The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

# Multiagent Connected Path Planning: PSPACE-Completeness and How to Deal with It

**Davide Tateo, Jacopo Banfi, Alessandro Riva,
Francesco Amigoni, Andrea Bonarini**
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Piazza Leonardo da Vinci, 32, Milano, Italy
{davide.tateo, jacopo.banfi, alessandro.riva, francesco.amigoni, andrea.bonarini}@polimi.it

## Abstract

In the Multiagent Connected Path Planning problem (MCPP), a team of agents moving in a graph-represented environment must plan a set of start-goal joint paths which ensures global connectivity at each time step, under some communication model. The decision version of this problem asking for the existence of a plan that can be executed in at most a given number of steps is claimed to be NP-complete in the literature. The NP membership proof, however, is not detailed. In this paper, we show that, in fact, even deciding whether a feasible plan exists is a PSPACE-complete problem. Furthermore, we present three algorithms adopting different search paradigms, and we empirically show that they may efficiently obtain a feasible plan, if any exists, in different settings.

## Introduction

The Multiagent Connected Path Planning problem, MCPP for short, was formally introduced in (Hollinger 2010; Hollinger and Singh 2012) as a simpler version of the Multirobot Informative Path Planning with Periodic Connectivity problem (MIPP-PC). In MCPP, a team of agents moving in a graph-represented environment must plan a set of start-goal joint paths which ensures global (multi-hop) connectivity at each time step, under some communication model.

Besides being useful in the context of MIPP-PC, the computational study of this problem is interesting on its own since it may silently appear as a subproblem in several information-gathering missions. For instance, in multirobot exploration, adopting continuous connectivity with a supervising base station can ensure the possibility of manually intervening in case of robots' faults and of delivering video streams from the robots to the base station (Rooker and Birk 2007). Another example is given by connectivity-aware planning for search and rescue (Feo Flushing et al. 2013), where particular connectivity requirements can give rise to a MCPP subproblem to solve. In (Bhattacharya, Likhachev, and Kumar 2010), distance constraints are incorporated in a simultaneous path planning and task execution problem. Distance constraints can be interpreted as particular communication constraints when full communication within a given range is assumed.



In (Hollinger 2010; Hollinger and Singh 2012), it is claimed that the decision version of MCPP asking for the existence of a plan that can be executed in at most a given number of steps is NP-complete. The NP membership proof, however, is not detailed. In this paper, we show that, in fact, even deciding whether a feasible plan exists is a PSPACE-complete problem. Our theoretical result implies that, not only a polynomial-time feasibility algorithm is unlikely to exist (unless P=PSPACE), but also feasibility certificates of polynomial size might be out of reach (unless NP=PSPACE). We also consider a variation of MCPP able to model collisions among agents, generalizing the graph-based Multirobot Path Planning problem (MPP) introduced by (Yu and LaValle 2013b), and show that our proof holds even in this case. This is interesting, since the MPP feasibility decision problem is in P (Yu and Rus 2015), while MPP time- and distance-optimal decision problems are NP-complete (Yu 2016; Banfi, Basilico, and Amigoni 2017).

Our negative theoretical result on the complexity of MCPP does not rule out the existence of problem instances that can be solved efficiently. Therefore, as an additional contribution, we present three algorithms for MCPP working under different search paradigms and we empirically show that they may efficiently obtain a feasible plan, if any exists, in different settings. To the best of our knowledge, these are the first practical algorithms specifically designed for MCPP.

It is worth mentioning that MCPP shares some similarities with the problem of connectivity maintainance as investigated by the Control community. In that case, the agents are usually required to maintain global connectivity either while pursuing basic coordinated tasks (like rendez-vous, formation control, and flocking) (Zavlanos, Egerstedt, and Pappas 2011) or in response to additional control terms, such as the teleoperation of one of the team members (Sabattini et al. 2013).

## Problem Definition

We consider an environment modeled as a multigraph $G = (V, E, C)$ describing both its physical and communication features. Locations are modeled by the set of vertices $V$ that can be obtained by discretizing the environment either by hand, or with the help of some automated technique (see, e.g., (Stump et al. 2011)). The first edge set, $E$, encodes the physical topology of the environment. The second set

of edges, $C$, encodes the communication topology, namely, the availability of a communication link between a pair of locations. As commonly done (Hollinger and Singh 2012; Stump et al. 2011), it is assumed that the set $C$ is not affected by false positives and that it does not vary as time evolves. These properties can be always enforced by constructing $C$ in a conservative way (for instance, by considering a disk communication model with small range). All the edges of the multigraph are assumed to be unweighted and undirected. Throughout the paper, we will use the following notation: $G_E = (V, E)$ and $G_C = (V, C)$.

Let $A = \{a_1, \ldots, a_m\}$ be a set of $m$ agents moving in $G$ as follows. Time evolves in discrete steps $t \in \mathbb{N}_0$: at any step, an agent can either remain still at its current vertex, or move along a physical edge (edges in $E$ have all the same length and can all be traversed in a single step). Two or more agents are allowed to occupy the same vertex at a time and to move along the same edge between two subsequent steps. In a robotic setting, this could be obtained by resorting to a local collision-avoidance mechanism (Hollinger and Singh 2012).

At a generic step $t$, the positions of the agents in $G$ are specified by a *state* $\pi_t = \langle p_1, p_2, \ldots, p_m \rangle$, where each $p_i \in V$ denotes the vertex where the $i$-th agent is located. We say that a state is *connected* if and only if the subgraph of $G_C$ induced by its occupied vertices is connected.

The agents must move from a start state $\pi^s$ to a goal state $\pi^g$, which are both connected. The Multiagent Connected Path Planning problem can be stated as follows:

**Problem 1.** *Given $\langle G, A, \pi^s, \pi^g \rangle$, find a time-stamped sequence of states $\pi^s = \pi_1, \pi_2, \ldots, \pi_k = \pi^g$ such that (1) each $\pi_j$ can be obtained from $\pi_{j-1}$ by moving each agent along at most one edge in $E$, and (2) each $\pi_j$ is connected.*

## PSPACE-Completeness

In this section, we prove that the decision version of Problem 1, called MCPP-D, is PSPACE-complete. Being a decision problem, in MCPP-D we simply ask for the existence of a time-stamped sequence of connected states $\{\pi_j\}$ leading from $\pi^s$ to $\pi^g$. To prove the PSPACE-completeness result, we first provide an overview of the Nondeterministic Constraint Logic (NCL) model of computation (Hearn and Demaine 2005). Then, we prove the PSPACE-hardness of MCPP-D by reducing it from a particular NCL decision problem. Finally, we argue about the PSPACE-membership of MCPP-D. Additionally, we discuss some MCPP-D variants of practical interest, whose PSPACE-completeness follow straightforwardly from the content of this section.

### Nondeterministic Constraint Logic

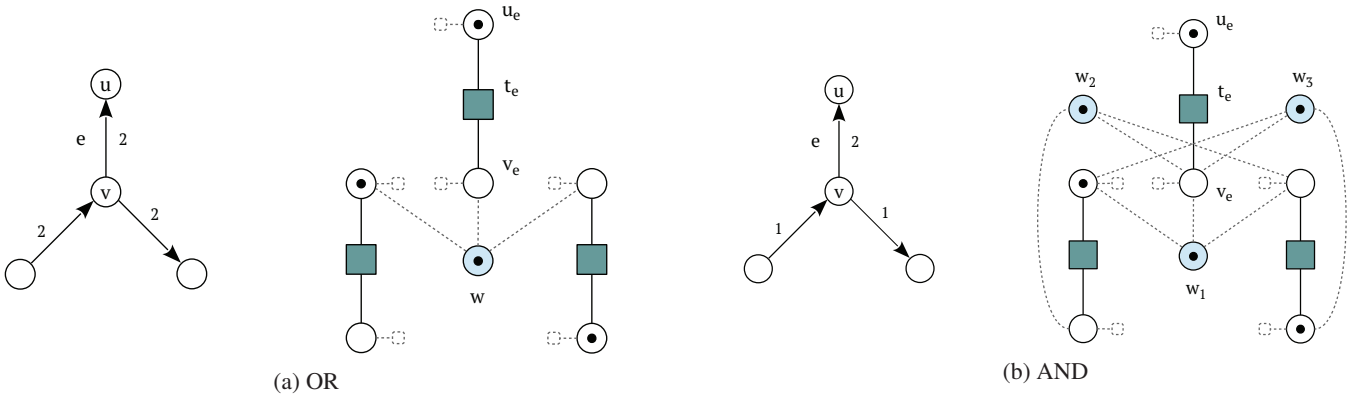
An NCL machine is specified by a *constraint graph*. This is an undirected graph $\widehat{G} = (\widehat{V}, \widehat{E})$ together with an assignment of non-negative integers to edges, called *weights*, and integers to vertices, called *minimum in-flow constraints*. A *configuration* of an NCL machine is an orientation of the edges such that the sum of incoming edge weights at each vertex is greater or equal to the minimum in-flow constraint of that vertex. A *move* from one configuration to another configuration is simply the reversal of a single edge such that the minimum in-flow constraints remain satisfied. Having defined the NCL machinery, the *configuration-to-configuration* decision problem, NCL-C2C for short, can be stated as follows:

**NCL-C2C**
INSTANCE: a constraint graph $\widehat{G} = (\widehat{V}, \widehat{E})$, and two configurations $A$ and $B$.
QUESTION: is there a sequence of moves leading from $A$ to $B$ in $\widehat{G}$?

It has been shown that NCL-C2C remains PSPACE-complete even when restricting the constraint graph to a particular simple (without self loops) topology, called AND/OR (Hearn and Demaine 2005). An AND/OR (simple) constraint graph is composed by only two types of vertices, called AND and OR, whose minimum in-flow constraint is fixed to 2. Both AND and OR vertices have three incident edges, whose weights are 1, 1, 2 for the former and 2, 2, 2 for the latter. An AND vertex takes its name from the fact that it behaves somehow like a logical AND: the edge with weight 2 can be directed outward if and only if both the edges with weight 1 are directed inward. An OR vertex, instead, behaves somehow like a logical OR: at least one edge must always been directed inward. Note that non-legal edge orientations depend on the specific type of vertex. For instance, in an OR vertex, its three incident edges cannot be oriented outward at the same time, but all the other possible combinations of orientations are valid. Henceforth, we will use AND/OR-NCL-C2C to denote the class of NCL-C2C instances where the constraint graph $\widehat{G}$ is simple and made of only AND and OR vertices.

### PSPACE-Hardness: Overview of the Reduction

Before going into the reduction details, it is convenient to provide a brief sketch of the reduction. From a *generic* instance of AND/OR-NCL-C2C, we show how to construct (in polynomial time) a *particular* instance of MCPP-D such that the former admits a *yes* answer if and only if the latter admits a *yes* answer. In particular, the AND/OR vertices defining the constraint graph $\widehat{G}$ of the generic AND/OR-NCL-C2C instance will be associated with two particular substructures of an MCPP-D multigraph $G$, called AND/OR gadgets, which will be implicitly composed together according to the topology of $\widehat{G}$. These substructures will be populated with some agents, whose legal movements in the physical graph $G_E$ (namely, movements that ensure global connectivity on the connectivity graph $G_C$ between two subsequent steps) will reflect the possibility of performing a single edge reversal in the AND/OR-NCL-C2C instance, and *vice versa*. In order to ensure this property, we will exploit an additional substructure contributing to forming the final MCPP-D graph $G$, containing a single agent called *coordinator*. The movements of the coordinator will allow to replicate, in the generic AND/OR-NCL-C2C instance, the fact that edge reversals have to be performed one at a time.




(a) OR (b) AND

Figure 1: From AND/OR NCL constraint graph vertices (left) to AND/OR MCPP-D gadgets (right). Dark green squares are the transition vertices, while light blue circles are the guard locations vertices. Solid lines represent edges in $E$, while dashed lines represent edges in $C$. All the white vertices in the multigraph are connected with each other in $G_C$: these communication links are here represented as small dashed boxes. The OR edges (a) are shown with a legal orientation, while the AND edges (b) with a non-legal one.

## PSPACE-Hardness: Reduction Details

We start by focusing on how to represent a generic NCL configuration as a state of a particular MCPP problem, deferring all the transition constraints to a later stage. Figure 1 shows how the AND/OR vertices of the NCL constraint graph $\widehat{G}$ are mapped to AND/OR gadgets of a multigraph $G$. For each edge $e = (u, v)$ in $\widehat{E}$, we create three vertices $u_e, t_e, v_e \in V$, and connect them in $G_E$ in sequence as shown. We call the $t_e$ vertices *transition vertices*. To represent the orientation of the edge $e$, we spawn an agent on this structure: if the edge points to $u$, the agent will be forced to reside on $u_e$, otherwise it will lie on $v_e$. All the $u_e$ and $v_e$ vertices obtained in this way (from all the edges in $\widehat{E}$) are connected with each other in $G_C$, in other words, the subgraph of $G_C$ induced by the $u_e$ and $v_e$ vertices is a clique. Finally, note that each edge $e \in \widehat{E}$ is mapped to a sequence of three vertices $u_e, t_e, v_e$ which appears as a substructure of two different AND/OR gadgets.

As a result, on the multigraph $G$ constructed so far, the agents' positions can be used to univocally identify the orientations in the constraint graph $\widehat{G}$, and *vice versa*. In particular, moving an agent from one extreme to another in each line of three vertices equals to change the corresponding edge orientation in the constraint graph $\widehat{G}$. We are thus able to represent the start and goal configuration $A, B$ as states $\pi^A, \pi^B$. However, there are still inconsistencies to solve. First, in the current multigraph, any agent position can contribute to the formation of a feasible state, with the exception of the transition vertices. Second, multiple agents could move simultaneously – but orientations in the constraint graph can be changed one at a time. Third, the agents cannot really move, since, from what said so far, each transition vertex $t_e$ does not communicate with any other vertex.

Let us start by addressing the first issue. In the constraint graph, the legality of a configuration is formally represented by the conjunction of the logical constraints imposed by each vertex. In fact, it is more convenient to focus on non-legal combinations of edge orientations locally to each vertex: if we are able to exclude their presence in the AND/OR gadgets, a connected state of agents would be readily converted in a legal configuration of the constraint graph. To this aim, consider first an OR vertex $v \in \widehat{V}$: we have to exclude that all the three incident edges are oriented outward, that is, in Figure 1(a), at least one agent must be on some $v_e \in V$. To ensure this constraint, in the corresponding OR gadget, we create a new vertex $w \in V$ where we spawn a *guard* agent. Note that vertex $w$, contrarily to all the other vertices belonging to the same gadget, is not shared among other gadgets. The purpose of a guard is to enforce the other agents of the same gadget to assume consistent positions. This can be achieved by imposing that $w$ communicates with some other gadget vertices. More precisely, for each $e \in \widehat{E}$ incident to the OR vertex $v \in \widehat{V}$, we add a communication edge $(w, v_e) \in C$. Hence, in order to connect a guard with the rest of the team, at least one agent must be on some $v_e \in V$, as we wanted. In case of an AND vertex, instead, we have three non-legal combinations of edge orientations to exclude. For each of them, we spawn a guard agent in a new vertex $w$. Then we connect each $w$ in $C$ following the very same reasoning as above. The final result is shown in Figure 1(b). Notice how a legal combination of edge orientations in the OR vertex is mapped to an OR gadget state where all the robots are connected to each other (Figure 1(a)). Looking at the AND reduction (Figure 1(b)), instead, notice how a non-legal combination of orientations is mapped to a state where one of the guards (that residing in $w_2$ in this example) is not connected to the rest of the team.

The other two issues introduced above are related to each other, as the purpose of the transition vertices is to synchronize the agent movements. Indeed, if we can force the agents to occupy at most one transition vertex at a time in the whole multigraph $G$, we obtain the desired result. To this aim, we create an additional structure called *coordination gadget*.

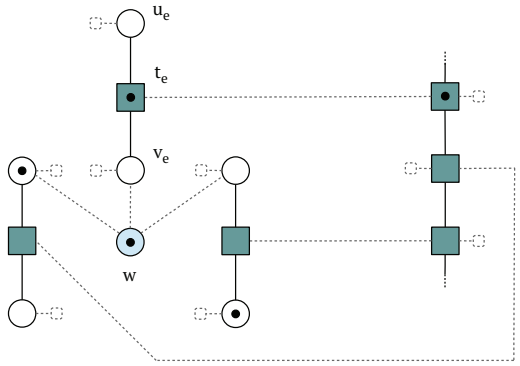



Figure 2: Connections between an OR gadget (left) and (a portion of) the coordination gadget (right).

This gadget contains a set of new vertices $s_1, s_2, \ldots, s_n$ in $V$, where $n = |\widehat{E}|$. These vertices are connected in $G_E$ to form a linear subgraph, that is, $(s_j, s_{j+1}) \in E$, for each $1 \leq j < n$. Each $s_j$ is then connected in $G_C$ to a different transition vertex $t_e$, and to all the non-transition vertices without making any distinction. To complete this synchronization tool, we spawn a coordinator agent in a vertex of the gadget – no matters which one. Figure 2 shows a portion of the coordination gadget and its connections with a generic OR gadget. Each time a vertex in the coordination gadget is occupied by the coordinator, the corresponding transition vertex $t_e$ is "enabled", i.e., connected to all the remaining agents. Due to the presence of a single coordinator, only one transition vertex is enabled at a time and can thus be occupied by an agent. Therefore, among all the AND/OR gadgets, only one agent at a time is able to change its position from $u_e$ to $v_e$ or *vice versa*. The reduction is now complete.

It is now straightforward to check that, if there exists a sequence of moves between $A$ and $B$ on the constraint graph $\widehat{G}$, we can replicate them on the constructed MCPP-D instance by having the coordinator moving to a suitable vertex to enable the corresponding state transition. On the other hand, if there exists a sequence of states leading from the start state $\pi^A$ to the goal state $\pi^B$, there must necessarily exists a sequence of moves leading from $A$ to $B$ in the original constraint graph. In particular, our construction, making use of guard agents, allows to replicate all the possible legal configurations of $\widehat{G}$, while the coordinator ensures that edge reversals can be performed one at a time. This proves that MCPP-D is PSPACE-hard.

## PSPACE Membership

By Savitch's theorem (Savitch 1970), to prove the PSPACE membership of MCPP-D we can focus on proving its membership in NPSPACE. To this aim, we have to show that there exists a non-deterministic algorithm deciding MCPP-D whose amount of stored bits is polynomially bounded.

First of all, note that the system state can be univocally described by encoding the $m$ agent positions on the $|V|$ multigraph vertices. This can be stored in polynomial space. Also, a non-deterministic algorithm deciding MCPP-D would not need to store previously traversed states, as the set of next possible states depends only on the current one. This implies that the state space can be traversed non-deterministically while storing, at each step, a polynomial number of bits. This proves that MCPP-D is in NPSPACE and thus in PSPACE. Therefore, we have proved the following theorem:

**Theorem 1.** *MCPP-D is PSPACE-complete.*

## MCPP Special Cases and Variants

The reader might have noticed that, in the reduction above, the constructed multigraph $G$ is rather strange: $G_E$ is not connected, and not all the vertices connected by an edge in $G_E$ are also connected in $G_C$. These two features are clearly not common, and would most likely be avoided by any reasonable communication-aware environment discretization (such as the one we used in our experiments; see the corresponding section). However, we can easily enforce the connectedness of $G_E$ with the help of some dummy vertices linking the different connected components to each other. With a little more effort, it is also possible to enforce $G_E$ to be a grid graph (either solid or with holes), implying that the hardness of the problem persists even when the physical graph is planar. Similarly, we could also add dummy communication edges to ensure that $E \subseteq C$ and the reduction would still work. (Due to lack of space, the complete construction is not reported. However, it would be easy to verify that this is indeed the case.) Also, note that the problem variant in which at least one of the agents is not allowed to change position (for instance, a fixed base station) is still PSPACE-complete, since guard agents are forced to remain still in our reduction.

Consider now the following variant of MCPP able to model collisions among the agents: two or more agents are neither allowed to occupy the same vertex at the same time, nor to move along the same edge between two subsequent steps. This model generalizes the graph-based Multirobot Path Planning model of (Yu and LaValle 2013b). It is immediate to notice that our proof holds even for this case, since the agents of our reduction are forced to move on non-overlapping portions of $G_E$. For an identical reason, the same MCPP variant in which the agents' goals are not assigned *a priori* (generalizing the model known as permutation-invariant MPP (Yu and LaValle 2013a)) remains PSPACE-complete as well.

# Algorithms

In this section, we tackle the problem of finding MCPP feasible solutions. Being the corresponding decision version PSPACE-complete, the problem is intrinsically hard to solve. Indeed, unless P=PSPACE, any complete algorithm would incur in an exponential worst-case runtime, as the number of possible next states equals the product of all the agents' *local moves*. By local move, we mean either remaining at the current vertex or moving along a graph physical edge in $E$. Local moves will be henceforth referred to as *movements*. However, leaving aside such worst-case scenarios, we present three different algorithms whose performance in realistic problem instances (shown in the next sec-

**Algorithm 1:** Sample-Based

```
1  function sampleBasedSearch (π^s, π^g)
2      p(π^s) ← π^s
3      while not timeout() do
4          closed ← {}
5          π ← π^s
6          while not timeout() do
7              if isOneStepReachable(π, π^g) then
8                  p(π^g) ← π
9                  return p
10             end
11             Ω ← sampleStates(π)
12             π^best ← selectCandidate(Ω)
13             if π^best ∈ closed ∨ π^best = NULL then
14                 break
15             end
16             p(π^best) ← π
17             π ← π^best
18             closed ← closed ∪ {π}
19         end
20     end
21     return Error
22 end
```

tion) may be acceptable practical purposes. Due to space constraints, for each algorithm, only the corresponding main function is presented as pseudocode.

## Sample-Based Algorithm

The first algorithm is dubbed Sample-Based (SB) and is reminescent of the approach proposed in (Rooker and Birk 2007) for the problem of multirobot exploration under continuous connectivity requirements. Algorithm 1 shows the pseudocode. We seek for a feasible solution in the form of a parent function $p(\cdot)$ which is initialized as $p(\pi^s) = \pi^s$. Starting from $\pi^s$, at each iteration, only a subset $\Omega$ of all the next possible states is considered. Specifically, the set $\Omega$ is constructed by generating connected states reached by performing movements randomly chosen with uniform probability. To select the next state, the $\mathrm{selectCandidate}$ subroutine evaluates each state in $\Omega$ through a given heuristic function and returns the state with lowest value. The process is iterated until either the agents get stuck, in the sense that the best state $\pi^{best} \in \Omega$ is already in the $\mathrm{closed}$ set (namely, it has already been visited), or a solution is found (namely, $\pi^g$ is reached). Fixing the number of samples in $\Omega$ at each step allows the algorithm to scale to problem instances of realistic size. As a drawback, the algorithm is not complete and may produce over-complicated paths, as we will see in the next section. The heuristic used to evaluate the goodness of a given state $\pi = \langle p_1, p_2, \ldots, p_m \rangle$ is equal to the sum of the distances between the current agents' positions $\{p_i\}$ and their corresponding goals. This heuristic can be easily computed in a pre-processing phase by means of standard shortest path algorithms and can be retrieved during the search phase in a constant amount of time. Notice that, for non-weighted graphs, the shortest path

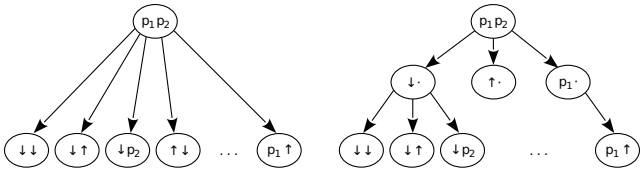



Figure 3: A state decomposition from flat (left) to hierarchical structure (right). In this example the agents can choose between moving up, moving down, or staying still. A dot means that the corresponding agent action has not been set yet.

can be computed efficiently by means of a breath-first search in $\mathcal{O}\left(m(|V| + |E|)\right)$. Finally, to further exploit the random sampling of the algorithm, the above procedure is repeated until a solution is found or a temporal deadline is met.

Even though the SB algorithm may remind the reader of the standard local beam search technique (Russell and Norvig 1995), they are in fact different. Local beam search (or its stochastic variant) aims at pruning the search tree, by sacrificing the completeness in favor of a decrease in the running time. To do that, all the candidates are first evaluated and then a selection process is performed. In the SB case, the problem addressed is different, as the number of candidates is itself exponential. In order to prevent such a local exponential behavior, the SB algorithm deliberately discards (at random) some candidates. The evaluation is then performed only on the restricted set of candidates $\Omega$, whose size can be easily tuned by means of a parameter.

## Randomized Sample-Based Algorithm

The second algorithm, which we call Randomized Sample-Based (RSB), follows the same scheme of the SB approach (refer again to Algorithm 1). The only difference between the two approaches lies in the implementation of the $\mathrm{selectCandidate}$ subroutine. In the RSB algorithm, we do not select the best next state: instead, we select a candidate in $\Omega$ according to a smart randomization scheme. While this may appear as a minor modification, it has a great impact on the theoretical properties of the algorithm. Using a randomized selection, from any state we have a non-zero probability of selecting any possible next state. Hence, any feasible sequence of states has a non-zero probability of being generated. Thus, RSB is probabilistically complete, i.e., given an infinite temporal horizon, it finds a feasible plan, if any exists. Notice that the SB algorithm can ensure the same convergence property only if $\Omega$ always contains a single sample, otherwise the state with the highest heuristic value will never be selected. If $|\Omega| = 1$, the SB approach collapses into a fully random search (in constrast, for $|\Omega| \to \infty$, SB becomes a pure greedy algorithm).

We employ a polynomial randomization scheme, introduced for the first time by (Bresina 1996). In particular, each candidate $\pi \in \Omega$ is ranked according to a heuristic function, and we assign it a weight $\omega(\pi)$ given by:

$$\omega(\pi) = \frac{1}{r(\pi)^{\delta}}, \tag{1}$$

**Algorithm 2:** Depth-First Search

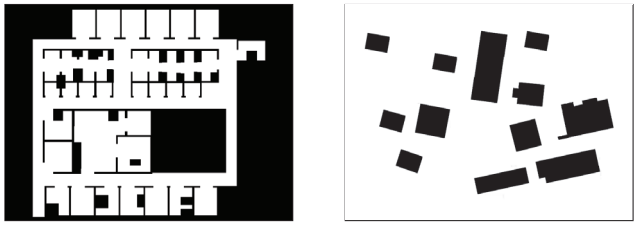
```
1  function depthFirstSearch(π^s, π^g)
2      p(π^s) ← π^s
3      stack.insert(π^g)
4      while stack ≠ {} do
5          π ← stack.back()
6          closed ← closed ∪ {π}
7          if π = π^g then
8              return p
9          end
10         π^n ← findBestChild(π, π^g, closed)
11         if π^n ≠ NULL then
12             stack.insert(π^n)
13             p(π^n) ← π
14         else
15             stack.pop()
16         end
17     end
18     return error
19 end
```

where $r(\pi)$ is the rank order of $\pi$ w.r.t. the other samples in $\Omega$, and $\delta$ is a parameter allowing to tune the randomization "greedyness": higher values of $\delta$ correspond to higher weights given to the most promising movements. The sampling probability of each state in $\Omega$ is then obtained by normalizing the weights.

## Depth-First Search Algorithm

The third algorithm we present is based on a two-level problem decomposition: a main search routine and a set of ancillary routines able to exploit the multiagent structure of the state space. The main routine is summarized by Algorithm 2, and is based on a Depth-First Search (DFS) approach. At each step, the current state $\pi$ is popped from a $\mathrm{stack}$ and added to the $\mathrm{closed}$ set. To move further, the search of the most promising state $\pi^n$ (reachable from $\pi$) is demanded to the $\mathrm{findBestChild}$ subroutine that returns the best next state not already present into the $\mathrm{closed}$ set.

In order to generate the possible next states from which the best one is later selected, we exploit the hierarchical structure reported in the example of Figure 3. In the hierarchy, the agents are thought as moving one at a time, thus forming a tree of partial states reachable by performing sequences of individual movements. The best state reachable from the current one can then be searched without explicitly enumerating all the possible joint movements, whose number would be exponential. More specifically, we move down into the tree by exploiting a priority queue ordered according to a heuristic value, given by the sum of the distances between the current agents' positions and their goal vertices (that is, the same used for SB). Notice that intermediate states encountered while traversing the tree are not necessarily connected: global connectivity is checked only when the search reaches a leaf. As a consequence, the number of computational steps required by a call to $\mathrm{findBestChild}$ becomes linear in the best case. In practice, the computational burden is greatly reduced. The priority queue is kept in memory to be reused when the algorithm backtracks again to the same state in the main routine.



(a) Office (b) Open

Figure 4: The bitmaps of the environments used in the experiments: an office-like indoor environment (a) and an open outdoor environment (b).

A similar approach, decomposing each planning step by considering only single agent moves, is presented in (Standley and Korf 2011) for classical multiagent path planning. However, while (Standley and Korf 2011) considers as next feasible states those corresponding to single agent moves, we exploit such a decomposition to effectively generate the next most-promising state, where, for each of the agents a move has been selected. This expedient could appear less efficient, since we can prune less states, but, in practice, it allows to overcome one of the intrinsic difficulties of the problem; for instance, let us consider a situation where a joint two-robots move is feasible, but the two single-robot moves would violate the connectivity constraint.

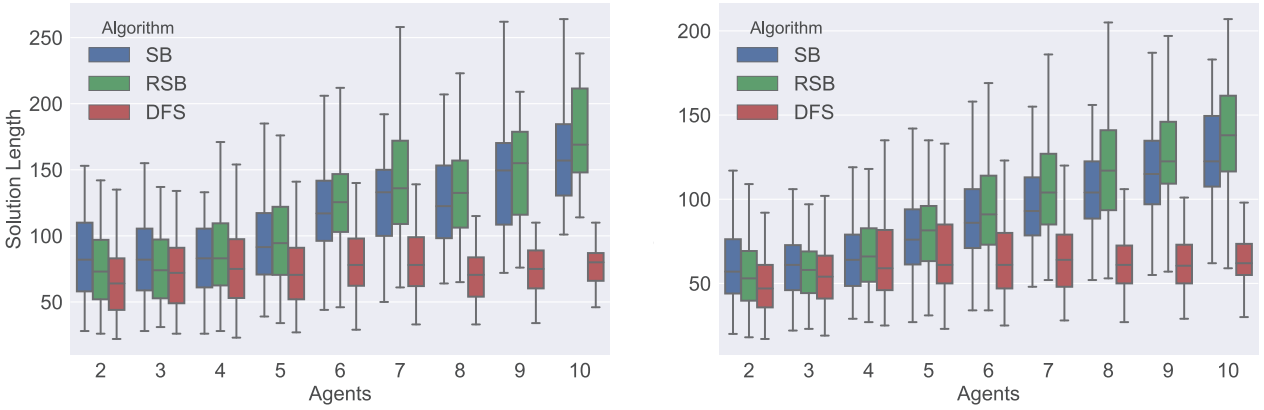
Finally notice that, while the DFS algorithm is complete for the feasibility problem, the SB algorithm can only be used to found a feasible plan and, even asymptotically, it is not able to decide whether a feasible plan exists or not. The RSB algorithm is probabilistically complete, but it cannot provide a result if a feasible plan does not exists.

# Experiments

In this section we evaluate and compare the above presented algorithms. For the experimental campaign, we consider the same two realistic environments employed by (Hollinger and Singh 2012) and here re-proposed in Figure 4. The first environment (Office) articulates in an indoor office-like structure of size $80 \times 60$ meters. The second environment (Open) represents an outdoor area characterized by open spaces and few obstacles, and whose total size is $200 \times 150$ meters. Both environments are discretized as grids of identical squared cells, retrieved from the original bitmaps of $800 \times 600$ pixels. Cells are obtained clustering $11 \times 11$ pixels for Office and $13 \times 13$ pixels for Open. As commonly done in the robotics literature (Rooker and Birk 2007; Hollinger and Singh 2012), we assume a distance-based communication and we consider three different communication ranges: 50 px, 100 px, and 150 px. Two cells are directly connected in $G_C$ if their centers are closer than the communication range. The number of agents varies from 2 to 10. For each of the experimental setting, we randomly generate 50 start-goal states. In order to obtain non-trivial instances, each pair of start-goal positions has to be at least

| #Agents | Office environment | | | | | | | | | Open environment | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Range: 50 px | | | Range: 100 px | | | Range: 150 px | | | Range: 50 px | | | Range: 100 px | | | Range: 150 px | | |
| | SB | RSB | DFS | SB | RSB | DFS | SB | RSB | DFS | SB | RSB | DFS | SB | RSB | DFS | SB | RSB | DFS |
| 2 | 41 | **43** | 42 | 42 | **44** | 43 | 38 | 41 | **48** | **50** | **50** | **50** | 49 | 48 | **50** | **50** | **50** | **50** |
| 3 | 37 | 37 | **41** | 40 | 39 | **44** | 30 | 30 | **38** | **50** | **50** | **50** | 43 | 39 | **50** | **50** | 49 | **50** |
| 4 | 42 | 42 | **45** | **40** | **40** | **40** | 29 | **32** | **32** | 49 | 49 | **50** | 40 | 42 | **44** | **50** | 49 | 49 |
| 5 | 32 | 31 | **33** | 33 | **34** | 31 | 25 | **26** | **26** | **48** | 47 | 44 | 43 | **47** | 41 | **50** | **50** | **50** |
| 6 | 36 | **37** | 32 | **29** | **29** | 25 | **33** | 32 | 26 | **49** | 44 | 35 | **49** | 46 | 35 | **50** | 49 | 46 |
| 7 | **35** | 34 | 27 | 33 | **36** | 28 | 25 | **26** | 18 | **45** | 44 | 32 | **46** | 45 | 29 | **50** | **50** | 46 |
| 8 | **32** | **32** | 21 | **28** | **28** | 18 | **21** | 19 | 14 | **48** | 39 | 26 | **45** | 44 | 24 | **50** | **50** | 47 |
| 9 | **30** | 28 | 24 | **31** | 27 | 19 | **27** | **27** | 18 | **45** | 24 | 20 | **49** | 40 | 27 | **50** | **50** | 47 |
| 10 | **33** | 19 | 16 | **22** | 21 | 14 | **21** | **21** | 10 | **41** | 25 | 22 | **42** | 35 | 21 | **50** | **50** | 41 |

Table 1: Solutions found by the three algorithms over 50 instances for each experimental setting. Bold indicates the best results.




(a) Office environment

(b) Open environment

Figure 5: The distributions of the solution lengths for the two environments, Office (left) and Open (right), represented as a boxplot. The communication range is set to 100 px while the number of agents varies from 2 to 10.

at distance 200 px. Furthermore, in generating a start or a goal state $\pi = \langle p_1, p_2, \ldots, p_m \rangle$, we enforce $p_j$ to be at least at distance $3/4$ of the communication range from $p_{j-1}$. The number of samples in $\Omega$ for SB and RSB is fixed to 100 for each iteration, as in (Rooker and Birk 2007), while the randomization exponent of Equation (1) is set to $\delta = 3$ (from preliminary experiments).

Table 1 reports the number of found solutions within a deadline of 5 minutes by SB, RSB, and DFS. The SB algorithm shows a good performance overall, even if DFS seems to work better for small teams. RSB provides a mixed performance between SB and DFS as the number of agent grows.

We can note that, increasing the communication range, the problem instances become harder in the Office environment, as the number of solutions found decreases. This can be due to the structure of the start and goal states: the greater the communication range, the higher the probability that start and/or goal states contain positions lying in different rooms. This makes feasible plans less straightforward to obtain. Conversely, in the Open environment, a larger communication range seems to simplify the problem, as the obstacles only marginally influence the paths.

The total fraction of instances solved by only one algorithm is $9.0\%$ in Office and $6.4\%$ in Open. In the former percentage, SB, RSB, and DFS contribute by $32.2\%$, $24.0\%$, and $43.8\%$, respectively; in the latter, the percentage contribution is $74.7\%$, $10.3\%$ and $14.9\%$. As expected, the SB algorithm dominates in the Open environment. However, in a structured environment such as Office, the performance of DFS is remarkable.

From a practical standpoint, we are also interested in assessing the quality of the solutions. To this aim, we consider the number of states composing each solution, and we call it *solution length*. Notice that this quantity captures the total time needed to complete the task, as customarily considered in the multiagent path planning literature (Hollinger and Singh 2012; Banfi, Basilico, and Amigoni 2017).

Figure 5 summarizes the distributions of the solution lengths. The instances considered are those solved by all the algorithms, where the communication range has been set to 100 px – but similar results hold for the other cases. As expected, DFS produces better paths, in general. Also, notice how the difference in solution length for the algorithms becomes larger and larger from 5 agents.

If the computational resources are not a concern, all the algorithms can be launched in parallel. This would increase the probability of finding a plan. Furthermore, a solution found by DFS should generally be preferred given its lower length, while an SB or RSB plan can be chosen if DFS is not able to produce any feasible solution within the given deadline. If none of the algorithms finds a solution, the instance could be simplified by either introducing intermediate goals, or considering a coarser discretization.

## Conclusions

Solving the Multiagent Connected Path Planning problem (MCPP) is of primary interest in different multiagent information-gathering missions subject to communication constraints. In this paper, we have shown that even deciding if a given instance of MCPP admits a feasible plan is PSPACE-complete. This means that – unless NP=PSPACE – there might be problem instances not even admitting the existence of "compact" feasibility certificates (i.e., of size polynomial w.r.t. that of the input). However, motivated by the need of solving MCPP in realistic scenarios, we presented three algorithms adopting different search paradigms, and we empirically showed that they may perform well in practice. Our experiments also showed that the hardness of MCPP stems from the difficulty of finding good heuristics for expanding an exponential number of neighbor states (even if a neighbor state is connected, pursuing the search along that direction may not reach the goal).

An interesting direction for future works is related to a more precise identification of the complexity profile of MCPP. Indeed, as we have shown, the problem is PSPACE-complete in the general case, even under reasonable assumptions about the structure of $G_E$ and $G_C$. However, it might be interesting to study if PSPACE-completeness persists when restricting the problem to multigraphs where $G_E$ is a grid graph and $G_C$ encodes a limited-range communication model (like in the instances considered in our experiments). From a practical standpoint, it would be interesting to try to improve the randomization strategy employed by our RSB algorithm to preserve its theoretical convergence property and improve performance.

## References


Banfi, J.; Basilico, N.; and Amigoni, F. 2017. Intractability of time-optimal multirobot path planning on 2D grid graphs with holes. *IEEE RA-L* 2(4):1941–1947.

Bhattacharya, S.; Likhachev, M.; and Kumar, V. 2010. Multi-agent path planning with multiple tasks and distance constraints. In *Proc. ICRA*, 953–959.

Bresina, J. 1996. Heuristic-biased stochastic sampling. In *AAAI/IAAI*, volume 1, 271–278.

Feo Flushing, E.; Kudelski, M.; Gambardella, L.; and Di Caro, G. 2013. Connectivity-aware planning of search and rescue missions. In *Proc. SSRR*, 1–8.

Hearn, R., and Demaine, E. 2005. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theor Comput Sci* 343(1-2):72–96.

Hollinger, G., and Singh, S. 2012. Multirobot coordination with periodic connectivity: Theory and experiments. *IEEE T Robot* 28(4):967–973.

Hollinger, G. 2010. *Search in the physical world.* Ph.D. Dissertation, Carnegie Mellon University.

Rooker, M., and Birk, A. 2007. Multi-robot exploration under the constraints of wireless networking. *Control Eng Pract* 15(4):435–445.

Russell, S., and Norvig, P. 1995. Artificial intelligence: A modern approach. Pearson.

Sabattini, L.; Secchi, C.; Chopra, N.; and Gasparri, A. 2013. Distributed control of multirobot systems with global connectivity maintenance. *IEEE T Robot* 29(5):1326–1332.

Savitch, W. 1970. Relationships between nondeterministic and deterministic tape complexities. *J Comput Syst Sci* 4(2):177–192.

Standley, T., and Korf, R. 2011. Complete algorithms for cooperative pathfinding problems. In *Proc. IJCAI*, 668–673.

Stump, E.; Michal, N.; Kumar, V.; and Isler, V. 2011. Visibility-based deployment of robot formations for communication maintenance. In *Proc. ICRA*, 4498–4505.

Yu, J., and LaValle, S. 2013a. Multi-agent path planning and network flow. In *Proc. WAFR*, 157–173.

Yu, J., and LaValle, S. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *Proc. AAAI*, 1443–1449.

Yu, J., and Rus, D. 2015. Pebble motion on graphs with rotations: Efficient feasibility tests and planning. In *Proc. WAFR*, 729–746.

Yu, J. 2016. Intractability of optimal multirobot path planning on planar graphs. *IEEE RA-L* 1(1):33–40.

Zavlanos, M.; Egerstedt, M.; and Pappas, G. 2011. Graph-theoretic connectivity control of mobile robot networks. *Proc. IEEE* 99(9):1525–1540.