Multirobot Reconnection on Graphs: Problem, Complexity, and Algorithms

Jacopo Banfi, Nicola Basilico, *Member, IEEE,* and Francesco Amigoni, *Senior Member, IEEE*

***Abstract*—In several multirobot applications in which communication is limited, the mission could require the robots to iteratively take coordinated joint decisions on how to spread in the environment and on how to reconnect with each other to share data and compute plans. Exploration and surveillance are examples of these applications. In this paper, we consider the problem of computing robots' paths on a graph-represented environment for restoring connections at minimum traveling cost. We call it the Multirobot Reconnection Problem (MRP), we show its NP-hardness and hardness of approximation on some important classes of graphs, and we provide optimal and heuristic algorithms to solve it in practical settings. The techniques we propose are then exploited to derive a new efficient planning algorithm for a relevant connectivity-constrained multirobot planning problem addressed in the literature, the Multirobot Informative Path Planning with Periodic Connectivity problem (MIPP-PC).**

***Index Terms*—Path planning for multiple mobile robot systems, networked robots, multirobot reconnection.**

I. Introduction

Communication is a central requirement for teams of autonomous mobile robots operating in the real world. In many real situations, the availability of global communication between the robots could be a far too optimistic assumption. As a consequence, robots build a multi-hop communication network in order to share information. This task is central in a number of multirobot missions like, for example, information gathering, search, exploration, and surveillance. Such missions, regardless of the primary objective to achieve, can be thought as iteratively evolving as follows: (a) all the robots establish communication between each other (possibly in a multi-hop fashion), (b) they agree on how to spread out in order to visit some locations, (c) they follow the planned paths, and (d) they reconnect to share the collected data (possibly with a base station) and start again from (b). In the literature, it is common to find *ad hoc* solutions trying to reconcile the primary mission objectives with a particular connectivity constraint [1]–[14], like requiring continuous or periodic connectivity throughout the mission. However, an alternative approach, generally valid, could prescribe to periodically perform planning according to the following two-phase scheme. First, good paths optimizing the primary objective are computed. Then, such paths are augmented in order to efficiently restore global multi-hop connectivity at the robots' final positions (see Fig. 1).

J. Banfi and N. Basilico are with the Department of Computer Science, University of Milan, Milano, Italy (email: {jacopo.banfi,nicola.basilico}@unimi.it).

F. Amigoni is with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Milano, Italy (email: francesco.amigoni@polimi.it).

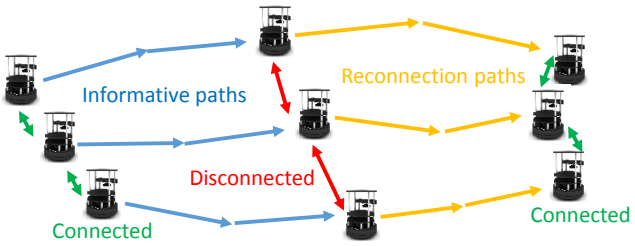


Fig. 1. The two-phase planning scheme considered in this work. First, the robots focus on the optimization of a primary mission objective (blue paths). Then, they augment the paths to re-establish global connectivity to share data and compute new plans (orange paths). Our work aims at providing effective algorithms for this latter multirobot reconnection phase. (TurtleBot 2 images are taken from http://www.turtlebot.com/turtlebot2/.)

In this paper, we propose techniques for handling the second phase of the above planning scheme. In particular, we aim at answering the following question: given a non-connected deployment of robots in a known graph-represented environment, which are the optimal robots' paths in order to reach global multi-hop connectivity? We deal with optimality in terms of the minimization of the traveling cost for robots. Specifically, the main contributions of this work are:

- a formulation of the Multirobot Reconnection Problem (Section III) and a series of complexity results proving its NP-hardness and hardness of approximation for important classes of graphs (Section IV);
- a resolution approach for the general case with an (generally inefficient) exact method and two efficient sub-optimal algorithms, as well as an exact polynomial algorithm for a particular, yet significant, class of environments (Section V);
- an evaluation campaign composed of two parts: an experimental analysis of the performance of our algorithms (Section VII) and an applicability assessment and comparison to a relevant use case previously studied in the literature [14] (Section VIII).

With this work, we shed light on the importance and challenges of optimal reconnection in multirobot scenarios. In particular, we show the advantages of achieving cost-effective reconnections in some specific problem settings. At the same time, we discuss the limits of our approach in terms of general intractability of the problem and of the consequent suboptimality of some of the proposed efficient algorithms.

II. Related Work

Reconnection constraints for teams of mobile robots received a significant attention in the recent multirobot systems

literature. The various contributions developed around this issue can be classified according to two different families of approaches. The first one assumes a planning formulation where mission objectives are jointly sought under compliance of reconnection constraints. For example, in multirobot exploration, several works concentrate on defining robots' local behaviors achieving reconnection at times dictated more or less strictly [1], [2], [4]–[7]. See [15] for a detailed survey. Other examples can be found in surveillance and search applications where team connectivity maintenance is considered as an additional mission objective. For instance, in [8], authors propose a reactive controller seeking link maintenance while considering transmission quality. In [9], the same objective is met for graph-represented environments also characterized by limited transmission ranges and interferences. Authors of [10] focus on a surveillance scenario in which a team of Unmanned Ground Vehicles (UGVs) is required to recurrently connect (under a limited-distance line-of-sight communication model) at the positions selected to sense the environment, with the aim to complete the task in the shortest possible time. Interestingly, in [11], discrete and continuous environment representations are combined and a problem decomposition is investigated. Here, team connectivity is assumed to be the primary goal and a distributed control strategy deals with the optimization of secondary mission-related objectives.

The approaches outlined above, in which mission objectives are jointly optimized with reconnection requirements, might, in some cases, over-constrain robots' movements [1], [2], [4], [5], [7], [9]–[11] potentially resulting in reductions of performance. Even relaxing the communication constraints, effective trade-offs between the satisfaction of reconnection constraints and other mission objectives might be hard to obtain [6], [8].

These drawbacks motivate the second family of approaches, where primary mission objectives are not subordinated to connectivity constraints and reconnection can be dealt with in a separated planning phase. In other words, robots first compute a plan for the primary mission objective without considering any connectivity requirement. Second, a *reconnection strategy* is exploited to regain connectivity if needed. These approaches are less popular and formally established, although the literature includes some solutions of this kind. In [16], robots are divided between those pursuing a mission and those serving as communication relays. Robots belonging to the first group plan without considering connectivity, which is handled by the second group of robots. These robots move to form a connected topology with robots from the first group. In [17] and [18], robots disconnect while exploring the environment and subsequently reconnect by undertaking a rendezvous. In [12], [13], a flexible method based on intermittent communication is proposed. This solution allows disconnections when the primary mission objectives are sought and guarantees that robots will reconnect infinitely often at some points in time. Plans satisfying this kind of intermittent connectivity are obtained by adopting a formalization based on Linear Temporal Logic. Robots can reconnect in subgroups, without the need to form a fully connected network. Differently from our work, robots are assumed to be able to communicate only when simultaneously present at the same physical location.

Specifically-tailored solutions aside, this second family of approaches currently lacks a general characterization of the multirobot reconnection problem. Our work contributes to this end by formalizing it, studying its resolution to optimality, and devising an application-independent resolution methodology that can be employed in realistic scenarios. We also deploy our methods into the multirobot setting proposed in [14]. This problem is called Multirobot Informative Path Planning with Periodic Connectivity (MIPP-PC) and is both highly relevant to our work, being appropriate for resolution with our methods, and well-representative of the literature background, since it can be tackled with both the two families of approaches presented above. In MIPP-PC, robots plan joint paths trying to maximize some reward function over a given discrete time horizon T and being subject to the requirement of regaining global connectivity each T_I time steps (see Section VIII for a detailed description). We show that, for large values of T_I, solutions belonging to the second family can be conveniently applied and we exploit our methods to obtain good mission performance. In doing this, we also relax the common assumption of error-free communication links (see, for example, [4], [5], [14]), providing reconnection strategies capable of handling communication errors and uncertainties.

III. Problem Formulation

We start by providing the formal specification of the multirobot setting we consider in this work. The environment is modeled with a multigraph $G = (V, E, C)$ describing its physical and communication features. Locations are modeled by a set of vertices V that can be obtained with discretization techniques (as, for example, those in [16], [19]). The first edge set, E, encodes the physical topology which is assumed to be undirected and connected. Each physical edge is associated with a positive rational number denoting the corresponding traveling cost, and we denote with $d(u, v) \in \mathbb{Q}_0^+$ the minimum cost for traveling between any two vertices $u, v \in V$ (staying still at the same vertex has null cost). Since we assume that our multigraph represents a physical environment, we impose that the generalized triangle inequality holds, namely that the cost $d(u, v)$ associated with a physical edge $(u, v) \in E$ is not greater than that of any other (u, v)-path computed on E. The second set of edges, C, encodes the communication topology, namely, the availability of a communication link between a pair of locations. Communication links in C can be pre-computed by means of a *link-detection mechanism* which, given $u, v \in V$, determines whether $(u, v) \in C$. Link-detection mechanisms can range from conservative visibility-based criteria [16] to more sophisticated approaches where estimates of the signal strengths can be considered in building a connectivity graph [20]. For now, we make the assumption (as done in [14] and [16]) that C is static (not changing over time) and not affected by false positives (in Section VI we relax this assumption). We also assume that no bandwidth constraint is imposed on any link $(u, v) \in C$. Throughout the paper, we will use the following additional notation to ease presentation: $G_E = (V, E)$ and $G_C = (V, C)$.

A team of m mobile robots $R = \{1, \ldots, m\}$ moves on G by traversing its physical edges. For the sake of simplicity, we assume that the robots have homogeneous locomotion capabilities, allowing us to work with a unique set of traveling costs (generalizations can be easily defined). This means that the traveling costs $d(\cdot, \cdot)$ introduced above can be interpreted either as distances or times under the assumption of an ideal plan execution phase. In Section VI we will discuss situations where, due to errors, this assumption does not hold and we will propose a method to handle them. A *configuration* specifying the location of each robot is denoted as $p = \langle p_1, p_2, \ldots, p_m \rangle$ with $p_i \in V$. A configuration is said to be *connected* iff the subgraph of G_C induced by the occupied vertices is connected. The rationale is that we consider the team to be connected when any two robots can exchange data via multi-hop relays operated by teammates over the communication links in C. Note that, according to this definition, two or more robots are assumed to be able to simultaneously occupy the same vertex. Collisions are managed by exploiting a low-level collision avoidance mechanism (as done in [14]).

Against this background, we consider a data-gathering multirobot mission that unfolds as a repeated sequence of steps where reconnection of robots is achieved as an independent problem solved in a dedicated planning stage. In particular, we assume that the mission evolves as follows:

(a) robots start the mission from a connected configuration;
(b) from the current (connected) configuration robots jointly compute:
  (b1) informative paths along which data are gathered, terminating in a configuration s within a pre-established deadline;
  (b2) a connected configuration p to reach starting from s after the above deadline has expired;
(c) robots follow their path and reach s (potentially losing connection);
(d) robots reach p, establish connection, then continue from (b).

The above scheme may apply to a range of applications broader than data-gathering ones. In fact, the only binding choice we make is to separate reconnection from mission primary objectives with Steps (b1) and (b2).

Step (d) entails a connection establishment protocol, allowing each robot to be aware that the whole-team reconnection took place successfully. Many different types of protocols can be executed at the connected configuration p, like distributed robot discovery or methods based on leader election as proposed in [21]. The details of this protocol deeply depend on the particular application and, as such, lie outside the scope of this work. What is important for our purposes is that, using these protocols, robots can detect that they are connected.

We focus on the problem entailed by Step (b2) under the basic assumption that *all* the robots must reconnect and that *any* connected configuration is a feasible solution. Both our model and our resolution techniques can be extended to capture additional constraints posed by particular applications. For instance, a reconnection task (instance of Step (b2)) can be performed for a subset of robots, while candidate vertices or configurations where to reconnect can be filtered with more specific feasibility rules. Nevertheless, all of these or similar variants require to compute an optimal connected configuration for some robots. In this work, we tackle the basic core version of this problem, also showing, in Section VIII, its significance in a use case. We leave for future works the study of refinements for cases where additional specific constraints are added.

Computing the optimal connected configuration p given that robots are in configuration s poses the need of selecting an optimality criterion. We decide to focus on the simple, yet relevant for mobile robots, minimization of the traveling cost. This is an optimality criterion widely used in applications like, notably, exploration [22]. Formally, when jointly moving from a configuration s to another configuration p, a robot r spends the traveling cost $d(s_r, p_r)$. To minimize the reconnection time and seek, at the same time, small waiting times for robots, we adopt as objective function the maximum traveled cost defined as $B(s, p) = \max_{r \in R} d(s_r, p_r)$. We also call *bottleneck robot* the robot inducing this maximum value.

Now, let $\mathcal{C}$ be the set of all joint connected configurations for a graph G_C. The *Multirobot Reconnection Problem* (MRP) can be stated as follows: given G and a non-connected starting configuration $s \notin \mathcal{C}$, compute $p^* = \arg\min_{p \in \mathcal{C}} B(s, p)$. That is, assign each robot to a new vertex such that reconnection between robots is established at the minimum traveling cost.

IV. Problem Analysis

In this section, we formally analyze the complexity of the MRP by providing results that will be functional in designing exact and approximate solving methods. As it might be expected, solving the MRP is hard both when searching for an optimal solution or when seeking for provably bounded approximations. However, besides its general intractability, the problem exhibits significant special cases for which optimal efficient algorithms can be given.

To prove the NP-hardness results, it is useful to consider the *decision version* of the MRP, called MRP-k. Here, we are given a multigraph $G = (V, E, C)$, a distance function $d(\cdot, \cdot)$ defined on E, a starting configuration s, and a value $k \in \mathbb{Q}^+$. The objective is to decide whether a connected configuration p such that $B(s, p) \leq k$ exists. The NP-hardness of the MRP (which is an optimization problem) directly follows from the NP-completeness of its decision counterpart [23].

We start by considering environments whose physical representation G_E is a tree, which interestingly exhibit different complexities depending on the topology of G_C.

Theorem 1. *The MRP is NP-hard even when G_E is a tree and $C \supset E$.*

Proof. It is easy to see that NP membership of the MRP-k holds: given a candidate solution stated in terms of robot-vertices occupations, one can easily verify in polynomial time that such configuration induces a connected subgraph on G_C and that the bottleneck robot does not travel a distance larger than k. NP-hardness can instead be shown by constructing a particular instance of the MRP-k from a generic instance of the

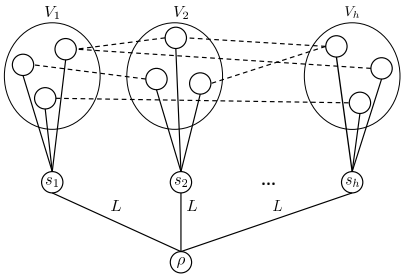


Fig. 2. Reduction from the Connected Coverage Problem: E is given by solid lines, C by solid and dashed lines.

Connected Coverage Problem (CCP) [24] which is defined as follows. Given a graph $G^{Cov} = (V^{Cov}, E^{Cov})$ and a partition of V^{Cov} denoted as $(V_1, \ldots, V_h)$ where $(v, q) \in E^{Cov}$ only if $v \in V_i, q \in V_j, i \neq j$, determine if there exist h vertices $\langle p_1, \ldots, p_h \rangle, p_i \in V_i$, inducing a connected subgraph in G^{Cov}. From any instance of CCP, we show how to build a corresponding instance of the MRP-k by setting $k = 1$ and giving a current configuration for h robots $s = \langle s_1, s_2, \ldots, s_h \rangle$ on a multigraph G, where (see Fig. 2) $V = V^{Cov} \cup \{\rho, s_1, \ldots, s_h\}$, $E = \{(\rho, s_i), (s_i, v)\}$ for all V_i and $v \in V_i$, traveling costs are unitary except for $d(\rho, s_i) = L = 2$, and $C = E \cup E^{Cov}$.

Basically, the upper part of our particular construction (the one included in the V_i sets) replicates the original CCP instance. The idea of the proof is to enforce the robots starting from s_i vertices, one for each V_i set, to reconnect in a single step by moving in a vertex belonging to the corresponding V_i if and only if the CCP instance admits a *yes* answer. More formally, if the CCP instance has a *yes* answer, then there always exists a connected configuration $p = \langle p_1, \ldots, p_h \rangle$ where $p_i \in V_i$ such that, by construction, $B(s, p) = 1$. Conversely, if the MRP-k instance has a *yes* answer, then the corresponding connected configuration has a vertex in each V_i: indeed, any other connected configuration violating this structure would necessarily require a bottleneck robot to travel at least a distance of 2 to reconnect in ρ. □

The above result can be easily extended to the more specific case of trees with unitary costs. However, when considering the complementary inclusion condition on C we obtain a different result.

Theorem 2. *The MRP-k is in P when G_E is a tree and $C \subseteq E$.*

We will prove this in Section V-C by providing an optimal polynomial-time algorithm for an important class of multigraphs in which G_E is a tree and having $C = E$ as a particular case. The algorithm can be easily extended to obtain an optimal algorithm for trees with $C \subset E$.

When removing the assumption that the physical topology is a tree, an additional more specific hardness result can be derived. Notice that the subset of instances defined in the following result does not contain those defined in Theorem 1 nor is strictly contained in them.

Theorem 3. *The MRP is NP-hard even on multigraphs where costs are unitary, both G_E and G_C have vertices with degree ≤ 4, and $C = E$, $C \subset E$, or $C \supset E$.*

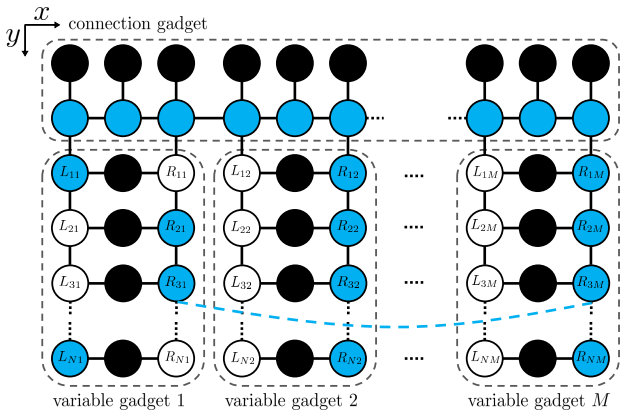


Fig. 3. Reduction from 3-SAT: gadgets on the $z = 0$ plane. Black vertices represent robots' starting positions. Blue items (vertices and dashed line) represent the contradictory argument used in the proof.

Proof. 3-SAT can be reduced to the MRP-k with vertex degree ≤ 4, unitary costs, and $C = E$. An instance of 3-SAT is specified by M variables and N clauses. A clause is the disjunction of 3 literals, where, if v is a Boolean variable, then v and $\overline{v}$ are positive and negative (negated) literals, respectively. The instance has a *yes* answer iff there exists a truth assignment for the M variables which satisfies all clauses. We construct an MRP-k instance on a three-dimensional xyz layout where, to ease visualization, we assume without loss of generality that each vertex lies at integer non-negative coordinates. More precisely, we combine three planar graphs, each lying on an xy plane for $z = 0, 1, 2$, respectively.

Let us first describe the graph lying on the xy plane with $z = 0$ whose top view is reported in Fig. 3. Black vertices denote the presence of a robot, thus partially defining the set R and the starting configuration s. Since $C = E$, each edge represents both a physical connection and a communication link. By inspecting the figure, we identify subgraphs that we call *gadgets*. A *variable gadget* v has 3 columns and N rows, one for each clause. Each row c is composed of three vertices, where the middle one is occupied by a robot and the left and right ones are empty and denoted as L_{cv} and R_{cv}, respectively. Subscript c indicates the row (clause) index while v refers to the variable associated with the gadget. The *connection gadget* lies above the variable gadgets at $y = 0, 1$ and is connected to the first row of L and R vertices. It is composed of a row of $3M$ terminal occupied vertices (at $y = 0$) and is connected to another row of $3M$ empty vertices (at $y = 1$) as shown in the figure.

The construction of the planar graphs at $z = 1$ and $z = 2$ makes use of *clause gadgets*. Consider a generic xz slice where $y \geq 2$ (each slice is constructed independently) and refer to Fig. 4 for visualization. We place a line of vertices entirely occupied by robots at $z = 2$; this line must be the shortest one having, for every variable v appearing in clause c, a one-hop path to the c-th row of the underlying variable gadget v. Such paths are composed of three intermediate vertices lying at $z = 1$ dubbed *clause connection vertices*: we denote them as C_c^h, C_c^k, and C_c^w, where h, k, and w are the

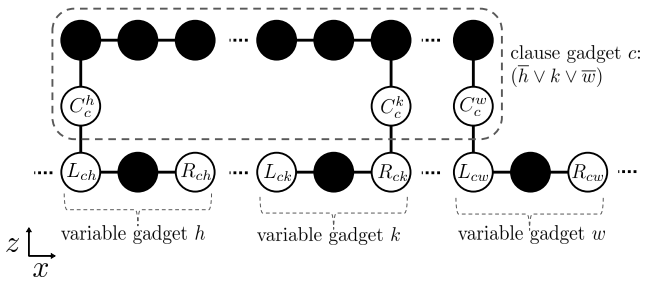


Fig. 4. Reduction from 3-SAT: slice of the xz plane with a clause gadget. Black vertices represent robots' starting positions.

variables appearing in clause c. Vertex C_c^v is connected to L_{cv} if the corresponding literal in clause c is negative (i.e., $\overline{v}$) and to R_{cv} if it is positive (i.e., v). Fig. 4 shows this construction for a clause $c = \overline{h} \vee k \vee \overline{w}$.

In the above construction, each clause gadget requires at most $3M + 3$ vertices and $3M$ robots, while the planar graph at $z = 0$ requires exactly $3MN + 6M$ vertices and $MN + 3M$ robots: therefore, this construction can be completed in polynomial time w.r.t. M and N.

We now show that the 3-SAT instance has answer *yes* iff the particular MRP-k instance admits a *yes* answer for $k = 1$. The idea underlying the above construction is to build a "mechanism" of moving robots such that all the 3-SAT clauses can be simultaneously satisfied by a truth value variable assignment iff all the robots belonging to the same variable gadget can align on the same side to replicate such a assignment (while being connected to all the other robots).

From 3-SAT to MRP-k. If the 3-SAT instance has answer *yes*, robots can reconnect by traversing each a single edge as follows (recall that $C = E$). Robots belonging to the connection gadget move down to occupy the line of vertices at $y = 1$, while in every variable gadget v either all robots move to the left (occupying vertices $L_{1v} \dots L_{Nv}$) or all robots move to the right (occupying $R_{1v} \dots R_{Nv}$) according to the corresponding literal value in the satisfied 3-SAT instance. For robots on a clause gadget, notice that the robot placed in a neighbor of *any* clause connection vertex C_c^v whose corresponding variable v is associated with the satisfaction of the clause can "trickle" one step down to $z = 1$, connecting with all the robots of the variable gadget v at $z = 0$. The remaining robots of the clause gadget can rearrange themselves to connect with such a robot by traversing at most a single edge.

From MRP-k to 3-SAT. Now we show that if the MRP-k instance has a *yes* answer for some configuration p then the 3-SAT instance can be satisfied. It is easy to notice that, in such p, robots on the connection gadget must necessarily move down to $y = 1$. The key point of the proof is to show that all the robots belonging to the same variable gadget v must necessarily align on the same side, thus producing a coherent assignment for v. By contradiction, suppose that in p at least two robots occupy different sides of the same variable gadget v. Consider now the first "portion" of robots not connected to the robots of the connection gadget (which have moved down) through robots belonging to the same variable gadget v with a lower y coordinate. The blue vertices of Fig. 3 show such a possible scenario for variable gadget 1: the second robot goes to R_{21}, and is not connected to the robots of the connection gadget since the first robot is in L_{11}. To form a globally connected configuration, there must necessarily exist, in such a portion of robots, a robot r connected to a robot r' belonging to another variable gadget v' with the same y coordinate $\hat{c}$ through the corresponding clause gadget. In the example of Fig. 3, this happens for the third robots of variable gadgets 1 and M (which are at vertices R_{31} and R_{3M}, respectively). In order for r and r' to be connected, it must be the case that the clause gadget lying at $y = \hat{c}$ contains a chain of connected robots in which two of them have trickled one step down to reach their clause connection vertices. (In our example, this is represented by the dotted blue line.) However, our construction ensures that, whenever two (or three) robots belonging to the same clause gadget trickle one step down, the chain of robots at the top of the clause gadget (which are initially connected in the starting configuration s) must necessarily break into two (or three) parts, thus implying that r and r' (and hence p) can never be connected. Therefore, all the robots belonging to the same variable gadget must align on the same side in order to have connectivity, allowing as a consequence to retrieve a coherent truth value variable assignment. Moreover, since p is connected, all the robots belonging to clause gadgets are also connected to the robots at $z = 0$ thanks to robots that have trickled one step down to occupy clause connection vertices, implying that all the 3-SAT clauses can be simultaneously satisfied. (If all the robots at $z = 0$ are connected when not considering the $z = 1, 2$ planes, it is perfectly fine for the original connected chains of robots of the clause gadgets to break in two or three parts, provided that the same clause is satisfied by more than one variable.) The proofs for cases $C \subset E$ and $C \supset E$ immediately follow from the deletion/addition of a communication edge from an arbitrary vertex lying at $y = 0$ in the communication gadget (while respecting the grid layout). □

Note that, in fact, the above reduction proves also the NP-hardness of the MRP instances in which G_E and/or G_C are restricted to be 3D-grid graphs with some "holes", a class of graphs often used as a convenient environment representation for multirobot systems operating in three-dimensional settings (see, for example, techniques used for path planning for autonomous UAVs over cubic grids discretizations [25]).

Finally, we show that an efficient algorithm for the MRP providing a good approximation factor is unlikely to exist.

Theorem 4. *Let $n = (|V|, |E|, |C|, m) \in \mathbb{N}^4$. For any polynomial-time computable function $\alpha(n)$, there does not exist an $\alpha(n)$-approximation algorithm*[1] *for the MRP, unless P=NP, even when:*

(a) *G_E is a tree and $C \supset E$, or*
(b) *G_E is an arbitrary graph and $C \subset E$ or $C = E$.*

Proof. Consider case (a) and suppose we have an $\alpha(n)$-approximation algorithm for such MRP instances. Then, it would be possible to solve any instance of CCP in polynomial

[1] An $\alpha(n)$-approximation algorithm guarantees in polynomial time a solution within a factor of $\alpha(n)$ of the optimal value [23].

time by using the reduction of the proof of Theorem 1 (as in Fig. 2) with $L = 1+\alpha(n)$. A *yes* answer for the CCP instance implies a reconnection minimum cost of 1, while a *no* answer holds iff such minimum cost is $1+\alpha(n)$. Therefore, the $\alpha(n)$-approximation applied to the MRP instance obtained from a *yes* instance of CCP would return a solution with cost at most $\alpha(n)$, while when having a *no* instance would return a cost of at least $1+\alpha(n)$. For case (b), we can use the same argument and construction of case (a) where E is augmented with all the communication links between any two vertices in V_i and V_j, all with distance $L-1$. In particular, this construction proves the case $C = E$, while $C \subset E$ follows from the removal of a single communication edge between ρ and any s_i. □

A direct consequence of Theorem 4 is that the MRP does not belong to the APX complexity class (the class of problems admitting a constant-factor approximation algorithm), even in the special cases (a) and (b).

V. Algorithms

In this section we tackle the resolution of the MRP. Specifically, in Section V-A we present an exact (but inefficient on large instances) method suitable for the general case, in Section V-B we devise efficient suboptimal approaches, and in Section V-C we provide an exact and efficient algorithm for a specific yet significant case.

A. Exact method

As the results of the previous section establish, efficient exact algorithms for the general case are out of reach. Nevertheless, the Mixed Integer Linear Programming (MILP)-based approach we propose in this section shows relatively good performance with problem instances of moderate size by building around the concept of *node separators*.

Definition 1. *Given a multigraph G, for any two vertices $u, v \in V$, a subset of vertices $N \subseteq V \setminus \{u, v\}$ is called a (u,v) separator if and only if after removing N from V (and all the incident edges from C) no path connects u and v in G_C.*

Let $\mathcal{N}(u, v)$ denote the set of all the (u, v) separators for a generic multigraph G, and let $\mathcal{N}_v = \cup_{u \neq v} \mathcal{N}(u, v)$ be the set of all separators with respect to v. Finally, for any vertex $v \in V$ and separator $N \in \mathcal{N}_v$, let $W_{N,v}$ be the set of vertices reachable from v in $G_C - N$. We encode the optimal configuration p^* with a set of binary variables z_{rv}, taking value 1 iff robot r is in vertex v. The occupied vertices in p^* induce a connected subgraph of G_C where we flag one vertex as "root". We model this with additional binary variables: y_v, taking value 1 iff vertex v is occupied, and x_v, taking value 1 iff vertex v is the root. A continuous variable w denotes the solution cost. The formulation reads as follows:

$$\min w \quad \text{s.t.} \tag{1}$$

$$\sum_{v \in V} z_{rv} = 1 \qquad \forall r \in R \tag{2}$$

$$w \geq \sum_{v \in V} d(s_r, v) z_{rv} \qquad \forall r \in R \tag{3}$$

$$\sum_{r \in R} z_{rv} \geq y_v \qquad \forall v \in V \tag{4}$$

$$y_v \geq z_{rv} \qquad \forall v \in V, \forall r \in R \tag{5}$$

$$\sum_{v \in V} x_v = 1 \tag{6}$$

$$x_v \leq y_v \qquad \forall v \in V \tag{7}$$

$$\sum_{u \in N} y_u + \sum_{u \in W_{N,v}} x_u \geq y_v \qquad \forall v \in V, N \in \mathcal{N}_v \tag{8}$$

Constraints (2) ensure that each robot is placed in one vertex. Constraints (3) and the objective function (1) set w as the maximum distance traveled by any robot. Constraints (4) and (5) ensure that $y_v = 1$ iff at least one robot occupies v. Constraint (6) chooses exactly one root and Constraints (7) enforce it to be occupied. Constraints (8) enforce the subgraph induced by the occupied vertices to be connected. To understand these last constraints, consider any vertex v that is part of the solution, i.e., with $y_v = 1$. If the solution encodes a connected configuration, then v must be connected to the root. Any separator $N \in \mathcal{N}_v$ disconnects v from one or more vertices while leaving it connected to those in $W_{N,v}$. Thus, either the root is in $W_{N,v}$ or it must be reachable through a vertex in N. So, in this last case, N must include at least one occupied vertex of the solution. (For $y_v = 0$ the constraints are ineffective.) Note that Constraints (8) are exponentially many. A typical strategy to handle them prescribes to initially feed the MILP solver without such constraints, and to gradually introduce them into the model after having detected their violation. In our case, violations of Constraints (8) can be checked by means of the polynomial-time procedure shown in [26]. Finally, define an ordering on the vertices in V, and let M_v^{m-1} be the set of vertices that are within $m-1$ hops from v in G_C (recall that m is the number of robots). To speed-up the model resolution, we can further impose that:

$$x_u + y_v \leq 1 \qquad \forall v, u \in V : v < u \tag{9}$$

$$\sum_{u \in M_v^{m-1}} x_u \geq y_v \qquad \forall v \in V \tag{10}$$

Constraints (9) break symmetries between equivalent solutions with different roots, while Constraints (10) enforce any vertex part of the solution to induce the choice of the root within the $(m-1)$-hops neighbors. Notice that, if any feasible solution $\hat{p}$ is available, the number of variables can be significantly reduced by instantiating z_{rv} only if $d(s_r, v) < B(s, \hat{p})$ (in this case, if the resulting model is unfeasible, then $\hat{p} = p^*$) or only if $d(s_r, v) \leq B(s, \hat{p})$ (in this case, the solver can be provided with an initial feasible solution).

Such a MILP-based resolution approach could be the preferred choice for those settings where the MRP can be solved

only once and offline, like in periodic information gathering missions. Nevertheless, the unavoidable scaling limits pose the need for heuristic efficient methods to tackle the MRP in the general case.

B. Heuristic methods

Let us consider a subset $P \subseteq V$ with $|P| \leq m$. If the subgraph of G_C induced by the vertices in P is connected, then we say, for short, that P is connected. Given any connected P and m robots in configuration s, the configuration p occupying vertices P and minimizing $B(s,p)$ can be efficiently computed by formulating and solving a Linear Bottleneck Assignment Problem (LBAP). In general, this problem seeks the best agent-task assignment minimizing the maximum assignment cost. Customarily, situations with more agents than tasks are dealt with by adding dummy null-cost tasks. Clearly, in our case, agents correspond to robots, tasks to vertices of P, and the assignment costs are given by the shortest traveling distances from s. To date, the best known algorithm for this problem runs in $O(m^{2.5}/\sqrt{\log m})$ [27]. We now propose three suboptimal methods leveraging LBAPs and restricting our attention to subsets where $|P| = m$. Such a restriction is safe under the mild assumptions we summarize in the following proposition (the proof is trivial).

Proposition 1. *Given an MRP instance, if:*

(A1) $s_r \neq s_{r'}, \forall r \neq r' \in R$*, i.e., no vertex is occupied by more than one robot in the starting configuration;*

(A2) *for any $u, v \in V$, there exists one shortest path between them on G_E such that if the physical edge $(i,j) \in E$ is traversed along the path then $(i,j) \in C$ too.*

then, for any solution p occupying less than m vertices there exists another one p' occupying exactly m vertices and such that $B(s,p') \leq B(s,p)$.

These two assumptions are often satisfied in reality. Indeed, different robots can be always assumed to occupy different locations for suitable environment discretizations as required in (A1), while (A2) should hold in any reasonable communication-aware environment discretization.

The first algorithm we present, which will serve mainly as a baseline against which comparing the other two algorithms, is based on the following simple observation: if we are able to correctly "guess" the right connected P, then solving a LBAP to assign the m robots to the vertices in P will give us the optimal solution. The *random-reconnection* heuristic tries to build a good solution by randomly choosing a connected P (with $|P| = m$ in virtue of Proposition 1) starting from each possible initial vertex $\hat{v} \in V$, upon which solving a LBAP for the m robots. For a given P, we define its connection neighborhood as $N(P) = \{v \in V \setminus P \mid \exists u \in P, (u,v) \in C\}$. Algorithm 1 shows the pseudo-code for computing a solution starting from a given initial vertex $\hat{v}$.

The final output of random-reconnection is the connected configuration with minimum cost among all the ones constructed starting from each $\hat{v} \in V$. The running time of this heuristic is $O\big(|V|(mT_{\text{NEIGH}} + T_{\text{LBAP}})\big)$, where T_{NEIGH} denotes the cost of updating the neighborhood $N(P)$ – this can be done in just $O(|V|)$ – and T_{LBAP} denotes the cost of the selected algorithm for solving a LBAP on m robots.

Algorithm 1 Random-reconnection from fixed $\hat{v}$

1: $P = \{\hat{v}\}$
2: **repeat**
3: $\quad \bar{v} = \text{random}(N(P))$
4: $\quad P = P \cup \{\bar{v}\}$
5: **until** $|P| = m$
6: **return** $p = \text{LBAP}(R, P)$

Clearly, the random-reconnection heuristic could benefit from trying to randomly build multiple connected configurations for each possible starting vertex $\hat{v}$ (by setting a number of restarts). However, this would come at the expense of an increased running time, which could not be affordable in some real-time settings. Moreover, since the number of connected configurations of size m containing a given vertex $\hat{v}$ increases exponentially with m, being able to randomly find the best connected configuration would still be highly unlikely. This poses the need of more informed reconnection heuristics. To this aim, we now present two complementary algorithms.

We denote the first informed reconnection algorithm as *forward-reconnection*. From the scheme introduced above, the algorithm inherits the restriction to subsets of vertices P such that $|P| = m$ (by Proposition 1), as well as the attempt to construct a good connected configuration for each possible starting vertex $\hat{v} \in V$. However, differently from random-reconnection, forward-reconnection does not necessarily need, in principle, the final computation of a LBAP, since the allocation of robots to vertices can be saved along the construction of P. In practice, however, the final computation of a LBAP on the vertices underlying the resulting connected configuration (which is not computationally expensive for typical team sizes) guarantees to return the best robot-vertices allocation for such vertices. The formal steps of forward-reconnection are reported in Algorithm 2, where U denotes the set of currently unassigned robots. Steps 3–4 select the next vertex to include as the one closest to an unassigned robot, while Step 6 assures the minimization of $B()$ on the constructed P.

Algorithm 2 Forward-reconnection from fixed $\hat{v}$

1: $U = R \setminus \{\arg\min_{r \in U} d(s_r, \hat{v})\}$, $P = \{\hat{v}\}$
2: **repeat**
3: $\quad (\bar{r}, \bar{v}) = \arg\min_{r \in U, v \in N(P)} d(s_r, v)$
4: $\quad P = P \cup \{\bar{v}\}$, $U = U \setminus \{\bar{r}\}$
5: **until** $|P| = m$
6: **return** $p = \text{LBAP}(R, P)$

Algorithm 2 is run for each possible starting vertex $\hat{v}$, and we then select the configuration with minimum cost. The running time of this heuristic can be expressed as $O\big(|V|(m^2|V| + mT_{\text{NEIGH}} + T_{\text{LBAP}})\big) = O\big(|V|(m^2|V| + T_{\text{LBAP}})\big)$.

The second informed heuristic we propose is called *backward-reconnection* and operates under a dual scheme w.r.t. forward-reconnection. For each possible starting vertex $\hat{v}$, we construct an initial configuration p where each robot occupies $\hat{v}$. The idea is to obtain a connected P of at most m vertices by iteratively relocating robots from the current configuration p to

a new configuration p' in such a way that $B(s,p)-B(s,p')$ is maximized. Algorithm 3 formalizes the m steps (initialization plus $m-1$ iterations) required to build P. At any iteration, $\hat{R}(v)$ is equal to the empty set if, in the current configuration p, v is occupied by one or no robots, otherwise it includes the (two or more) robots occupying v. Then, let $\hat{R}=\cup_{v\in P}\hat{R}(v)$ be the robots temporarily sharing their vertices with at least another robot.

Algorithm 3 Backward-reconnection from fixed $\hat{v}$

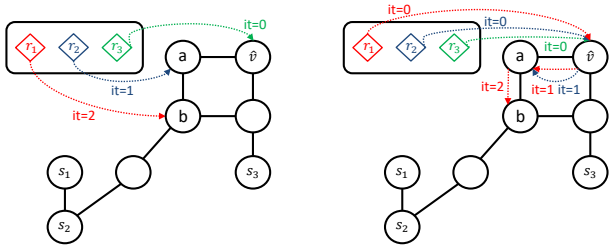
1: $P=\{\hat{v}\}$, $p=\langle\hat{v},\dots,\hat{v}\rangle$
2: $P^*=P$, $p^*=p$
3: **for** $i=1,\dots,m-1$ **do**
4: $\quad\bar{r}=\arg\max_{r\in R} d(s_r,p_r)$, $\bar{v}\leftarrow p_{\bar{r}}$
5: $\quad\bar{w}=\arg\min_{w\in P\cup N(P)} d(w,s_{\bar{r}})$
6: $\quad P=P\cup\{\bar{w}\}$, $p_{\bar{r}}\leftarrow\bar{w}$
7: $\quad$**if** $\bar{v}$ is not occupied **then**
8: $\quad\quad r'=\arg\min_{r\in\hat{R}\setminus\{\bar{r}\}} d(s_{\bar{r}},\bar{v})$, $p_{r'}\leftarrow\bar{v}$
9: $\quad$**for each** $v\in P\setminus\{\bar{w}\}$ **do**
10: $\quad\quad Q=\hat{R}(v)$
11: $\quad\quad$**for each** $r\in Q$ **do**
12: $\quad\quad\quad$**if** $d(s_r,\bar{w})<d(s_r,v)$ **then**
13: $\quad\quad\quad\quad p_r\leftarrow\bar{w}$
14: $\quad\quad$**if** v is not occupied **then**
15: $\quad\quad\quad r'=\arg\min_{r\in Q} d(s_r,v)$, $p_{r'}\leftarrow v$
16: $\quad$**if** $B(s,p)<B(s,p^*)$ **then**
17: $\quad\quad P^*=P$, $p^*=p$
18: **return** $p=\text{LBAP}(R,P^*)$

After the initialization, the algorithm computes in Step 4 the bottleneck robot $\bar{r}$ for p and its current location $\bar{v}$. In Steps 5 and 6 this robot is relocated to the vertex $\bar{w}$ which is the closest to $s_{\bar{r}}$ (that robot's starting vertex) among those in $P\cup N(P)$, that is the vertices currently offering a communication link to the team. (Contrarily to forward-reconnection, we consider also the vertices in P since they could actually include the starting position of the bottleneck robot.) Steps 7 and 8 handle the fact that the previous relocation could have left $\bar{v}$ unoccupied. In such a case, a robot $r'\in\hat{R}$ is assigned to $\bar{v}$; again r' is chosen as the one whose starting vertex $s_{r'}$ is closest to $\bar{v}$. The same criterion is adopted in Steps 9–15 to evaluate if the newly included vertex $\bar{w}$ enables a convenient relocation of any robot in $\hat{R}$. Precisely, in Step 12 the algorithm checks if any robot r (sharing its vertex with at least another one) can strictly reduce its traveling distance if relocated to $\bar{w}$; if so, it is relocated in Step 13. Steps 14 and 15 undo one of such operations if it has left a vertex of p unoccupied. Since the relocation of a non-bottleneck robot could also worsen the current bottleneck value, Steps 16–17 are required to keep track of the best configuration p^* and corresponding set of vertices P^* found so far. Finally, Step 18 is the analogous of that of Algorithm 2, but operates on the vertices of the best configuration p^* found throughout the $m-1$ iterations. Note that this algorithm could, in principle, output a connected configuration with $|P|<m$ (this follows from the selection of $\bar{w}$ from $P\cup N(P)$ instead of $N(P)$ in Step 4). However, Proposition 1 does not exclude the existence of an optimal configuration occupying less than m vertices, even when assumptions (A1) and (A2) hold.

As before, we run Algorithm 3 for all starting vertices $\hat{v}$ and we select the minimum cost configuration overall. The running time of this heuristic can be expressed as $O\big(|V|(m|V|+mT_{\text{NEIGH}}+T_{\text{LBAP}})\big)=O\big(|V|(m|V|+T_{\text{LBAP}})\big)$.

Both the above algorithms have been designed so that they could be easily parallelized among robots by equally partitioning the vertices $\hat{v}\in V$ from which constructing and evaluating connected configurations. This introduces a potential speedup of $\simeq m$, thus reducing the running time of the algorithms of almost one order of magnitude for sufficiently large teams.

We conclude this section with a simple example showing the different behaviors of forward-reconnection and backward-reconnection on the small problem instance depicted in Figs. 5(a)-(b). Here, we have three robots whose initial configuration is specified by the vertices marked with $s_i, i=1,2,3$. The edges of the graph denote both the sets C and E, and we assume unitary lengths on E. In both cases, we analyze the behavior of the algorithm during the construction of a solution from the same vertex $\hat{v}$ shown in the figure.



(a) forward-reconnection (b) backward-reconnection

Fig. 5. An example showing the behavior of forward- and backward-reconnection algorithms on a small problem instance with $G_E=G_C$ and three robots.

Let us first focus on forward-reconnection (Fig. 5(a)). At the beginning (iteration 0), robot r_3 is assigned to vertex $\hat{v}$, so that $P=\{\hat{v}\}$ and $U=\{r_1,r_2\}$ (Step 1). Then, in iteration 1, the unassigned robot closest to $N(P)$ is r_2 at distance 3 from a (Step 3), so we have $P=\{\hat{v},a\}$ and $U=\{r_1\}$ (Step 4). Finally, in iteration 2, b is the vertex in $N(P)$ closest to s_1 (Step 3), so that $P=\{\hat{v},a,b\}$ (Step 4). In this case, the final computation of the LBAP (Step 6) confirms the robots on the positions initially found by the heuristic.

Focusing now on backward-reconnection (Fig. 5(b)), we initially construct (iteration 0) a feasible configuration with $P=\{\hat{v}\}$ where all robots occupy $\hat{v}$ (Step 1). In iteration 1, the bottleneck robot $\bar{r}$ is r_1 at distance 4 from s_1 (Step 4). Therefore, r_1 is relocated in $\bar{w}=a$, i.e., its closest vertex in $P\cup N(P)$ (Steps 5-6). In this iteration, also r_2 is relocated in a (Step 12-13), leaving only r_3 in $\bar{v}$ (which would worsen its travel cost by moving to a). In iteration 2, the bottleneck robot is again r_1 at distance 3 from s_1 (Step 4). Now, r_1 is relocated in $\bar{w}=b$ (Steps 5-6), and r_2 and r_3 are left in their current vertices, since their relocation would leave them unoccupied (in Step 10, Q is now always empty). Note that, in both iterations, P^* is always updated (Steps 16-17). The obtained robots-vertices assignment is equal to the one obtained in the previous case, and does not change after the resolution of the corresponding LBAP (Step 18).

C. Special graphs

As anticipated in Section IV, there exist special environments in which the MRP can be efficiently solved to optimality. We identify such cases in a class of instances we call *bridge-connected trees* and we prove the existence of an exact polynomial-time algorithm for them. Besides providing a tractability result for fairly significant scenarios, this result also proves Theorem 2.

We call *line-connected component* a multigraph $G_i = (V_i, E_i, C_i)$ with the following properties: $|V_i| > 1$, G_{E_i} is a linear graph, and G_{C_i} is complete (that is, for any $u, v \in V_i$ it holds that $(u, v) \in C_i$). Given two line-connected components G_i and G_j, we call any $v \in V_i \cap V_j$ a *bridge* vertex and we define the following class of multigraphs.

Definition 2. *$G = (V, E, C)$ is a bridge-connected tree if it is obtained by the union of $k \geq 1$ line-connected components $G_1, \ldots, G_k$, G_E is a tree, and for any $G_i, G_j, |V_i \cap V_j| \leq 1$.*

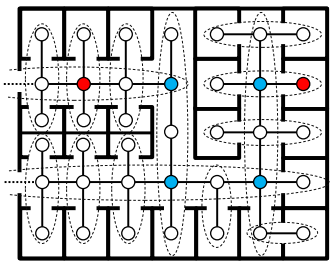


Fig. 6. A bridge-connected tree in a portion of an office floor: blue vertices form the minimal set of bridges to connect the red ones in line-of-sight.

Bridge-connected trees represent possible discretizations of different real-world scenarios. Indeed, tree graphs may represent the preferred choice for the discretization of indoor environments with a tree topology, like floors of office buildings or schools (this can also be done automatically; see [28]). From the communication point of view, examples relate to settings where one wants to secure a safe line-of-sight transmission, or where the environment is covered by bridged network infrastructures (such as wireless networks). Fig. 6 depicts a bridge-connected tree where dashed lines group the line-connected components.

It is easy to prove that the following lemma holds for bridge-connected trees.

Lemma 5. *Given a bridge-connected tree G, for any $u, v \in V$, there exists a unique minimal set of bridges $\mathcal{B}_{min}(u, v)$ (possibly empty, and computable in $O(|V|)$) such that u and v are in communication if and only if $\mathcal{B}_{min}(u, v)$ is occupied by some robots. Moreover, if two robots placed in u and v are in communication, so is any other robot lying on the (u, v)-path on G_E.*

Following the intuition provided by the lemma, the optimal algorithm we provide constructs and evaluates connected configurations under a given bottleneck requirement. That is, given a non-connected starting configuration s, we fix a pair $(\hat{r}, \hat{v})$ where $\hat{r} \in R$ and $\hat{v} \in V$ and search for a connected configuration $p_{\hat{r}\to\hat{v}}$ where robot $\hat{r}$ goes to vertex $\hat{v}$ and $B(s, p_{\hat{r}\to\hat{v}}) = d(s_{\hat{r}}, \hat{v})$. Due to this requirement, robots cannot travel a distance larger than $d(s_{\hat{r}}, \hat{v})$. Clearly, if for each $(\hat{r}, \hat{v})$ we compute $p_{\hat{r}\to\hat{v}}$ or prove that it does not exist (formally assigning $B(s, p_{\hat{r}\to\hat{v}}) = \infty$), then $p^* = \arg\min_{(\hat{r},\hat{v})} B(s, p_{\hat{r}\to\hat{v}})$ is an optimal solution for our problem. The key idea is to show that $p_{\hat{r}\to\hat{v}}$ (or the unfeasibility result) can be computed in polynomial time on bridge-connected trees. This intuition is embedded in an algorithm which we call ***BCT-Opt***.

To simplify notation let us call $B_{\hat{r}\hat{v}}$ the bottleneck requirement equal to $d(s_{\hat{r}}, \hat{v})$. The steps of BCT-Opt for a fixed $(\hat{r}, \hat{v})$ pair are reported in Algorithm 4.

Algorithm 4 BCT-Opt for fixed pair $(\hat{r}, \hat{v})$

```
1: # Compress phase
2: L = ∅
3: for each r' ∈ R \ {r̂} do
4:     let u be the closest vertex to v̂ on G_E s.t. d(s_r', u) ≤ B_r̂v̂
5:     if u ≠ v̂ then
6:         L = L ∪ {u}
7: for each l ∈ L do
8:     L' = L \ {l}
9:     if ∃u ∈ L' s.t. l ∈ (u, v̂)-path on G_E then
10:        L = L \ {l}
11: B = ⋃_{l∈L} B_min(v̂, l)
12: P = L ∪ B ∪ {v̂}
13: if |P| > m then return unfeasible
14: # Assign phase
15: p_r̂→v̂ = LBAP_VARIANT(R \ {r̂}, P \ {v̂}) ∪ (r̂, v̂)
16: if B(s, p_r̂→v̂) > B_r̂v̂ then return unfeasible
17: return p_r̂→v̂
```

BCT-Opt works in two phases: *compress* and *assign*. In the compress phase, the algorithm tries to build a "skeleton" of the solution in terms of vertices to be occupied. In particular, the algorithm starts by constructing a set $\mathcal{L}$ containing, for each robot $r' \in R \setminus \{\hat{r}\}$, the vertex u closest to $\hat{v}$ on G_E (excluding the same $\hat{v}$) reachable by r' without violating the bottleneck requirement $B_{\hat{r}\hat{v}}$ (Steps 2-6). In Steps 7-10, we delete from $\mathcal{L}$ any vertex lying on a path on G_E between $\hat{v}$ and any other $u \in \mathcal{L}$. Then, in Step 11, we compute $\mathcal{B} = \bigcup_{l\in\mathcal{L}} \mathcal{B}_{min}(\hat{v}, l)$, and use it to derive the tentative skeleton of our configuration as $P = \mathcal{L} \cup \mathcal{B} \cup \{\hat{v}\}$ in Step 12. Note that, by construction, P induces a connected subgraph on G_C where the vertices in $\mathcal{L}$ are the outermost vertices of P on G_E w.r.t. $\hat{v}$.

If it happens that $|P| > m$ we conclude that no feasible solution exists. Otherwise, we perform the assign phase composed of some additional steps. First, we solve a LBAP between robots in $R \setminus \{\hat{r}\}$ and vertices in $P \setminus \{\hat{v}\}$, setting the cost of assigning a robot r' to $w \in P \setminus \{\hat{v}\}$ equal to $d(s_{r'}, w)$ (Step 15). Note that, if $|P| < m$, we need to add $m - |P|$ dummy vertices with null cost for each robot. However, contrarily to a classical LBAP algorithm, each robot assigned to a dummy vertex is moved to its closest vertex in $P \cup N(P)$. Then, if the obtained bottleneck value is greater than $B_{\hat{r}\hat{v}}$, the algorithm concludes that no feasible solution exists for the fixed pair $(\hat{r}, \hat{v})$ (Step 16). Otherwise, it returns the connected configuration $p_{\hat{r}\to\hat{v}}$. Now we prove the following lemma.

Lemma 6. *Given an initial configuration s and bottleneck requirement $B_{\hat{r}\hat{v}}$, a connected configuration q with $q_{\hat{r}} = \hat{v}$*

and $B(s,q) = B_{\hat{r}\hat{v}}$ exists if and only if the above algorithm returns a solution.

Proof. Let us focus on the "only if" case (the "if" case follows from the algorithm construction). We denote with $p_{\hat{r}\to\hat{v}}$ the solution our algorithm would build under the $B_{\hat{r}\hat{v}}$ bottleneck requirement and with $P = \mathcal{L} \cup \mathcal{B} \cup \{\hat{v}\}$ as its underlying structure of vertices as discussed above. We say that a solution q *matches* the structure P if and only if at least one robot is placed in each of the vertices of P. We first show that any solution q with $q_{\hat{r}} = \hat{v}$ and $B(s,q) = B_{\hat{r}\hat{v}}$ with structure not matching P can be transformed into a solution matching P without violating $B_{\hat{r}\hat{v}}$. (Notice that, if $\mathcal{L} = \emptyset$, then $P = \{\hat{v}\}$ and the structure is already matched.) Then, we show that the existence of such a transformed q implies the existence of the solution computed by BCT-Opt under the bottleneck requirement $B_{\hat{r}\hat{v}}$.

Consider all the vertices $l \in \mathcal{L}$ not occupied by any robot in q, and let r_l be the robot having induced the presence of vertex l in Steps 3-6 of the algorithm (if there are multiple robots arbitrarily choose one). First, notice that even in this case Lemma 5 ensures that l and $\hat{v}$ are connected in q through $\mathcal{B}_{min}(\hat{v}, l)$, since l lies on the path between q_{r_l} and $\hat{v}$ (connected by hypothesis on q). Now, starting from q, move robot r_l to l and move "upward" to l (think of $\hat{v}$ as the root of G_E) also any other robot in the subtree on G_E rooted in l and not including the branch leading to other vertices in P. Call such tree $\mathcal{T}$. This guarantees that the robots we moved are connected to $\hat{v}$ by the previous argument, since no robot belonging to $\mathcal{B}_{min}(\hat{v}, l)$ has been moved. Moreover, this modification does not violate the bottleneck requirement. While this holds by definition for robot r_l, for any other moved robot r' there are two possibilities. If $s_{r'}$ belonged to $\mathcal{T}$, then r' can move at least up to l without violating $B_{\hat{r}\hat{v}}$ (this follows from the construction of $\mathcal{L}$). Otherwise, r' could have stopped at l, without going deeper in G_E, and hence traveling less than $B_{\hat{r}\hat{v}}$.

Being the vertices in $\mathcal{L}$ occupied by robots in the new q, it follows from Lemma 5 that all the vertices in $\mathcal{B}$ must now be occupied, and the structure P is now matched. At this point, notice that it is easy to construct from the new q a feasible solution to the LBAP solved in the *assign* phase respecting the bottleneck requirement $B_{\hat{r}\hat{v}}$ (robots assigned to dummy vertices can always be placed at least at their corresponding vertices u obtained in Step 4 without violating $B_{\hat{r}\hat{v}}$ and the connection requirement by construction), so that the obtained assignment on $P \setminus \{\hat{v}\}$ will also respect the same bottleneck. □

The following theorem straightly follows from the previous lemma, since the algorithm tries any possible combination of robot-vertex $(\hat{r}, \hat{v})$ bottleneck assignments:

Theorem 7. *BCT-Opt produces in polynomial time an optimal solution to the MRP on bridge-connected trees.*

In particular, our current implementation of BCT-Opt (which is based on off-the-shelf graph libraries) has a worst-case running time of $O\big(|V|^3 + |V|m(m^2|V| + T_{\text{LBAP}})\big)$. However, the $|V|^3$ term, which is due to the precomputation of all the shortest paths with an algorithm for generic graphs, could be reduced by writing an *ad hoc* procedure for trees. The $m^2|V|$ term, due to Steps 7-10, could also be reduced by resorting to a depth-first search. Note that this theorem implies that, while the problem is NP-hard on graph topologies where G_E is a tree and $C \supset E$ (Theorem 1), this is no longer true when $C = E$. Indeed, in this last case, Theorem 7 holds considering $|V_i| = 2$ for each line-connected component G_i.

VI. Reconnection Faults

In this section we discuss some simple techniques to handle uncertainties that will be functional to our experimental evaluation. Indeed, in realistic settings, some of the assumptions made in our resolution approach may not hold. In practice, blind executions of reconnection strategies would be subject to *reconnection faults* originating from two main issues: navigation uncertainties and false positives in the communication topology. The techniques we propose here are based on an additional coordination level computed in Step (b) of the general mission scheme outlined in Section III. Specifically, after the computation of a set of informative paths (Step (b1)) and of a connected configuration p to reach (Step (b2)), the robots compute (think of it as an additional Step (b3)) also a backup configuration p_{bkp} which will surely guarantee reconnection on a restricted version of G_C composed by the safest communication links (typically, links between locations that are within a limited distance and in line of sight).

To understand when the robots should switch to the execution of such a backup plan, recall that, under ideal conditions, reconnection in a configuration p from a starting non-connected configuration s takes place as soon as the bottleneck robot reaches its assigned location in p (as discussed in Section III, we assume that traveling costs can be interpreted either as distances or times). In practice, due to navigation errors, the travel times could be uncertain. This could advance or delay the time at which a robot reaches its destination. Let us momentarily set aside robot faults (permanent functional failures) and assume that each robot eventually reaches its location. Then, by waiting a sufficiently long time τ once reached their destinations, robots obtain tolerance for navigation time uncertainties (note that waiting is clearly preempted as soon as reconnection is established). If uncertainty is modeled with a Markov Decision Process, then τ can be defined as the minimum time horizon for which the probability of having each robot at its destination is above a chosen threshold.

If, after waiting τ time units, the team is still not connected, then the reconnection plan for p is erroneously relying on the presence of some link which turned out to be unavailable or absent. It is easy to see how this can only happen due to false positives in C (the set of communication links). This could be due to the link-detection mechanism (if, for example, the employed signal propagation model overestimates actual signal strengths [29]) or simply due to the environment (if, for example, some interference is temporarily present). In this case, each robot switches to the backup plan previously agreed

upon. This plan specifies a new connected configuration p_{bkp} computed on a restricted version of G_C composed by the safest communication links and it always exists (since eventually robots can meet in the same location). In Section VIII-A we will show how to employ waiting time and backup plans to handle reconnection recoveries.

Finally, robot faults can also be handled by adding *multiplicity requirements* to the backup configuration p_{bkp}. A multiplicity requirement is defined by an integer k and imposes that each vertex in p_{bkp} must be assigned to at least k robots. Under this requirement we can *a priori* guarantee that upon the fault of any group of at most $k-1$ robots, reconnection among all the remaining robots can always be achieved. (Notice that, called $i \leq m$ a positive integer, requiring multiplicity with $k \geq \frac{m}{i}$ forces p_{bkp} to be composed of no more than i vertices.)

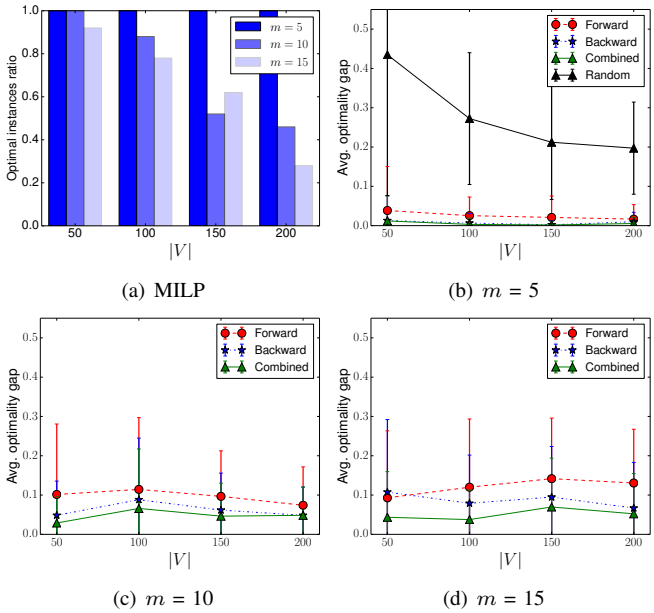
VII. Experimental Evaluation: Algorithm Performance

We start our experimental evaluation with an application-independent assessment of the resolution approach we proposed in Section V for the MRP: the exact method (Section V-A), the forward-reconnection and backward-reconnection heuristics (Section V-B), BCT-Opt (Section V-C), and the random-reconnection heuristic as baseline.

The MILP involved in the exact method is solved with GUROBI (vers. 6.0.5) [30] as follows. The model initially contains only Constraints (2)-(7) and (9)-(10). Violations of Constraints (8) are checked for each LP relaxation solution by means of the separation procedure described in [26], which is related to the resolution of a max-flow problem on a suitable graph derived from G_C. We use GUROBI's default parameters, with the only exception of giving higher branching priority to the x_v variables. The heuristics are implemented in pure Python, except for the usage of the igraph library [31], written in C, which is also used for the MILP separation procedure. All the experiments are run on a Linux machine equipped with a i5-4310M processor and 8 GB of RAM.

We generate a dataset of random instances with the following method. Choose a random point on a plane, and add it to the set of vertices. Add and connect a vertex at a random distance in $[\frac{D_{max}}{2}, D_{max}]$. Iterate this until G_E has the desired number of vertices. Complete E by randomly adding an edge to each (u, v) if $d(u, v) \leq D_{max}$. Set $G_C = G_E$ and, for any u, v where $d(u, v) \leq 1.5D_{max}$, add (u, v) in C with probability p_c. Finally, generate an initial configuration as follows. First, place randomly a robot in a vertex of the graph. Then, iteratively place other robots in vertices for which the minimum hop distance in G_C to any previously placed robot is maximum. Our experiments show that this method with $D_{max} = \frac{5}{|V|}$ and $p_c = 0.3$ is able to generate non–trivial instances. For each $(|V|, m)$ considered in our results we generate 50 random instances.

Using forward-reconnection and backward-reconnection for pruning the number of variables as discussed in Section V-A allows us to optimally solve a significant number of instances and to achieve good scalability. In particular, we are able to optimally solve slightly more instances by creating z_{vr} variables only if they can offer a solution strictly better than the one found by the heuristics – even at the price of not being able to feed the model with an initial feasible solution (we experimented both options with all the random instances generated as described above). However, in that case, the GUROBI solver is sometimes not able to produce any feasible solution within the user-defined deadline (for instance, this happens if the heuristic solution is already optimal). Therefore, if some information about the gap from the optimum is desired, feeding the model with a good initial feasible solution could be the best choice.

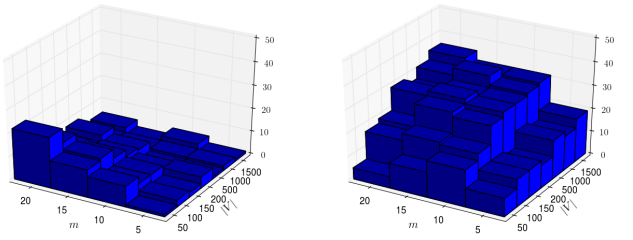


(a) MILP (b) m = 5 (c) m = 10 (d) m = 15

Fig. 7. Number of random instances solved with the exact formulation (a). Comparison between the heuristics (b), (c), (d) (optimality gap of random-reconnection for $m = 10, 15$ is very high and is not reported for clarity).

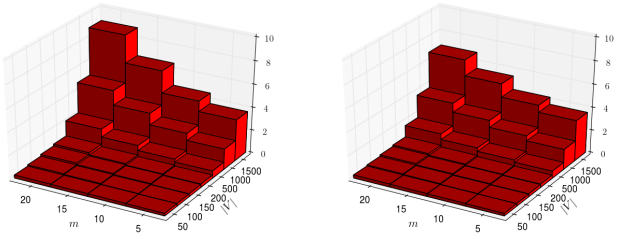
Fig. 7(a) reports the percentage of instances optimally solved by the MILP within 30 minutes. We get optimal solutions with up to 200 vertices and 15 robots. Clearly, as the instances increase in size, the percentage of terminations decreases. While we always get optimal solutions with 5 robots, the ratio decreases for 10 and 15 robots, with only a slight difference between the two ratios for a given number of vertices.

Figs. 7(b), 7(c), and 7(d) summarize the performance of the heuristics. We also consider the *combined* algorithm that outputs the best solution obtained by forward-reconnection and backward-reconnection. Each graph reports the average optimality gap of forward-reconnection and backward-reconnection over the optimally-solved instances (the gap is defined as $\frac{w-w^*}{w^*}$ where w^* and w are the optimal and heuristic costs, respectively). The average gap of the random-reconnection baseline is reported only for 5 robots for reasons of clarity since, with more robots, it lies most of the times above 80% (and always above 40%). This result already offers a key insight about the need of smarter reconnection algorithms. Note that, for a fixed m, it may be the case that the number of instances solved by the MILP is different for two different values of $|V|$: however, the comparison is consistent



(a) Forward outperforms backward (b) Backward outperforms forward

Fig. 8. Number of times forward- (backward-) reconnection outperforms backward- (forward-) reconnection on random instances.



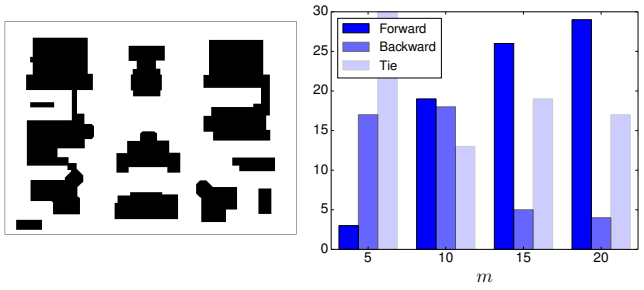
(a) Forward-reconnection (b) Backward-reconnection

Fig. 9. Average computing times on random instances (in seconds).

for a fixed number of vertices. For 5 robots, backward-reconnection turns out to outperform forward-reconnection in almost every instance, with an average gap always below 2.5%. Instead, with more robots, forward-reconnection significantly impacts on the gap achieved by the combined algorithm. Forward-reconnection works under the rationale that good solutions can be obtained with locally optimal choices from the start configuration s. The optimality of this scheme is more frequent on our random instances when the number of robots is large and vertices are few, since the optimal configuration is more likely to lie relatively nearby the starting one. When, instead, robots are few and the number of vertices is large, the optimal configuration p^* has an higher probability of involving vertices that are far away from s. This makes it more difficult to find p^* starting from s and a backward approach tends to be more profitable.

Fig. 8 shows how no heuristic among forward-reconnection and backward-reconnection fully dominates the other. The histograms count, for each robots-vertices profile, how many times one heuristic returns a solution strictly better than the one found by the other (here we consider instances with up to 20 robots and 1500 vertices). Forward-reconnection, despite not being in general as good as backward-reconnection, is able to outperform it in a significant number of instances. (The percentage of ties among the 1400 instances considered is roughly 44%.) This justifies, from an empirical point of view, their combined employment. The random-reconnection baseline, instead, is able to outperform only forward-reconnection in a single instance with $|V| = 50$ and $m = 5$ (gap 10%).

Fig. 9 reports the average computing time required by forward-reconnection and backward-reconnection. The final



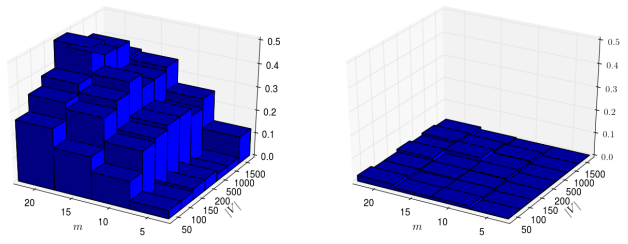
(a) (b)

Fig. 10. How many times a heuristic outperforms the other with a fine-grained discretization (b) on the real environment (a).

solution is obtained almost instantaneously for up to 500 vertices, and scales according to our complexity analysis for larger instances. In general, up to 10 s on average are required for 20 robots and 1500 vertices. However, by resorting to a parallelized implementation completely written in C, a solution could potentially be obtained in fractions of a second. The computing times of random-reconnection are not reported, being almost comparable with those of backward-reconnection (only slightly shorter).

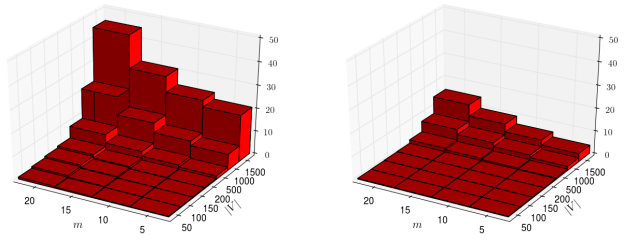
Fig. 10(a) shows a top-view of a real outdoor environment whose area is approx. 300 m $\times$ 225 m (free area is depicted in white). We discretize the environment obtaining a uniform grid graph G_E in which vertices correspond to centers of free cells with edge equal to 15 m (209 vertices). G_C is obtained with a link-detection mechanism using limited distance and line of sight between locations (the distance range is set to 56 m). We consider 5 robots starting from 50 random configurations (drawn with the same method previously described). Results show how the MILP always terminates within 30 minutes, while the average gaps of forward-reconnection, backward-reconnection, and their combined usage are very low and equal to 3.6%, 0.7%, and 0.5%, respectively. The gap of random-reconnection, instead, is much larger (24%). We re-run the same set of experiments by switching to a more fine-grained discretization with 7.5 m for the cell edge (838 vertices). Surprisingly, in 90% of the instances the solutions found by forward-reconnection or backward-reconnection on the fine-grained grid outperform the optimal solution on the coarse-grained one (average gap of 10%). This suggests how the optimality loss from the heuristic can be mitigated by a more accurate environment representation. For completeness, we report in Fig. 10(b) the number of times a heuristic outperforms the other in the fine-grained discretization for different team sizes. The results confirm our previous findings: when the optimal configuration tends to be far from the starting one (instances with 5-10 robots), backward-reconnection can work better. In the opposite case (instances with 15-20 robots), since the bottleneck robot needs to travel only a few grid cells to reconnect the team, forward-reconnection is better. Even in this case, random-reconnection shows a very poor performance, never being able to provide better solutions than those found by the two informed heuristics.

Finally, we evaluate BCT-Opt (Section V-C) on a set of



(a) Forward-reconnection (b) Backward-reconnection

Fig. 11. Heuristics average optimality gap on bridge-connected trees.



(a) BCT-Opt (b) Backward-reconnection

Fig. 12. Average computing times on bridge-connected trees (in seconds).

50 random bridge-connected trees (Definition 2). We start from a comparison with forward-reconnection and backward-reconnection, whose optimality gaps against BCT-Opt are reported in Fig. 11. While forward-reconnection shows poor performance, backward-reconnection obtains a very small average gap overall. The random-reconnection baseline behaves extremely bad even on bridge-connected trees, as its average optimality gaps (not shown in the figure) are always much larger than those of forward-reconnection for all the considered vertices-robots pairs. Comparing the computing time required by BCT-Opt and backward-reconnection in Fig. 12, we can see that backward-reconnection could be a better option for large instances (at the expense of optimality guarantee). The intuition behind this is that BCT-Opt tries to build a feasible solution from all the possible vertex-robot pairs, while the heuristics try to build feasible solutions just by enumerating all the possible initial vertices.

VIII. Experimental Evaluation: Comparison in a Use Case

To show the relevance of our method, in this section we apply it to a significant application setting for which resolution techniques have been proposed in literature: the Multirobot Informative Path Planning with Periodic Connectivity problem (MIPP-PC) introduced in [14].

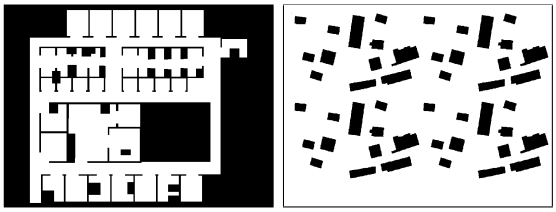
In the MIPP-PC, a team of m robots plans a set of joint paths maximizing an arbitrary objective function $F(\cdot)$ over a time horizon T and under the constraint of reconnecting every T_I time units. The environment is modeled, as we do in this paper, as a physical graph G_E and a communication graph G_C. Let us call A_r and $C(A_r)$ a generic path of robot r and the corresponding temporal cost, respectively. The MIPP-PC seeks a collection of paths $A_1, \dots, A_m$ such that: $C(A_r) \leq T$ for any robot r, the team is connected at τT_I for $\tau \in \{0, \dots, \lfloor T/T_I \rfloor\}$, and $F(\cdot)$ is maximized.

In [14], authors tackle the problem with a receding horizon approach. For $i = 1, \dots, m$, according to a lexicographic order, they first optimize A_i leaving $A_{j \neq i}$ fixed (robots with $j > i$ are initially assumed to remain at their starting position). Optimization of A_i is performed by sampling feasible paths over the horizon $\{0, \dots, T_I\}$. Once the path of the last robot m has been computed, optimization restarts from the first robot (subsequent robots will now have paths moving them around). Robots paths are cyclically generated in such a way until a termination criterion is met (convergence or a maximum number of iterations). Then, the robots follow the planned paths and the above procedure is repeated. A target search problem is then cast to the MIPP-PC and its resolution is experimentally evaluated. In such a problem, the objective function $F(\cdot)$ is instantiated to the (discounted) probability of finding the target within the time horizon and the constraint of reconnecting every 5 time units is imposed to the team. Due to the relatively tight reconnection period, the search space for each robot's path is limited and a complete enumeration of all the feasible paths can be done in reasonable time. However, with larger reconnection periods, a complete enumeration would be too computationally expensive. Also, the work leaves open the problem of recovery in presence of errors. We show how our MRP resolution approach presented in Section V, together with the basic recovery techniques discussed in Section VI, can be effectively used for dealing with these two issues.

To this end, we present a planning algorithm for the MIPP-PC obtained by a refinement of the mission scheme presented in Section III and based on the employment of our resolution approach for the MRP. The algorithm, called *UFSR*, develops according to three macro-phases: *Unconstrained joint planning*, *Feasibility pruning*, and *Solution Refinement*. The pseudo-code is shown in Algorithm 5.

Algorithm 5 UFSR algorithm

```
# Unconstrained joint planning
A = unconstrained_joint_planning(F, T_I)
if A is connected at T_I then
    return A
# Feasibility pruning
i = 0
repeat
    i = i + 1
    let Ā be the restriction of A to the first T_I − iδ steps
    let s be the final joint position in Ā
    A = Ā ∪ MRP(s)
until A is connected within T_I
# Solution refinement
t_r = T_I − iδ
while t_r + δ' ≤ T_I do
    truncate A at t_r
    generate a set of feasible joint moves of length δ':
    J_C = {J_1, ..., J_|J_C|}
    J* = arg max_{i=1,...,|J_C|} F(J_i)
    A = A ∪ J*
    t_r = t_r + δ'
return A
```



(a) SDR (b) MOUT

Fig. 13. MIPP-PC simulation environments.

In the *Unconstrained joint planning* phase (Step 2), a set of joint paths A maximizing $F(\cdot)$ along $\{0, \dots, T_I\}$ is computed by disregarding reconnection. To do this, we employ a relaxed variant of the iterative scheme from [14]. If A is connected at T_I, then the robots can execute the set of paths, otherwise we proceed to the next phase.

In the *Feasibility pruning* phase (Steps 6-12), A is modified to reconnect at T_I according to the following iterative procedure. Given an integer $\delta \leq T_I$, consider subpaths of A, $\bar{A}$, of length equal to $T_I - i\delta$, with i being the current iteration of the procedure. Then, build a new A by solving a MRP problem where robots have to reconnect from their last positions in $\bar{A}$ in at most $i\delta$ steps. We repeat this procedure increasing i until a feasible A is obtained.

In the *Solution Refinement* phase (Steps 14-22), we try to improve A (now ensured to be feasible) by considering *feasible joint moves* of step size δ' (possibly different from the previous δ). More precisely, call t_r the time step for which the previous phase returned a valid reconnection path. Now, fix the solution A until step t_r, momentarily forgetting the plan for the subsequent steps, and generate a set of candidate feasible joint moves of length δ', $J_C = \{J_1, \dots, J_{|J_C|}\}$, where each J_i has the form $\{\{A_1^{t_r} = \bar{A}_1^{t_r}, \dots, A_1^{t_r+\delta'}\}, \dots, \{A_m^{t_r} = \bar{A}_m^{t_r}, \dots, A_m^{t_r+\delta'}\}\}$, i.e., it represents a set of m paths of length δ', one for each robot, starting from the corresponding robots' positions in A at t_r. For enforcing feasibility, we run an algorithm for the MRP for each candidate J_i and keep it only if the former has returned a solution respecting a time budget of $T_I - (t_r + \delta')$. We then choose among J_C the best joint move according to $F(\cdot)$, consistently update t_r, and iterate until t_r exceeds T_I. Note that, if $t_r + \delta' \leq T_I$, we can always construct a non-empty J_C by exploiting the previous MRP solution.

A. MIPP-PC experiments

We develop a numerical multirobot target search simulator in Python/NumPy, similar to that used in [14], and consider the same environments (Fig. 13): the SDR building from Radish dataset [32] ($80\,\mathrm{m} \times 60\,\mathrm{m}$) and McKenna MOUT site ($400\,\mathrm{m} \times 300\,\mathrm{m}$), where communication range is set to one fourth of the map diagonal. Differently from [14], we adopt a uniform grid discretization to have a more precise timing in gaining global reconnection. We set cell sizes of $1.5\,\mathrm{m} \times 1.5\,\mathrm{m}$ and $8\,\mathrm{m} \times 8\,\mathrm{m}$ for the SDR building and the McKenna MOUT site, respectively (1037 and 1096 free cells, respectively). At each time step, the target randomly moves to an adjacent cell and robots can detect it from a distance of at most 2 cells. We assume the presence of a link-detection mechanism which computes the signal strength S at a distance d_m as $S = P_{d_0} - 10N\log_{10}(d_m/d_0)$ where d_0 is the distance reference value (set to one fourth of the map diagonal), P_{d_0} is the signal strength at d_0 (set to the typical cutoff value of $-93\,\mathrm{dB\,m}$), and N is the path loss factor (set to 2) [29].

We consider two settings: with and without the presence of errors. In the error-free setting, robots' motion is perfect and the link-detection mechanism correctly computes the availability of communication links as pairs of cells whose signal strength S is higher than $-93\,\mathrm{dB\,m}$. In the error-prone setting, we assume that robots can be affected by motion errors: any intended movement will not occur with probability 0.1. Moreover, if two vertices are in limited-distance line-of-sight or S between them is above $-83\,\mathrm{dB\,m}$, they are connected by a true link in G_C. Otherwise, the link will be a false positive with probability 0.25. Links that satisfy the limited-distance line-of-sight condition are used to compute backup plans. In both settings, we assume that each robot will remain functional throughout the whole mission. We set T_I to 24 time units and compare the following three algorithms:

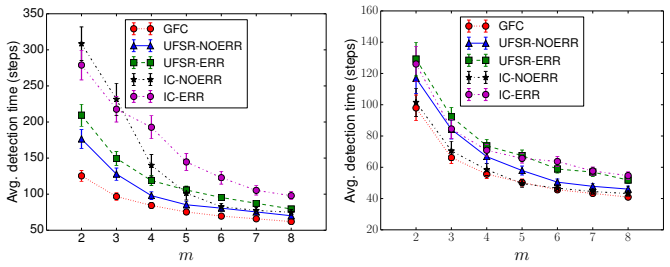
Implicit coordination with complete path enumeration, 4-periodic (IC): the algorithm from [14] where path enumerations are made along 4 time units. (With our discretization, an enumeration along 5 or more time units would be computationally prohibitive.) In absence of errors, reconnecting every 4 steps implies reconnection every 24 steps.

UFSR, 24-periodic: the algorithm we described above. In the unconstrained joint planning step, robots' paths are chosen greedily, in the feasibility pruning step, we set $\delta = 1$; in the solution refinement step, we set $\delta' = 1$ and use a simple greedy strategy to generate candidate joint moves. All the MRPs are solved by running in parallel forward- and backward-reconnection and choosing the best solution (combined algorithm).

Greedy with full communication (GFC): this algorithm corresponds to the first phase of the UFSR algorithm and is run assuming perfect global communication among all the graph vertices. Greedy plans are recomputed each 4 time steps.

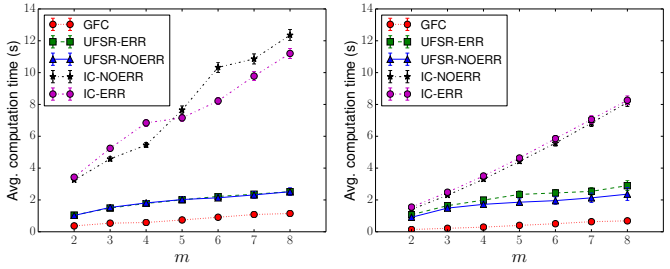
Finally, we adapted IC and UFSR to handle reconnection faults as discussed in Section VI. We fix τ values to 1 and 5 for IC and UFSR, respectively, and we derive worst-case lower bounds on the probability of being reconnected after $4 + 1$ and $24 + 5$ steps as 0.991^m and 0.978^m, respectively, assuming that the final configuration will be connected in G_C.

In order to provide a fair and implementation-independent comparison of the quality of the solutions returned by the algorithms, we assume planning to be instantaneous. Figs. 14(a) and 14(b) show the average detection times (plotted with the 95% confidence interval) obtained on 250 runs in the SDR and MOUT environments. Focusing on the SDR environment in the error-free setting, it is possible to see that UFSR allows to obtain better performance than IC up to 4 robots; then, performance is comparable with that of IC. This is what we expected: for large teams, there are greater chances that all the robots are able to spread across the environment



(a) SDR (b) MOUT

Fig. 14. Average target detection times (in time units).



(a) SDR (b) MOUT

Fig. 15. Computing times for planning of IC, UFSR, and GFC.

while maintaining global connectivity, even when they are constrained to reconnect after few time units. In the same environment, the introduction of errors significantly impacts on the performance of IC for more than 3 robots, while, for UFSR, performance is only slightly worse: this results in generally better performance for UFSR. In the MOUT environment we obtain significantly smaller gaps. Surprisingly, GFC turns out to be substantially comparable to IC for all the team sizes. This is mainly due to the openness of the MOUT environment, combined with the relatively large extension of the communication range. The results suggest that, in such an open environment, it might be generally better to keep the whole team under frequent communication, reoptimizing the plan often. Few cases aside, IC provides lower detection times than UFSR, both in the error-free and error-prone settings.

To conclude, Fig. 15 reports the planning times of our current implementation, assuming that all the robots contribute to the resolution of the MRPs in parallel. UFSR is always able to obtain plans in less than $4\,\mathrm{s}$, hence being suitable for real-time settings.

IX. Conclusions

Solving the Multirobot Reconnection Problem (MRP) is a fundamental challenge for several multirobot systems operating under communication constraints. In this paper, we formalized the MRP within the scope of a generic multirobot mission and we showed its hardness in the general case and the existence of a polynomial-time exact algorithm for special cases. We also proposed exact and heuristic methods, supported by error-handling procedures, and assessed them with extensive tests, also in a periodic reconnection use case.

A first promising avenue for future research is related to the identification of more sophisticated methods to compute the connectivity graph G_C, since current methods are mainly suited for ensuring connectivity in a conservative way. Recently, we have proposed online multirobot navigation strategies for collecting samples to build a signal strength map of the environment in terms of a Gaussian Process [33], [34]. These strategies could be used to estimate G_C in a less conservative, yet still sound, way. We are currently working on offline path planning algorithms to be used for building G_C when the environment is fully known in advance [35].

Moreover, we envision future research focusing on the development of sophisticated techniques for handling reconnection faults. Network reliability [36] is a topic deeply connected to this task, since the problem of computing the reconnection probability of a robot configuration under different sources of uncertainties can be translated to such a #P-complete problem[2]. How to exploit network reliability techniques in our setting is an interesting and promising direction for future work.

Finally, we believe that our hardness results can be further specified by showing the NP-hardness of the MRP in cases where G_E is a 2D grid graph coincident with G_C. To date, we could not find any reduction to prove this conjecture, leaving the problem open for further investigation.

References

[1] R. Arkin and J. Diaz, "Line-of-sight constrained exploration for reactive multiagent robotic teams," in *Proc. AMC*, pp. 455–461, 2002.
[2] M. Rooker and A. Birk, "Multi-robot exploration under the constraints of wireless networking," *Control Eng Pract*, vol. 15, no. 4, pp. 435–445, 2007.
[3] P. Mukhija, K. Krishna, and V. Krishna, "A two phase recursive tree propagation based multi-robotic exploration framework with fixed base station constraint," in *Proc. IROS*, pp. 4806–4811, 2010.
[4] Y. Pei, M. Mutka, and N. Xi, "Connectivity and bandwidth-aware real-time exploration in mobile robot networks," *Wireless Communications and Mobile Computing*, vol. 13, no. 9, pp. 847–863, 2013.
[5] J. Banfi, A. Quattrini Li, N. Basilico, F. Amigoni, and I. Rekleitis, "Asinchronous multirobot exploration under recurrent connectivity constraints," in *Proc. ICRA*, pp. 5491–5498, 2016.
[6] V. Spirin, S. Cameron, and J. de Hoog, "Time preference for information in multiagent exploration with limited communication," in *Proc. TAROS*, pp. 34–45, 2013.
[7] E. Jensen, E. Nunes, and M. Gini, "Communication-restricted exploration for robot teams," in *Multiagent Interaction without Prior Coordination Workshop at AAAI*, 2014.
[8] M. Hsieh, A. Cowley, V. Kumar, and C. Taylor, "Maintaining network connectivity and performance in robot teams," *J Field Robot*, vol. 26, pp. 111–131, 2008.
[9] A. Mosteo, L. Montano, and M. Lagoudakis, "Guaranteed-performance multi-robot routing under limited communication range," in *Proc. DARS*, pp. 491–502, 2009.
[10] D. Anisi, P. Ogren, X. Hu, and T. Lindskog, "Cooperative surveillance missions with multiple unmanned ground vehicles (UGVs)," in *Proc. CDC*, pp. 2444–2449, 2008.
[11] M. Zavlanos and G. Pappas, "Distributed connectivity control of mobile networks," *IEEE Trans Robot*, vol. 24, no. 6, pp. 1416–1428, 2008.
[12] Y. Kantaros and M. Zavlanos, "Simultaneous intermittent communication control and path optimization in networks of mobile robots," in *Proc. CDC*, pp. 1794–1799, 2016.
[13] Y. Kantaros and M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Trans Automat Contr*, vol. 62, no. 7, pp. 3109–3121, 2016.

[2]#P denotes the class of problems computed by nondeterministic polynomial-time Turing machines that have the additional facility of outputting the number of accepted computations [37].

[14] G. Hollinger and S. Singh, "Multirobot coordination with periodic connectivity: theory and experiments," *IEEE Trans Robot*, vol. 28, no. 4, pp. 967–973, 2012.

[15] F. Amigoni, J. Banfi, and N. Basilico, "Multirobot exploration of communication-restricted environments: A survey," *IEEE Intell Syst*, vol. 32, no. 4, pp. 48–57, 2017.

[16] E. Stump, N. Michal, V. Kumar, and V. Isler, "Visibility-based deployment of robot formations for communication maintenance," in *Proc. ICRA*, pp. 4498–4505, 2011.

[17] J. de Hoog, S. Cameron, and A. Visser, "Autonomous multi-robot exploration in communication-limited environments," in *Proc. TAROS*, pp. 68–75, 2010.

[18] A. Torsten, *Autonomous Exploration by Robot Teams: Coordination, Communication, and Collaboration*. PhD thesis, Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria, 2015.

[19] A. Kolling and S. Carpin, "Extracting surveillance graphs from robot maps," in *Proc. IROS*, pp. 2323–2328, 2008.

[20] A. Visser and B. A. Slamet, "Including communication success in the estimation of information gain for multi-robot exploration," in *Proc. WiOPT*, pp. 680–687, 2008.

[21] A. Das, J. Spletzer, V. Kumar, and C. Taylor, "Ad hoc networks for localization and control," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 3, pp. 2978–2983, 2002.

[22] A. Quattrini Li, F. Amigoni, and N. Basilico, "Searching for optimal off-line exploration paths in grid environments for a robot with limited visibility," in *Proc. AAAI*, pp. 2060–2066, 2012.

[23] D. Williamson and D. Shmoys, *The design of approximation algorithms*. Cambridge University Press, 2011.

[24] D. Anisi, *On cooperative surveillance, online trajectory planning and observer based control*. PhD thesis, Royal Institute of Technology, Stockholm, Sweden, 2009.

[25] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *J Intell Robot Syst*, vol. 57, no. 1-4, pp. 65–100, 2010.

[26] E. Álvarez-Miranda, I. Ljubić, and P. Mutzel, "The maximum weight connected subgraph problem," in *Facets of Combinatorial Optimization* (M. Jünger and G. Reinelt, eds.), pp. 245–270, Springer, 2013.

[27] R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009.

[28] H. Choset and J. Burdick, "Sensor-based exploration: the hierarchical generalized Voronoi graph," *Int J Robot Res*, vol. 19, no. 2, pp. 96–125, 2000.

[29] P. Bahl and V. Padmanabhan, "Radar: An in-building RF-based user location and tracking system," in *Proc. INFOCOM*, pp. 775–784, 2000.

[30] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2015.

[31] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006.

[32] A. Howard and N. Roy, "The robotics data set repository (Radish)," 2003.

[33] J. Banfi, A. Quattrini Li, N. Basilico, I. Rekleitis, and F. Amigoni, "Multirobot online construction of communication maps," in *Proc. ICRA*, pp. 2577–2583, 2017.

[34] P. Penumarthi, A. Quattrini Li, J. Banfi, N. Basilico, F. Amigoni, J. O'Kane, I. Rekleitis, and S. Nelakuditi, "Multirobot exploration for building communication maps with prior from communication models," in *Proc. MRS*, pp. 90–96, 2017.

[35] A. Riva, J. Banfi, C. Fanton, N. Basilico, and F. Amigoni, "A journey among pairs of vertices: Computing robots' paths for performing joint measurements," in *Proc. AAMAS*, 2018. To appear.

[36] C. Colbourn, *The combinatorics of network reliability*. Oxford University Press, 1987.

[37] L. Valiant, "The complexity of computing the permanent," *Theor Comput Sci*, vol. 8, no. 2, pp. 189 – 201, 1979.

Jacopo Banfi received a M.Sc. degree in Computer Science in 2014 and a Ph.D. in Information Technology in 2018 from the Politecnico di Milano (Italy). From February 2013 to April 2014, he worked at his M.Sc. thesis at the Dalle Molle Institute for Artificial Intelligence Studies (Manno, Switzerland). Since December 2017, he is a research assistant at the University of Milan (Italy). His research interests lie in the field of cooperative multirobot systems, with a particular focus on informative and constrained multirobot path planning problems.

Nicola Basilico (M'16) received a M.S. degree in Computer Science in 2007 and a Ph.D. in Information Technology in 2011 from the Politecnico di Milano (Italy). Since 2014, he is Assistant Professor in Computer Science at the University of Milan (Italy). His main research interests develop in the artificial intelligence area with focus on multi-agent systems, algorithmic game theory, and autonomous robotics.

Francesco Amigoni (SM'17) got the Laurea degree in Computer Engineering from the Politecnico di Milano in 1996 and the Ph.D. degree in Computer Engineering and Automatica from the Politecnico di Milano in 2000. From December 1999 to September 2000 he has been a visiting scholar at the Computer Science Department of the Stanford University (USA). From February 2002 to April 2007 he has been an assistant professor at the Politecnico di Milano. From May 2007 he is an associate professor at the Dipartimento di Elettronica, Informazione e Bioingegneria of the Politecnico di Milano. His main research interests include: agents and multiagent systems, autonomous mobile robotics, and the philosophical aspects of artificial intelligence and robotics.