On Provably Safe and Live Multirobot Coordination With Online Goal Posting

Anna Mannucci, Lucia Pallottino, and Federico Pecora

***Abstract*—A standing challenge in multirobot systems is to realize 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 article, we propose a centralized, 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 behavior 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*—Formal methods in robotics and automation, intelligent and flexible manufacturing, multirobot systems, planning, scheduling and coordination.**

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) 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, and (R6) 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 centralized 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 noncooperative tasks), and assume decoupled motion planning and control (which holds, e.g., when robots are driven by car-like or differential-drive kinodynamics). Precedence constraints are computed and revised online (while accounting for kinodynamic feasibility) to safely regulate access to and progress through contiguous overlapping configurations of pairs of paths.

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 contributions of this article aim to overcome this limitation, specifically, as follows.

1) We formally define the conditions necessary to ensure liveness by design, that is, avoiding blocking, deadlocks, and livelocks. This is achieved by generalizing 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 replanning.
2) We 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 to discard goals and paths that lead to blocking; (b) three extensions of the original algorithm to 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.

Manuscript received September 18, 2020; revised March 9, 2021; accepted April 17, 2021. Date of publication June 8, 2021; date of current version December 6, 2021. This work was supported in part by the EU's Horizon 2020 research and innovation program under Grant 732737 (ILIAD), in part by the Swedish Knowledge Foundation (KKS) under the Semantic Robots research profile, and Vinnova under Project AutoHauler. This paper was recommended for publication by Associate Editor J. O'Kane and Editor P. R. Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Anna Mannucci.)*

Anna Mannucci is with the Research Center "E. Piaggio," University of Pisa, Italy, and Dipartimento di Ingegneria dell'Informazione, University of Pisa, Pisa 56126, Italy, and also with the Center for Applied Autonomous Sensor Systems (AASS), Örebro University, 70281 Örebro, Sweden (e-mail: anna.mannucci90@gmail.com).

Lucia Pallottino is with the Research Center "E. Piaggio," University of Pisa, Italy, and Dipartimento di Ingegneria dell'Informazione, University of Pisa, 56126 Pisa, Italy (e-mail: lucia.pallottino@unipi.it).

Federico Pecora is with the Center for Applied Autonomous Sensor Systems (AASS), Örebro University, 702 81 Örebro, Sweden (e-mail: federico.pecora@oru.se).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TRO.2021.3075371.

Digital Object Identifier 10.1109/TRO.2021.3075371

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

We empirically evaluate performance of all proposed solutions with several different heuristics in terms of time and rate of mission completion. Experiments with simulated robots both in benchmark scenarios and in a real mine (elicited by an ongoing collaboration with industries, see Section VIII-D2) confirm the effectiveness of the approach for realistically sized fleets (≤ 40 robots tested) with reasonable computational complexity.

The rest of the article is organized as follows. Relevant state-of-the-art is presented in Section II. The problem tackled in this article 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–VII formally analyze the factors that may prevent the supervisory controller of [3], [4] from ensuring liveness, proposing strategies for overcoming this limitation while ensuring safety. Simulations, analysis and discussion of results are given in Sections VIII and IX, respectively, and Section X concludes this article.

II. State-of-the-Art

The literature addressing multirobot coordination is vast,[1] spanning multiarm 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 *prioritized 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. Prioritized 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 multirobot 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 decentralized fashion with a token-based approach. However, both methods assume holonomic disc-shaped fleets, and require robots to be synchronized 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 generalizes 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 decentralized, prioritized, receding horizon version of the velocity tuning method to manage

[1]The scope of this section is not to give a comprehensive overview of multirobot 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 multirobot 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].

TABLE I
STATE-OF-THE-ART: A SUMMARY OF SELECTED STRATEGIES

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	$O(\lvert\mathcal{C}\rvert)$ / $O(\lvert\mathcal{C}\rvert n^2)$ / $O(\lvert\mathcal{C}\rvert 2^{n\log n})$	✓	✓	✓	✓	✓	✓

D, decentralized or distributed; C, centralized; $\mathcal{O}$, set of obstacles other than 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] and this approach are robust to clock desynchronization.

possibly heterogeneous nonholonomic vehicles at traffic intersections. Priorities are used to define the sequence at which each robot solves its own optimization problem (passing before all or after all the other robots which have already decided). Although this leads to suboptimal 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 article 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 synchronized 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 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 precomputed 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 summarizes 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 summarized in the table are those that maximize achievement of the requirements (R1–R6) outlined in Section I.

III. PROBLEM DEFINITION

Multirobot Fleet. We describe our multirobot 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 $q_i\in\mathcal{Q}_i$; $f_i(q_i,\dot{q}_i)\le 0$ 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.

Robot Goals. When idle, a robot i may be assigned to a *noncooperative, 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 noncooperative 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 $f_i(q_i,\dot{q}_i)\le 0$ [see Fig. 1(a)]. Idle robots are associated with a path of length

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

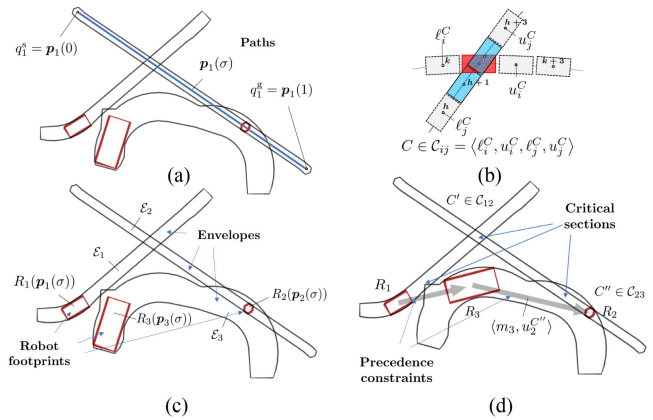

Fig. 1. Main concepts of the approach in [3], [4].

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 both synthesizing and revising at runtime 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 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 $Q\subset\bigcup_{i\in\mathcal{R}}\mathcal{Q}_i \Rightarrow \mathcal{E}_i\cap\bigcup_{q_i\in Q} R_i(q_i)=\emptyset$.

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 $\sigma_j(t)<u_j^C\Rightarrow\sigma_i(t)\leq m_i$, 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

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

$$r_{ij}(\mathrm{k})=\sup_{\sigma\in[\sigma_i(t_i),u_i^C]}\left\{\mathcal{E}_i^{[\sigma_i(t_i),\sigma]}\cap\mathcal{E}_j^{[\sigma_j(t_j),u_j^C]}=\emptyset\right\} \quad (1)$$

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.,

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

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.

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

Algorithm 1: Coordination at time k.

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

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.

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

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 $\mathrm{k} \ge 1$, 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 $\sigma_j(t_j+\Delta_j^{\mathrm{stop}}) \le \ell_j^C$, $t_j \in [\mathrm{k}T_c, (\mathrm{k}+1)T_c)$ (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 [4] for details).

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 s (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 $T_c \ge \max_{i\in\mathcal{R}} T_i$. Robots

[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.

Algorithm 3: The revise function with global reordering (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; $\bar{\Sigma}^{\mathrm{old}}$ 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.

$\mathcal{T}^{\mathrm{rev}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{del}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{add}} \leftarrow \emptyset$, $\mathcal{T}^{\mathrm{upd}} \leftarrow \emptyset$, $\mathcal{T}^{l} \leftarrow \emptyset$;
$D_\infty = \emptyset$, $\mathcal{C}^{\mathrm{rev}} \leftarrow \emptyset$;
foreach $C \in \mathcal{C}_{ij}, \mathcal{C}_{ij} \in \mathcal{C}$ do
  if $\sigma_i(t_i) \ge u_i^C \vee \sigma_j(t_j) \ge u_j^C$ then
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$;
    $(h,k) \leftarrow$ get order from $\mathcal{T}^{\mathrm{old}}$;
    $\mathcal{T}^{\mathrm{del}} \leftarrow \mathcal{T}^{\mathrm{del}} \cup \{\langle m_h, u_k^C\rangle\}$;
  else
    if $C \in \mathcal{C}^{\mathrm{new}}$ then
      $(h,k) \leftarrow$ get FCFS order;
      $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
      $\mathcal{T}^{\mathrm{add}} \leftarrow \mathcal{T}^{\mathrm{add}} \cup \{\langle m_h, u_k^C\rangle\}$;
    else
      $(h,k) \leftarrow$ get previous order from $\mathcal{T}^{\mathrm{old}}$;
      $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
      $\mathcal{T}^{\mathrm{upd}} \leftarrow \mathcal{T}^{\mathrm{upd}} \cup \{\langle m_h, u_k^C\rangle\}$;
    if $\sigma_i(t_i+\Delta_i^{\mathrm{stop}}) \le \ell_i^C \wedge \sigma_j(t_j+\Delta_j^{\mathrm{stop}}) \le \ell_j^C$ then
      $\mathcal{C}^{\mathrm{rev}} \leftarrow \mathcal{C}^{\mathrm{rev}} \cup \{C\}$;
    $\mathcal{T}^{l} \leftarrow \mathcal{T}^{l} \cup \{\langle m_h, u_k^C\rangle\}$;
$\mathcal{T}^{l} \leftarrow \mathcal{T}^{\mathrm{upd}} \cup \mathcal{T}^{\mathrm{add}}$;
$(D_\infty, \mathcal{L}) \leftarrow \mathrm{updateGraph}(D_\infty, \mathcal{L}, \mathcal{T}^{\mathrm{old}}, \mathcal{T}^{\mathrm{del}}, \mathcal{T}^{\mathrm{add}})$;
foreach $C \in \mathcal{C}^{\mathrm{rev}}$ do
  $(h,k) \leftarrow$ get order from heuristic while enforcing condition (ii) and (iii) of Theorem 3 (or (i) of Theorem 4);
  if $\langle m_h, u_k^C\rangle \notin \mathcal{T}^{l}$ then
    $m_h \leftarrow$ update according to (1), $s_h(t_h)$ and $s_k(t_k)$;
    $\mathcal{T}' \leftarrow \mathcal{T}^{l} \setminus \{\langle m_k, u_h^C\rangle\} \cup \{\langle m_h, u_k^C\rangle\}$;
    $(D'_\infty, \mathcal{L}') \leftarrow \mathrm{updateGraph}(D_\infty, \mathcal{L}, \mathcal{T}^{l}, \langle m_k, u_h^C\rangle, \langle m_h, u_k^C\rangle)$;
    live $\leftarrow$ true;
    foreach $w \in \mathcal{L}'(e_{hk}) - \mathcal{L}(e_{hk})$ do
      foreach $\Phi(w) \subseteq \mathcal{T}'$ do
        if $\Phi(w)$ is nonlive then
          live $\leftarrow$ false;
          break;
      if live then $(\mathcal{T}^{l}, D_\infty, \mathcal{L}) \leftarrow (\mathcal{T}', D'_\infty, \mathcal{L}')$;
$\mathcal{T}^{\mathrm{rev}} \leftarrow \mathcal{T}^{l}$;
return $(D_\infty, \mathcal{L}, \mathcal{T}^{\mathrm{rev}})$

are not required to be synchronized 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 an unreliable transmission protocol as described in [4], which ensures safety by relating number of retransmissions 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 1) $\Delta_i^{\mathrm{stop}} \geq T_c$, as in Algorithm 1, sampling occurs at the beginning (line 1), while communication occurs at the end (line 7); 2) $g_i(q_i, \dot{q}_i, \ddot{q}_i, u_i^{\mathrm{acc}}, u_i^{\mathrm{dec}}, t)$ is a conservative model of robot i's dynamic; 3) 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,[5] 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 characterize 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 safety holds 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 and under A6, the following three 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].

[5]For example, 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, failure of the overall communication network or the coordinator, or a malicious dynamic obstacle.

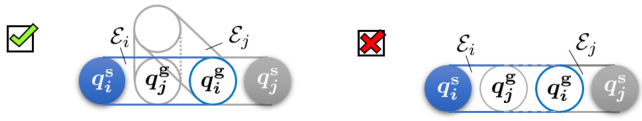

Fig. 2. Admissible (left) and not admissible (right) envelopes.

- 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 liveness 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, we aim to define *how* to ensure that $\mathcal{E}(\mathrm{k})$ remains admissible for all k.

A possible strategy 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 $\mathcal{Q}_i^{\mathrm{free}} \times \mathbb{R}$, 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 replanned. 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, progressively defining the conditions and methods needed to *avoid*, *prevent*, or *repair* blocking (Section V), deadlocks (Section VI), and livelocks (Section VII). 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

TABLE II
A GUIDE TO STRATEGIES PROPOSED IN SECTIONS V–VII

	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)	—

prevent blocking/deadlock/livelock from happening. Prevention strategies may require knowledge of, and affect, the motions of all robots in the fleet. If these 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 run-time, and precedences can be changed online. For the reader's convenience, a short summary of the strategies proposed in Sections V–VII is given in Table II.

V. BLOCKING

The concept of blocking is strictly related to the exclusive spatio-temporal ownership of destinations [6], [9]. In this section, we define a set of sufficient conditions on the set of active goals $\mathcal{G} = \bigcup_{i\in\mathcal{R}} \boldsymbol{p}_i(1)$ 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). As summarized in Table II, we enforce blocking avoidance by using a generalization of the well-formed infrastructure concept [6] (hence by constraining the locations of goals), which ensures the multirobot trajectory planning problem is solvable with any schedule of goals (Section V-A). Also, we achieve global blocking prevention with *any online posted* goal by formulating an admissibility check which precautionarily prevents committing to possibly blocking goals (Section V-B). Local prevention and repair will be addressed via replanning, as discussed in Section VI-D2a.

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 $\mathbb{G}^{j\neq i}(q_i^{\text{g}}) = \{q_j^{\text{g}} \in \mathbb{G}_j : R_i(q_i^{\text{g}}) \cap R_j(q_j^{\text{g}}) = \emptyset\}$. 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[6] 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 $f_i(q_i, \dot{q}_i) \leq 0$.

[6]We here extend the definition provided in [6] to tackle kinematic constraints and generic robot footprints.

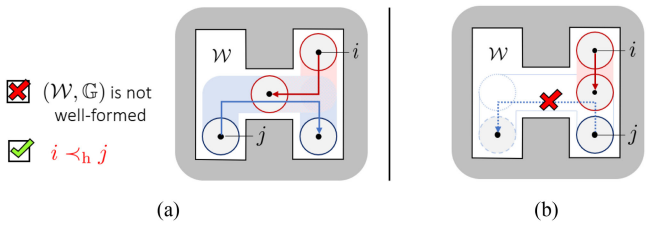

Fig. 3. (a) Example where Theorem 2 may be too conservative: even if $(\mathcal{W}, \mathbb{G})$ is not well-formed, there exists a feasible $\mathcal{T}$ s.t. both the robots may reach their destination. (b) Example of entrapment: robot j cannot accept the new mission (dotted arrow) to prevent being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

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 $q_i^{\text{s}}(0) \in \mathbb{G}_i$. At time t, a new path $\boldsymbol{p}_i$ to $q_i^{\text{g}} \in \mathbb{G}_i$ can be accepted while ensuring safety and $\neg$(F1) if $\boldsymbol{p}_i$ is $\mathbb{G}^{j\neq i}(q_i^{\text{g}})$-avoiding and $R_i(q_i^{\text{g}}) \cap R_j(q_j^{\text{g}}(t)) = \emptyset, \forall j \neq i$.

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 run-time (R2), then Theorem 2 is too conservative [see Fig. 3(a)]. To address this limitation, similarly to [9], we formulate Theorems 3 and 4 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 replan its path 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 5), 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-D2.

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 $t_0 + \Delta^{\text{plan}}$. The new path $\boldsymbol{p}_i$ can be accepted while ensuring P1 and $\neg$(F1) if:

1) $\mathcal{C}^{\text{end}}(\mathcal{E}_i, \mathcal{E}_j) \cap \mathcal{C}^{\text{end}}(\mathcal{E}_j, \mathcal{E}_i) = \emptyset$ for all $(i, j \neq i) \in \mathcal{R}^2$;
2) for each $C \in \mathcal{C}^{\text{end}}(\mathcal{E}_j, \mathcal{E}_i)$, it is possible to impose $\langle m_j, u_i^C \rangle \in \mathcal{T}$ while C is active;
3) for each $C \in \mathcal{C}^{\text{end}}(\mathcal{E}_i, \mathcal{E}_j)$, it is possible to impose $\langle m_i, u_j^C \rangle \in \mathcal{T}$ 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 and conditions 2 and 3 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 1 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 2 and 3 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 conditions 2 and 3 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 there may exist 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 [43]. 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. The new path $\boldsymbol{p}_i$ can be accepted while ensuring P1, $\neg$(F1) and preventing entrapment if

1) it satisfies Theorem 3.
2) 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 $\{q_i^{\text{g}}, q_i^{\text{b}}\}$;
3) 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 2.

Proof: The sketch of proof builds upon Theorem 3, which ensures that P1 and $\neg$(F1) hold due to A1. Also, note the following.

(a) Since paths are updated sequentially, condition 2 ensures q_i^{b} to be reachable at t_0 and condition 3 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 $q_j^{\text{g}} \notin \mathbb{G}_i$, i.e., $q_j^{\text{g}} \neq q_j^{\text{b}}$. Furthermore, conditions 2 and 3 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 $\boldsymbol{p}_j^{\text{b}}(t_0)$ interfering with the new $\boldsymbol{p}_i \cup \boldsymbol{p}_i^{\text{b}}$), 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 2 and 3). 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

In this section, we address F2, characterizing deadlocks and nonlive states (i.e., states which may lead to deadlocks [44]) and relating them to the set $\mathcal{T}$ (Section VI-A). As summarized in Table II, we first analyze the conditions under which the heuristic function h avoids deadlocks (Section VI-B). Aiming at increasing flexibility, we formalize a global prevention strategy based on reversing precedence orders to prevent nonlive sets in $\mathcal{T}(\mathrm{k})$ (Section VI-C). We formally prove the strategy is complete (Theorem 8) but has exponential complexity in the worst case (see Footnote 8 and its empirical validation in Section VIII-C). Finally, we propose two local prevention/repair methods based on partial reordering (Section VI-D1) and replanning (Section VI-D2). Their practical effectiveness will be investigated in Section VIII-D, both when the two local strategies are combined, and when they are used on their own.

A. Definitions

Let $\mathcal{T}_\rho(t) = \bigcup_{i\in\mathcal{R}}\{\rho - argmin_{m_i \geq \sigma_i(t_i)}\{\langle m_i, u_j^C\rangle \in \mathcal{T}(t)\}\}$ 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 follows.

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 $\mathcal{T}_\rho$

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

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 [44] 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[7]

$$\begin{aligned}
&V_1 \in D_1 \iff V_\infty \in D_\infty\\
&E_1 \subseteq E_\infty,\ |E_1| \leq n,\ |E_\infty| \leq n(n-1).
\end{aligned}$$

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 Definition 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 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 $\Phi(w)$ 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.

B. 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

[7]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 $e_{ji} \in E_\rho$.

while addressing requirements R1–R6. In particular, a heuristic function h is *totally ordering* if: 1) for every $(i,j) \in \mathcal{R}^2$, $i \neq j$, for all $(C, C') \in \mathcal{C}_{ij}^2$ then either $i \prec_{\mathrm{h}} j$ or $j \prec_{\mathrm{h}} i$ at both C and C'; 2) $i \prec_{\mathrm{h}} j \wedge j \prec_{\mathrm{h}} k \Rightarrow i \prec_{\mathrm{h}} k$ 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 $\sigma_j(t_0 + \Delta_j^{\mathrm{stop}}) > \ell_j^C$. 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 $\sigma_k(t_0 + \Delta_k^{\mathrm{stop}}) \leq \ell_k^{C'}$. ■

Consequently, deadlocks may happen if goals are 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:

1) 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 $(i,j) \in \mathcal{R}^2$, $j > i$;
2) 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: Conditions 1 and 2 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. ■

C. Global Prevention

If, on the 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 the 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.[8] 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 $\mathcal{L}$ (which can be exponential in the worst case). In Section VIII-C, we analyze empirically the practical feasibility of the proposed approach in terms of scalability with the number of robots.

2) Global Reordering Algorithm: Let us first prove the conditions under which checking for nonlive sets and reordering 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 replanning) 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, $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0+1)T_c)$, $t_1 \in [\mathrm{k}_1 T_c, (\mathrm{k}_1+1)T_c)$), 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 behavior 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

[8] One of the best algorithms for finding all cycles in a directed 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 $O(n^2[1 + \sum_{i=1}^{n-1} \binom{n}{n-i+1}(n-i)!]) \approx O(2^{n \log n})$ by way of Stirling's approximation.

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 3 of Theorem 3 (and hence with condition 1 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 conditions 2 and 3 of Theorem 3 (or condition 1 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 $\mathcal{T}' \leftarrow (\mathcal{T}^{l-1} \setminus \{\langle m_i, u_j^C \rangle\}) \cup \{\langle m_j, u_i^C \rangle\}$ 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) *Preloading.* Theorems 8.a and 8.b are used to *preload 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 preloaded $\mathcal{T}$ is then refined according to heuristic decisions while enforcing condition 2 and 3 of Theorem 3 (or 1 of Theorem 4) as specified by the proof of Theorem 8.c (lines 22–34).

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 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-D2a); $\bigcup_{i\in\mathcal{R}} s_i(t_i)$ last communicated status (each containing $\sigma_i(t_i)$).
Params : re − order true if re-ordering is enabled; re − plan true if re-plan is enabled; stat − replan true if re-plan should be static (Section VI-D2b); 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 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 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 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 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.

D. Local Prevention and Repair

In this section, we present two local strategies for deadlock prevention and recovery, namely, partial reordering and replanning. 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 analyze 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].

1) Partial Reordering: 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 5), 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 5). The resulting time complexity is polynomial in the number

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 $(\mathcal{T}', \mathcal{T}_1', \Psi') \leftarrow (\mathcal{T}, \mathcal{T}_1, \emptyset)$;
2 $\Phi^{\text{rev}} \leftarrow$ get the reversible constraints in $\Phi(w)$;
3 foreach $\langle m_h, u_k^C\rangle \in \Phi^{\text{rev}}$ do
4 $m_k \leftarrow$ update according to (1), $s_k(t_k)$ and $s_h(t_h)$;
5 $\mathcal{T}' \leftarrow \left(\mathcal{T}' \setminus \{\langle m_h, u_k^C\rangle\}\right) \cup \{\langle m_k, u_h^C\rangle\}$;
6 $\mathcal{T}_1' \leftarrow$ update closest constraint in $\mathcal{T}'$ for h and k;
7 $\Psi' \leftarrow$ update nonlive sets in $\mathcal{T}_1'$;
8 if $|\Psi'| < |\Psi|$ then
9 return $(\mathcal{T}', \mathcal{T}_1', \Psi')$;
10 return $(\mathcal{T}, \mathcal{T}_1, \Psi)$

of robots.[9] However, deadlocks may happen (whenever all the constraints of a nonlive set in $\mathcal{T}_1$ cannot be safely reversed).

2) Replanning: 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 Theorems 2–4 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 replan 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 organized as follows. First, in Section VI-D2a, we formally state the condition under which Algorithm 1, with lines 2–4 specified according to Section V, safely supports replanning. Then, in Section VI-D2b, we investigate the efficacy of replanning in recovering from deadlocks. As in [4], both the analyses are given while assuming $\tau_{\max}^{\text{ch}} \geq 0$ but $\eta = 0$, that is, messages can be delayed but not lost (i.e., $\eta = 0$).

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

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 replanning. Also, let $\bar{q}_i(\mathrm{k}) = \boldsymbol{p}_i^{\text{old}}(\bar{\sigma}_i(\mathrm{k}))$.

[9] 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 $O(n^2)$.

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:

1) $q_i \in \bigcup_{\sigma=0}^{\bar{\sigma}_i(\mathrm{k}_0)} \boldsymbol{p}_i^{\text{old}}(\sigma) \Rightarrow q_i \in (\boldsymbol{p}_i^{\text{old}} \cap \boldsymbol{p}_i)$, i.e., the new path should overlap the previous one till the configuration corresponding to the last critical point.
2) $\bar{\sigma}_i(\mathrm{k}) \leq \bar{\sigma}_i(\mathrm{k}_0)$ for $\mathrm{k} \in \mathbb{N}$, $\mathrm{k}_0 \leq \mathrm{k} < \mathrm{k}_1$.
3) 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: 1) The condition preserves continuity. 2) 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. 3) 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 $\boldsymbol{p}_i^{\text{old}}(\ell_i^{C'}) = \boldsymbol{p}_i(\ell_i^C) \wedge (\ell_j^{C'} = \ell_j^C \vee u_j^{C'} = u_j^C)$. ■

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 $R_j(\boldsymbol{p}_j(\sigma)) \cap \mathcal{E}_i(\mathrm{k}_1) \neq \emptyset \Rightarrow \exists C \in \mathcal{C}_{ij}(\mathrm{k}_1)$ such that $\ell_j^C < \sigma < u_j^C$ for all $\sigma \in [0, 1]$. ■

2) Liveness: Theorems 3 and 4 are extended to support *any replanning* 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 replanning affects the set $\mathcal{E}$ without modifying the set $\mathcal{G}$, it does not explicitly require Θ to be locked. Condition (1), 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 (1–2) 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) Replanning to Handle Deadlocks. Relying on results of Section VI-D2a, Algorithms 4 (lines 10–23), 6, and 7 realize a decoupled replanning 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 replanning (Algorithm 4 at lines 14–17), a replanning 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 replanning to the current goal for each $i \in \mathcal{R}(w)$ are 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

Algorithm 6: The rePlan function.

Input: $\mathcal{R}(w)$ deadlocked robots.
Params : 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', C') ← update P and C according to Π(i);
8          if (E', T) is admissible then
9              (P, C) ← (P', C');
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 ≠ ∅ ;
```

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  p_i^old ← p_i, O^tmp ← O;
2  (T, Σ̄, D_1) ← get current sets and graph(1);
3  S_w ← get the weakly connected component(2) of R(w)
     in D_1;
4  foreach j ≠ i do
5      O^tmp ← O^tmp ∪ {R_j(p_j(1))};
6      if j ∈ R(S_w) then O^tmp ← O^tmp ∪ {R_j(p_j(σ̄_j))} ;
7  p_i ← planPath(p_i^old(σ̄_i), p_i^old(1), O^tmp);
8  if p_i ≠ ∅ ∧ p_i ≠ p_i^old then p_i ← ⋃_{σ=0}^{σ̄_i} p_i^old(σ) ∪ p_i ;
9  return p_i
```

(1) Allowing multi-threading, we assumed here the variable to be opportunely locked during updates. (2) 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 $j \in \mathcal{R}(w)$.

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).

Remark 1: To address conditions 1 and 2 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 2 holds, 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

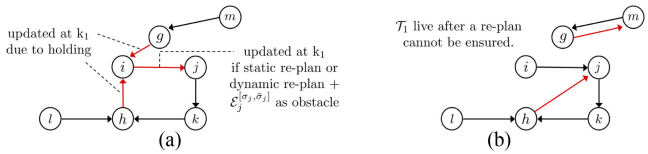

Fig. 4. Liveness of re-planning: (a) static vs. (b) dynamic.

stopping point is then removed only when the replan terminates (Algorithm 6 at line 13). This addresses condition 2. Combined with line 8 of Algorithm 7, it also addresses condition 1. Finally, condition 3 of Theorem 9 is handled in line 10 of Algorithm 6.

In the following, we analyze different design choices for Algorithms 4 and 6.

1) Parallel vs. sequential replan: 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 is proposed.

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

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

2) Static vs. dynamic replanning:

If $\boldsymbol{p}_i$ is successfully replanned, 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, $\bar{\sigma}_j(\mathrm{k}_1) \geq \bar{\sigma}_j(\mathrm{k}_1 - 1)$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 replanning), 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 replanning, 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 recommunicated. Note that dynamic replanning allows deadlocks to be anticipated, so it may reduce the probability of a robot being locally trapped.

Summarizing, 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].

VII. Livelocks

We now characterize livelocks (F3), defining the conditions under which it is possible to avoid (Section VII-A) or prevent (Section VII-B) them (see the last column of Table II).

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 replan, 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 highlights one of the main pitfalls of coordination strategies based on replanning, namely, they cannot ensure the absence of livelocks and hence liveness guarantees. Conversely, coordination strategies based on temporal refinements are livelock-free by construction if robots never backtrack along their paths.

Nevertheless, replanning 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 $\boldsymbol{p}_i^{\text{old}} \in \mathcal{P}(t_0)$, $t_0 \in [\mathrm{k}_0 T_c, (\mathrm{k}_0+1)T_c)$. Assume a new path $\boldsymbol{p}_i$ satisfying Theorems 9 and 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 4, a sufficient condition for livelock prevention is that $(\boldsymbol{p}_i^{\text{old}})^{-1}(\bar{q}_i) \leq \boldsymbol{p}_i^{-1}(\bar{q}_i)$.

Proof: Condition (1) 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 overconstrain 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 minimize the probability of livelocks occurring, it may be reasonable to exploit replanning 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.

1) Performance achievable with different heuristics are investigated in a synthetic (Test 1) and in a realistic scenario (Test 4.2).
2) An empirical validation of Theorems 6 and 7 is shown in a simulated warehouse-like scenario (Test 2).
3) The performance of our function for computing cycles in D_∞ (global reordering) and D_1 (partial reordering) is evaluated when increasing the graph density (Test 3).
4) The trade-off between complexity vs. completeness of global and local deadlock prevention/repair strategies (Algorithms 3 and 4, respectively) is investigated in a benchmark scenario with online path planning (Tests 4.1) and in a realistic application with a manually defined roadmap (Test 4.2).

Selected moments during all experiments are shown in video [43].

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. Toward this aim, 10 robots are required to perform 10 forward and 10 backward missions along paths with the same shape. Paths have been computed offline [see Fig. 5(a)] and simulations were run three times, randomizing 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 reordering (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 summarized 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

TABLE III
HEURISTICS USED IN THE TESTS

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

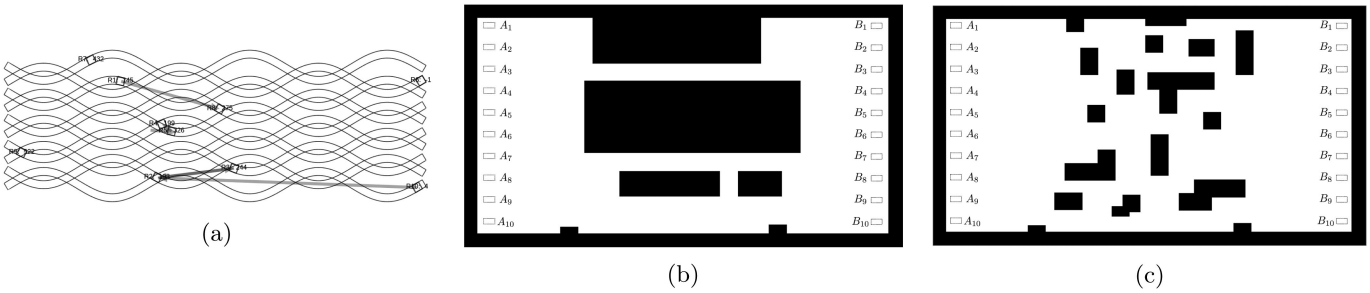

Fig. 5. (a) A snapshot of Test 1 (arrows between robots indicate precedence constraints). (b) Corridor environment used in Test 2. (c) Random environment used in Test 4.1.

TABLE IV
TEST 1: PERFORMANCE WITH DIFFERENT HEURISTICS
(DEFINED ACCORDING TO TABLE III)

$N = 4$, $\max_{\mathrm{k}}(\lvert\mathcal{C}_i(\mathrm{k})\rvert) = 16$		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: $T_c = 2\,\mathrm{s}, T_i = 0.03\,\mathrm{s}, v_i^{\max} = \pm 6\,\mathrm{m/s}, u_i^{\max} = \pm 4\,\mathrm{m/s}^2$, $\eta = 0$, $\tau_i^{\max} = \tau_i^{\min} = 0.5\,\mathrm{s}$, footprints: $0.5 \times 0.5\ \mathrm{m}^2$. N: spatial period for the sinusoidal paths. Performance has been evaluated on an Intel Core i7-6700 CPU 3.40 GHz $\times$ 8 processor, 15.6 GiB, and refers to the current implementation [47] with line 2 of Alg.1 implemented according to Section VI-D1 (partial reordering).

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 [43]).

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-C1), and in D_1 ($O(n^2)$ in the worst case; see Section VI-D1). The former is evaluated by measuring the time required by executing lines 21 and 27 in Algorithm 3 (i.e., global reordering) when incrementally adding edges to build the graph. We compare two realizations 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).

Conversely, 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 s (avg. 0.137 s, max. 0.142 s, std 0.042 s in three 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)] and 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 $T_{\mathrm{rev}}/\mathcal{C}$ 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

1) avoidance, that is, the original algorithm proposed in [4] with the FCFS heuristic (col. 2);
2) global prevention, that is, Algorithm 3 (col. 3);
3) local prevention/repair, that is, Algorithm 4 either when Algorithms 5 and 6 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 2 and 3.

TABLE V
COMPLEXITY OF COMPUTING CYCLES IN D_∞ WHILE INCREASING THE GRAPH DENSITY

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 i7-5500 U CPU @ 2.40 GHz × 4 processors, 7.7 GiB and refers to the current implementation [47] of lines 21 and 27 of Algorithm 3.

TABLE VI
TEST 4.1. PERFORMANCE WITH DIFFERENT DEADLOCK PREVENTION/RECOVERY STRATEGIES IN A BENCHMARK ENVIRONMENT

Metric		FCFS	Global re-ordering	Part. re-ordering + dyn. re-plan	Partial re-ordering	Static re-plan	Dynamic 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
$\lvert\{C \in \mathcal{C}\}\rvert$ analyzed	avg. each T_c, 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%$^{(c)}$).

Algorithm 3 and all the different versions of Algorithm 4 were run using the distance heuristic. All tests were repeated three 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.

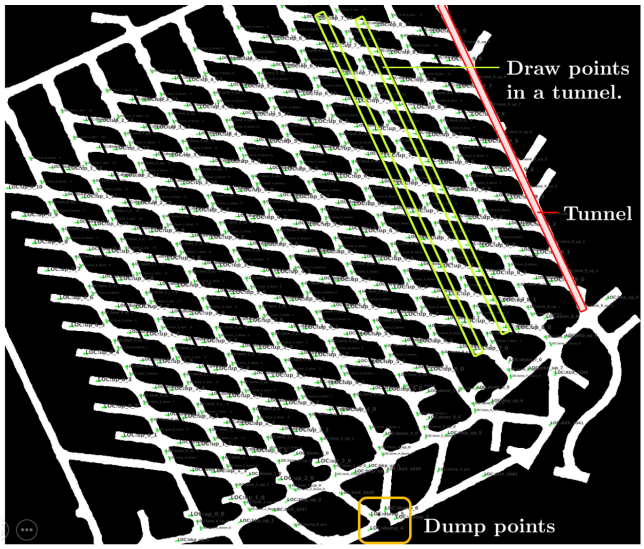

Fig. 6. Test 4: the realistic scenario.

Results are summarized as follows. 1) The FCFS heuristic ensures the absence of nonlive sets but has the worst performance in terms of time to completion. 2) Algorithm 3 ensures liveness. Also, in this scenario, its mean computational overhead is comparable with other strategies. However, $T_{\text{rev}}/T_c > 1$ may prevent 1 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. 3) Algorithm 4 using jointly partial reordering and replanning provides the best trade-off between mission time and computational overhead, and shows the practical ability to maintain liveness. Conversely, both partial reordering and replanning 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

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

TABLE VII
TEST 4.2, PERFORMANCE WITH DIFFERENT DEADLOCK PREVENTION/RECOVERY STRATEGIES IN A REALISTIC ENVIRONMENT

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%
$T_{\mathrm{rev}}/\mathcal{C}$ (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%
$\lvert\{C \in \mathcal{C}\}\rvert$ analyzed		avg., tot	201, 4399	177, 4073	246*, 306*	80, 2791	82, 2970
Nonlive sets in $\mathcal{T}_1$	Detected, solved	avg.	0, -	0, -	29*, 20*	0, -	2, 2
	$\langle m_i, u_j^C \rangle$ as $\prec_{\mathrm{h}}$	avg.	100%	11%	-	63%	-

Notes: Algorithm 4 was run while enabling only partial reorder strategy. No collision was observed.
* Simulation ended without completing all the missions. a: with respect to avg. time without coordination (avg. 55.8 s, std. dev. 103.2 s).
Parameters: $T_c = 6\mathrm{s}, T_i = 0.03\mathrm{s}, v_i^{\max} = \pm 30\mathrm{m/s}, u_i^{\max} = \pm 6\mathrm{m/s}^2, \eta = 0, \tau_i^{\mathrm{ch}} \in [0.01, 0.5]\mathrm{s}$, footprints: $11.4 \times 3.1\ \mathrm{m}^2$.
A maximum of three 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 3.40 GHz $\times$ 8 processor, 15.6 GiB, and refers to the current implementation [47].

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 four LHDs for each tunnel, which results in a maximum of 20 intersecting envelopes. The limited maneuvering capabilities of LHDs in the narrow tunnels prevents an effective use of the replanning strategy to deal with deadlocks. Therefore, Algorithm 4 was run while enabling only partial reordering. 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, 1) 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). 2) 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. 3) Partial reordering, while less computationally demanding, forfeits completeness (13% mission completion with the IDs heuristic). 4) The global reordering strategy ensures liveness. However, as in Test 4.1, $T_{\mathrm{rev}}/T_c > 1$ may prevent 1 from holding; either fewer robots, or a greater T_c may prevent this issue.

IX. ANALYSIS AND DISCUSSION

In this section, we further discuss the hypotheses tested and how they are validated by the results presented in Section VIII. We also contextualize the open issues and limitations of the proposed approaches and new avenues for future work.

Generality with respect to robot platforms, planners, and controllers. This property is inherited from [3]. Generality with respect to motion planners is validated by the use of different planners (manually designed paths in Test 1, an off-the-shelf implementation of RRT-Connect in Tests 2 and 4.1, and Bézier splines in Test 4.2). For simplicity, but without loss of generality, all tests considered homogeneous fleets of robots with car-like kinodynamics. This simplifies the real (articulated) kinodynamics of LHDs used in Test 4.2. Future work involving one of our industrial partners (Epiroc) will call for the inclusion of industrial-grade motion planners for these vehicles.10

Generality with respect to heuristics. We have formally proven that our approach ensures safety (Theorem 1) and liveness (while using Algorithm 3) with *any* heuristic. However, Tables IV (Test 1) and VII (Test 4.2) have empirically shown that different heuristics impact performance. This opens new research avenues toward integrating data-driven optimization methods (e.g., [49]) for learning effective heuristics online, with the aim of optimizing traffic flow or other application-specific objective functions (e.g., in the scenario of Test 4.2, minimizing the time to deliver the desired tonnage to the dump points10).

Provable safety with A1–A6. All tests employed a well-formed infrastructure, hence verifying A1. Also, A2–5.2 and A6 held in all tests, and injected random delays satisfied A5.3. As claimed in Theorem 1, no collision was observed. Conversely, Theorem 4, which enforces A1 in an infrastructure that is not well-formed, has not been validated empirically. Note, however, that the admissibility check called for in Theorem 4 should be considered in the context of a goal scheduling strategy, which is beyond the scope of this article. In future work, we will investigate integrated task assignment and motion planning strategies that reduce the rate of goal rejection on account of Theorem 4. The module will be inspired by the optimal approach in [50] to account for multirobot interference.

Complexity vs. Completeness of Deadlock Prevention Strategies. Tests 3 and 4.2 validated that, even if complete, the global reordering strategy is practically exploitable only when the size of the largest connected component of the precedence graph D_∞ is lower than 10. Note that the complexity of computing cycles is only partially related to the spatial distribution of the set $\mathcal{C}$, since $C \in \mathcal{C}_{ij} \Rightarrow e_{ij} \in D_\infty$. On the one hand, the analysis provides a practical strategy to design paths/assign missions to lower such complexity. On the other hand, the use of limited horizon paths (i.e., reasoning only about the minimal amount of future steps to prevent nonlive sets iteratively) or kinodynamically informed

lookaheads on precedence constraints ρ may be exploited in future work to lower the complexity of the global reordering algorithm or to improve performance of the local reordering one. In parallel, partially coupled methods such as [26] may be investigated to enhance the effectiveness of the proposed replanning strategies for deadlock prevention/repair.

X. Conclusion

We formalized a centralized algorithm for coordinating generic (possibly heterogeneous) multirobot systems subject to online asynchronous goal assignment and (possibly dynamic) precedences. The approach decoupled motion planning from coordination, which was 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 article, we focused on liveness. We 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, which may be caused by improper goal and precedence assignments, 1) we extended the concept of well-formed infrastructure given in [6] to generic robots (not necessarily disc-shaped); 2) we have formally proven that blocking never happens if the infrastructure is well-formed; 3) since this assumption imposes a tight constraint on the set of goals that can be posted, we proposed and formally validated two theorems to prevent blocking also when the assumption is relaxed. Deadlocks may be caused by inadequate priority assignments, which impose circular waits among robots. Hence, extending the characterization of deadlocks given in [3], we formally proved that 4) if goals are posted asynchronously, then static hierarchical priorities are not able to ensure liveness; 5) 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 generalized this result to any user-defined heuristic function, by proposing 6) a global and two local algorithms for deadlock prevention and recovery. We formally proved that the global algorithm ensures deadlock-free motion at the cost of exponential computation in the worst case and therefore was exploitable only when safety constraints may couple at most 10-robot motions. Receding horizon techniques may be explored in the future to further improve the efficiency of the global approach in large/more coupled fleets. The two local algorithms for deadlock prevention and recovery were based on reordering precedences and replanning paths, respectively. These were 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 showed the practical ability of these algorithms to prevent and recover from deadlocks in the tested scenarios. Finally, we characterized livelocks, formally proving that if robots are not allowed to drive back along their paths and replanning is not used (e.g., as with Algorithm 3), then livelocks will never happen. Further investigations will be devoted to including data-driven methods for learning heuristics to optimize performance and traffic management, improving the efficacy of replanning 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 Mag.*, vol. 29, no. 1, pp. 9–9, 2008.

[2] *Semantic Robots*, 2019. [Online]. Available: 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. Plan. Sched.*, 2018, pp. 485–493.

[4] A. Mannucci, L. Pallottino, and F. Pecora, "Provably safe multi-robot coordination with unreliable communication," *IEEE Robot. Automat. Lett.*, vol. 4, no. 4, pp. 3232–3239, Oct. 2019.

[5] T. Lozano-Perez, "Spatial planning: A configuration space approach," in *Auton. robot vehicles*. Berlin, Germany: Springer, 1990, pp. 259–271.

[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*, vol. 12, no. 3, pp. 835–849, Jul. 2015.

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

[8] P. Spirakis and C. K Yap, "Strong NP-hardness of moving many discs," *Inf. Process. Lett.*, vol. 19, no. 1, pp. 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. Plan. Scheduling*, 2015, pp. 324–332.

[10] S. Akella and S. Hutchinson, "Coordinating the motions of multiple robots with specified trajectories," in *Proc. IEEE Int. Conf. Robot. Aut.*, 2002, vol. 1, pp. 624–631.

[11] 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.

[12] 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.

[13] A. Coskun and J. M O'Kane, "Online plan repair in multi-robot coordination with disturbances," in *Proc. Int. Conf. Robot. Autom.*, 2019, pp. 3333–3339.

[14] Z. Yan, N. Jouandeau, and Arab Ali Cherif, "A survey and analysis of multi-robot coordination," *Int. J. Adv. Robotic Syst.*, vol. 10, no. 12, 2013, Art. no. 399.

[15] P. A O'Donnell and T. Lozano-Pérez, "Deadlock-free and collision-free coordination of two robot manipulators," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1989, vol. 89, pp. 484–489.

[16] C. Tomlin, I. Mitchell, and R. Ghosh, "Safety verification of conflict resolution manoeuvres," *IEEE Trans. Intell. Transp. Syst.*, vol. 2, no. 2, pp. 110–120, Jun. 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.*, vol. 3, no. 1, pp. 3–11, Mar. 2002.

[18] L. Pallottino, V. G Scordio, A. Bicchi, and E. Frazzoli, "Decentralized cooperative policy for conflict resolution in multivehicle systems," *IEEE Trans. Robot.*, vol. 23, no. 6, pp. 1170–1183, Dec. 2007.

[19] L. Chen and C. Englund, "Cooperative intersection management: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no.2, pp. 570–586, Feb. 2016.

[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.*, vol. 13, no. 4, pp. 1433–1447, Oct. 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.*, vol. 12,3, pp. 922–934, Jul. 2015.

[22] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robot. Autom. Mag*., vol. 4, no. 1, pp. 23–33, Mar. 1997.
[23] J. Van Den, S. J. Berg, M. Guy Lin, and D. Manocha, Reciprocal n-body collision avoidance. in *Robotics Research*. Berlin, Germany: Springer, 2011, pp. 3–19.
[24] S. M LaValle, *Planning Algorithms*. Cambridge, U.K.: 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," *Int. J. Robot. Res.*, vol. 3, no. 4, pp. 76–88, 1984.
[26] G. Wagner and H. Choset. "M*: A complete multirobot path planning algorithm with performance bounds," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 3260–3267.
[27] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1–4, 1987, Art. no. 477.
[28] J. Van DenBerg and M.Overmars, "Kinodynamic motion planning on roadmaps in dynamic environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 4253–4258.
[29] J. P Van DenBerg and M. H Overmars, "Prioritized motion planning for multiple robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2005, pp. 430–435.
[30] M. Bennewitz, W. Burgard, and S. Thrun, "Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots," *Robot. Autom. Syst*., vol. 41, no. 2–3, pp. 89–99, 2002.
[31] K. Kant and S. W Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," *Int. J. Robot. Res.*, vol. 5, no. 3, pp. 72–89, 1986.
[32] 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.
[33] 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.
[34] M. Mansouri, B. Lacerda, N. Hawes, and F. Pecora, "Multi-robot planning under uncertain travel times and safety constraints," in *Proc. 28th Int. Joint Conf. Artif. Intell.,* vol. 7, 2019, pp. 478–484.
[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*., vol. 9, no. 1, pp. 8–21, Mar.–Jun. 2017.
[36] J. Gregoire, S. Bonnabel, and A. De La Fortelle, "Priority-based intersection management with kinodynamic constraints," in *Proc. IEEE Eur. Control Conf.*, 2014, pp. 2902–2907.
[37] J. Gregoire, Priority-based coordination of mobile robots. Ph.D. thesis, 2014, *arXiv:1410.0879*.
[38] R. Ghrist, J. M O'Kane, and S. M LaValle, "Computing pareto optimal coordinations on roadmaps," *Int. J. Robot. Res.*, vol. 24, no. 11, pp. 997–1010, 2005.
[39] E. G Coffman, M. Elphick, and A. Shoshani, "System deadlocks," *ACM Comput. Surv.*, vol. 3, no. 2, pp. 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 *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 662–669.
[41] R. Ghrist and S. M Lavalle, "Nonpositive curvature and pareto optimal coordination of robots," *SIAM J. Control Optim*., vol. 45, no. 5, pp. 1697–1713, 2006.
[42] J. Gregoire, S. Bonnabel, and A. de La Fortelle, "Optimal cooperative motion planning for vehicles at intersections," *IEEE IV 2012 Workshop Navigation, Accurate Positioning Mapping Intell. Vehicles*, 2013, *arXiv:1310.7729*.
[43] A. Mannucci, L. Pallottino, and F. Pecora, *Provably safe and live multirobot coordination, Youtube*, 2020. [Online]. Available: https://youtu.be/dFwf8gkItYU
[44] M. P. Fanti and M. Zhou, "Deadlock control methods in automated manufacturing systems," *IEEE Trans. Syst., Man, Cybern. A., Syst. Humans*, vol. 34, no. 1, pp. 5–22, Jan. 2004.
[45] D. B Johnson, "Finding all the elementary circuits of a directed graph," *SIAM J. Comput.*, vol. 4, no. 1, pp. 77–84, 1975.
[46] T. P. Baker, "Stack-based scheduling of realtime processes," *Real-Time Syst.*, vol. 3, no. 1, pp. 67–99, 1991.
[47] F. Pecora, A. Mannucci, and C. S. Swaminathan, "A framework for multi-robot motion planning, coordination and control," 2020. [Online]. Available: https://github.com/FedericoPecora/coordination_oru, *version 0.6.1*.
[48] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An efficient approach to single-query path planning," in *Proc. ICRA. Millennium Conf., IEEE Int. Conf. Robot. Autom. Symposia Proc.*, 2000, vol. 2, pp. 995–1001.
[49] R. Calandra *et al.*, "Bayesian optimization for learning gaits under uncertainty," *Annals Mathematics Artificial Intell.*, Springer, vol. 76, no. 1, pp. 5–23, 2016.
[50] Nam and D. A. Shell, "Assignment algorithms for modeling resource contention in multirobot task allocation," *IEEE Trans. Autom. Sci. Eng*., vol. 12, no. 3, pp. 889–900, Jul. 2015.

Anna Mannucci received the bachelor's degree in electronic, master's degree in robotics and automation, and the Ph.D. degree in robotics and automation from the University of Pisa, Pisa, Italy, in 2013, 2016, and 2020.

She is Postdoctoral Researcher with the Multi-Robot Planning and Control Laboratory, Örebro University, Örebro, Sweden. She is one of the Principal developers and maintainers of the coordination_oru library.

Lucia Pallottino received the "Laurea" degree in mathematics and the Ph.D. degree in robotics and industrial automation from University of Pisa, in 1998 and 2002.

She is Associate Professor with the Centro di Ricerca "E. Piaggio" and the Dipartimento di Ingegneria dell'Informazione at the University of Pisa, Pisa, Italy. She is Deputy Director of Centro di Ricerca "E. Piaggio," Pisa, Italy. Her research interests include robotics, motion planning, and optimal control of multirobot systems and coordination of multirobot vehicles.

Dr. Pallottino 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).

Federico Pecora received the graduation in computer science engineering from the University of Rome "La Sapienza," Rome, Italy. He received an M.Sc. in computer science engineering from the University of Rome "La Sapienza," Rome, Italy, in 2003, where he also received his Ph.D. in computer science engineering, in 2007.

He is Associate Professor in Computer Science with Örebro University, Örebro, Sweden, where he leads the Multi-Robot Planning and Control Laboratory. His research interests include 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, multirobot coordination, and the integration of these with robot motion planning and control.