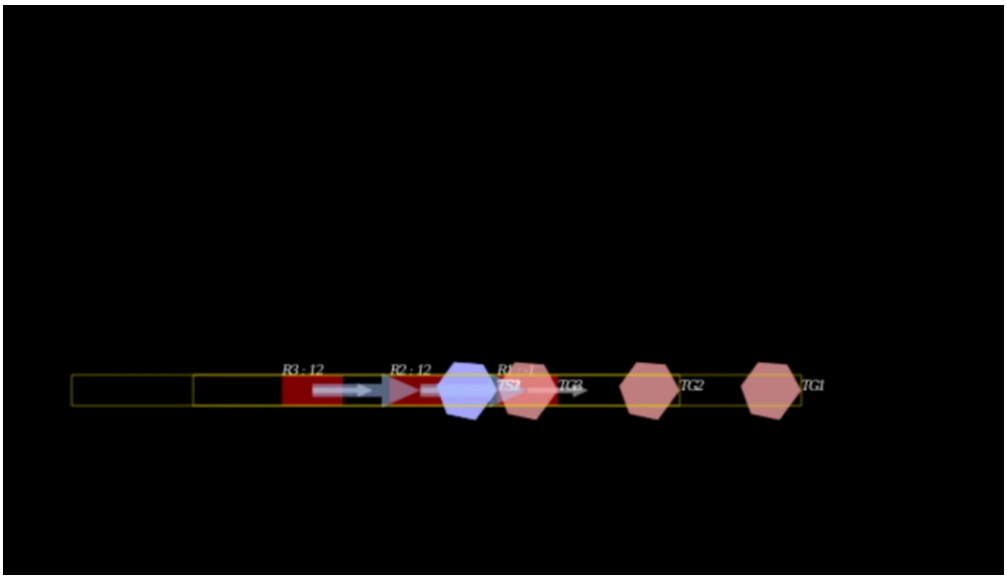


Figure 7.31: Three Robots in Line: Application of Greedy Algorithm

In this case the blocking is not automatically avoided since the first robot is assigned to the best task

for it that is the first one, but this brings to blocking situation since other robots cannot reach their final position.

It is possible to notice that in this case the blocking is not automatically avoided, given that the Greedy algorithm is a 'blind' algorithm and so when a task is allocated to a robot, the presence of other robots of the fleet is not considered.

Instead, the Systematic Algorithm avoids the blocking due to the presence of $\mathcal{F}$ Function; so in order to avoid that situation also when Systematic Algorithm $\alpha = 1$, some checks are applied in order to avoid that the first and only solution that is found can end with a blocking.

Chapter 8

Discussion and Conclusion

The aim of this thesis was to investigate heuristic optimization methods for assign tasks to fleets of mobile robots while taking into account the subsequent path planning problem and the coordination problem, where the OAP should take into account both the motions of individual robots and how these motions interfere with each other.

In the literature it is possible to find different algorithms that can solve the OAP, for example exact algorithms, local search algorithms, and genetic algorithms. In this study, a modified systematic algorithm is proposed. The choice of this kind of algorithm is related to the fact that the purpose of this work is to find the best assignment for a fleet of AVG (i.e. the global maximum of the objective function). In fact, a systematic algorithm guaranteed to find the optimal solution (if it exists).
Comparing to classical systematic algorithms that can be found in literature, the proposed algorithm has additional constraints that may reduce the number of feasible solutions and so the computational cost of the algorithm. Comparing the developed systematic algorithm with local search methods for solving the same problem, is possible to notice that this algorithm often provided a better solution, since local search algorithms may find only a local optimum of the objective function.

On one hand, this algorithm can find the optimal solution (if it exists) in a reasonable time both on small instances (if interference is considered) and on big instances (if interference is not considered). On the other hand, aiming at larger problems, some changes are introduced that allow finding an optimal assignment quickly even for problems of considerable size. This algorithm is capable to find a solution that avoids the deadlock situation considering the interference function.

Considering the results obtained through simulation and comparison with other algorithms, these bode well for an application of this algorithm to different MRTA problems although the proposed algorithm required more time to solve the problem respect to local search algorithms. Moreover, the additional constraint does not guarantee to eliminate some unfavourable or sub-optimal solutions, so in the worst case all possible solutions are analyzed. Future work will aim at considering also cooperative tasks and

a more refined traffic $\mathcal{F}$ Function. In conclusion, this algorithm can be used for real applications in environments where is possible to find different types of robots and tasks.

Bibliography

[1] Thapa M. Dantzig G. *Linear programming: Introduction*, volume Volume 1. Springer, 1997.

[2] Lorenzo Sabattini, Valerio Digani, Cristian Secchi, and Cesare Fantuzzi. Optimized simultaneous conflict-free task assignment and path planning for multi-agv systems. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1083–1088. IEEE, 2017.

[3] Changjoo Nam and Dylan A Shell. Assignment algorithms for modeling resource contention in multirobot task allocation. *IEEE Transactions on Automation Science and Engineering*, 12(3):889–900, 2015.

[4] Stefano Giordani, Marin Lujak, and Francesco Martinelli. A distributed algorithm for the multi-robot task allocation problem. pages 721–730, 06 2010.

[5] Lorenzo Sabattini, Valerio Digani, Matteo Lucchi, Cristian Secchi, and Cesare Fantuzzi. Mission assignment for multi-vehicle systems in industrial environments. *IFAC-PapersOnLine*, 48(19):268 – 273, 2015. 11th IFAC Symposium on Robot Control SYROCO 2015.

[6] M. Turpin, N. Michael, and V. Kumar. Concurrent assignment and planning of trajectories for large teams of interchangeable robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 842–848, 2013.

[7] Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory planning and assignment in multirobot systems. 01 2013.

[8] A. Settimi and L. Pallottino. A subgradient based algorithm for distributed task assignment for heterogeneous mobile robots. In *52nd IEEE Conference on Decision and Control*, pages 3665–3670, 2013.

[9] Brian Gerkey and Maja Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotic Res.*, 23:939–954, 09 2004.

[10] Brian Gerkey and Maja Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotic Res.*, 23:939–954, 09 2004.

[11] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[12] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[13] Mukund N. Thapa George B. Dantzig. *Linear Programming: Theory and extensions*, volume Volume 2. Springer, 1 edition, 2003.

[14] Chien-Chung Shen and Wen-Hsiang Tsai. A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Transactions on Computers*, C-34(3):197–203, March 1985.

[15] I. Ahmad and M. Kafil. A parallel algorithm for optimal task assignment in distributed systems. pages 284–290, March 1997.

[16] Irfan Younas, Farzad Kamrani, Christian Schulte, and Rassul Ayani. Optimization of task assignment to collaborating agents. 04 2011.

[17] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer-Verlag Berlin Heidelberg, 2 edition, 2001.

[18] Gamal Attiya and Yskandar Hamam. Optimal allocation of tasks onto networked heterogeneous computers using minimax criterion. 01 2003.

[19] Hongtao Hu, Yiwei Wu, and Tingsong Wang. A metaheuristic method for the task assignment problem in continuous-casting production. *Discrete Dynamics in Nature and Society*, 2018:1–12, 10 2018.

[20] John W. Chinneck. *Practical Optimization: A Gentle Introduction*, volume Chapter 13. 06 2015.

[21] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics - Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer, 2nd printing. edition, 2008.

[22] Quang Vinh Dang, Izabela Nielsen, Kenn Steger-Jensen, and Ole Madsen. Scheduling a single mobile robot for part-feeding tasks of production lines. *Journal of Intelligent Manufacturing*, 25, 12 2014.

[23] Quang Vinh Dang, Izabela Nielsen, and Kenn Steger-Jensen. Mathematical formulation for mobile robot scheduling problem in a manufacturing cell. *IFIP Advances in Information and Communication Technology*, 384:37–44, 01 2012.

[24] Anna Mannucci, Lucia Pallottino, and Federico Pecora. Provably safe multi-robot coordination with unreliable communication. *IEEE Robotics and Automation Letters*, 4(4):3232–3239, 2019.

[25] Federico Pecora, Henrik Andreasson, Masoumeh Mansouri, and Vilian Petkov. A loosely-coupled approach for multi-robot coordination, motion planning and control. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*, 2018.

[26] Paolo Forte. Task Assignment Algorithms. `https://github.com/PaoloForte95/coordination_oru`, 2019.

[27] Laurent Perron and Vincent Furnon. OR-Tools Software. `https://developers.google.com/optimization/`, 2019.

[28] Laurent Perron and Vincent Furnon. OR-Tools Installation. `https://developers.google.com/optimization/install`, 2019.