Project number IST-25582

CGL

Computational Geometric Learning

k-Color Multi-Robot Motion Planning

STREP

Information Society Technologies

Period covered:	November 1, 2011–October 31, 2012
Date of preparation:	October 21, 2012
Date of revision:	October 21, 2012
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

k-Color Multi-Robot Motion Planning*

Kiril Solovey and Dan Halperin

Abstract: We present a simple and natural extension of the *multi-robot motion planning* problem where the robots are partitioned into groups (colors), such that in each group the robots are interchangeable. Every robot is no longer required to move to a specific target, but rather to some target placement that is assigned to its group. We call this problem *k-color multi-robot motion planning* and provide a sampling-based algorithm specifically designed for solving it. At the heart of the algorithm is a novel technique where the k-color problem is reduced to several discrete multi-robot motion planning problems. These reductions amplify basic samples into massive collections of free placements and paths for the robots. We demonstrate the performance of the algorithm by an implementation for the case of disc robots in the plane and show that it successfully and efficiently copes with a variety of challenging scenarios, involving many robots, while a simplified version of this algorithm, that can be viewed as an extension of a prevalent sampling-based algorithm for the k-color case, fails even on simple scenarios. Interestingly, our algorithm outperforms a state-of-the-art implementation for the standard multi-robot problem, in which each robot has a distinct color.

1 Introduction

Motion planning is a fundamental problem in robotics and has applications in different fields such as the study of protein folding, computer graph-

School of Computer Science, Tel-Aviv University, {kirilsol,danha}@post.tau.ac.il

* This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827 (CGL—Computational Geometry Learning), by the Israel Science Foundation (grant no. 1102/11), and by the Hermann Minkowski–Minerva Center for Geometry at Tel Aviv University.

ics, computer-aided design and manufacturing (CAD/CAM), and computer games.

The problem of motion planning, in its most basic form, is to find a collision-free path for a robot from start to goal placements while moving in an environment cluttered with obstacles.

An obvious extension of this problem is *multi-robot motion planning*, where several robots share a workspace and have to avoid collision with obstacles as well as with fellow robots. In many situations it is natural to assume that some robots are identical, in form and in functionality, and therefore are indistinguishable. In this setting every target position should be occupied by some robot of a kind (and not necessarily by a specific robot).

We consider the problem of *k-color multi-robot motion planning*—a simple and natural extension of the multi-robot problem where the robots are partitioned into k groups (colors) such that within each group the robots are interchangeable. Every such group has a set of target positions, of size equal to the number of robots in that group. Every robot is no longer required to move to a specific target, but rather to some target position that is assigned to its group. However, we still require that all the target positions will be covered by the end of the motion of the robots. We term the special case where $k = 1$ the *unlabeled multi-robot motion planning* problem.

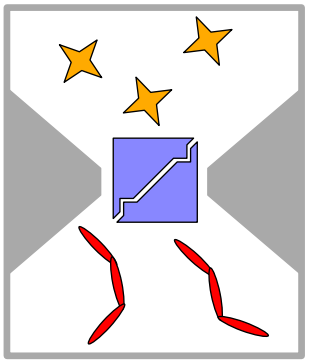


Fig. 1: An example of a 3-color scenario where three different groups of robots occupy the same workspace. The star-shaped (orange) robots are required to exchange "rooms" with the snake (red) robots while the two puzzle-like (purple) robots should return to their start positions in the end of the motion. Obstacles are drawn in gray.

As an example consider a fleet of mobile robots operating in a factory that are given the task of cleaning a set of specific locations. The robots are indistinguishable from one another, and therefore any robot can be assigned to any location. Now assume that in addition to the mobile robots, another class of maintenance robots is employed by the factory; again, we consider all the maintenance robots to be of the same kind and interchangeable for the given task. This turns the unlabeled problem into a k-color problem, where $k = 2$ in this case.

From now on we will refer to the classic multi-robot motion planning problem as *fully-colored*, as it is a special case of the k-color problem where k is equal to the number of robots and every group is of size one.

1.1 Previous Work

Throughout this paper we will assume some familiarity with the basic terms in the area of motion planning. For more background on motion planning, see, e.g., [6, 16].

The first efforts in motion planning in general, and the multi-robot case in particular, were aimed towards the design of *complete* algorithms, guaranteed to find a solution when one exists or report that none exists otherwise. Schwartz and Sharir were the first to give [22] a complete algorithm for a multi-robot problem, specifically dealing with the case of coordinating disc robots in the plane. The running time of their algorithm is exponential in the number of robots. A work by Hopcroft et al. [10] presented soon after suggested that in some cases the exponential running time may be unavoidable, showing that even the relatively simple setting of rectangular robots bound in a rectangular region is PSPACE-hard in the number of robots.

The hardness of the multi-robot problem involving a large number of robots can be attributed to its high number of *degrees of freedom* (or *dofs*)—the sum of the dofs of the individual robots. Some efforts were made in the direction of reducing the effective number of dofs. Aronov et al. [2] showed that for systems of two or three robots a path can be constructed, if one exists, where the robots move while maintaining contact, thus reducing the number of dofs by one or two, depending on the number of robots. van den Berg et al. [4] proposed a general scheme for decomposing a multi-robot problem into a sequence of subproblems, each consists of a subset of robots, where every subproblem can be solved separately and the results can be combined into a solution for the original problem. This method reduces the number of dofs that need to be treated simultaneously from the number of dofs of the entire problem to the number of dofs of the largest subproblem.

An opposite approach to the complete planners is the *decoupled* approach, trading completeness with efficiency. Decoupled algorithms solve separate subproblems (usually for individual robots) and combine the individual solutions into a global solution. Although this approach can be efficient in some cases, it does not guarantee finding a solution if one exists and usually works only for a restricted set of problems. An example of such an algorithm can be found in the work of van den Berg and Overmars [3] where every robot is given a priority and for each robot, the motion path is constructed to avoid collision with both static obstacles and lower-priority robots that are considered as moving obstacles. In other works, as in Leroy et al. [17], individual paths are computed and velocity tuning is performed to avoid collision between robots.

In recent years, the *sampling-based* approach for motion-planning problems has become increasingly popular due to its efficiency, simplicity and the fact that it is applicable to a wide range of problems. Unlike the complete planners that explicitly build the *configuration space* of a given problem, the state of all possible configurations of a robot, sampling-based algorithms con-

struct an implicit representation of a robot configuration space by sampling this space for valid robot placements and connecting nearby samples. The connections between samples form a *roadmap* whose vertices describe valid placements for the robot and the edges represent valid paths from one placement to the other. Due to the implicit representation of the configuration space and their simplicity, sampling-based algorithms tend to be much faster than complete planners in practice, and are applicable to problems with a large number of dofs such as the multi-robot problem. Although these algorithms are not complete, many of them are *probabilistically complete*, that is, they are guaranteed to find a solution, if one exists, given sufficient amount of time. Examples of such algorithms are the PRM algorithm [11] by Kavraki et al. and the RRT algorithm [15] by Kuffner and LaValle. Such algorithms can be easily extended to the multi-robot case by considering the fleet of robots as one large composite robot [21]. Several tailor-made sampling-based algorithms have been proposed for the multi-robot case [9, 24]. For more information on sampling-based algorithms see, e.g., [16].

An abstract form of the multi-robot motion planning problem is the *pebble motion on graphs problem* [14]. This is a general case of the famous *15-puzzle* [18] where pebbles occupying distinct vertices of a given graph are moved from one set of vertices to another, where the pebbles are bound to move on the edges of the graph. In [5] an unlabeled version of the pebble problem is discussed, as well as other variants, such as a grid topology of the graph. In [8] the feasibility of a k-color variant of the pebble problem on a general graphs is discussed. We also mention the work [19] where an algorithm is given for a fairly general pebble problem. A recent work by Wagner et al. [26] combines their technique for multi-agent pathfinding in discrete environments [25] with an implicit representation of the roadmap, presented by Švestka and Overmars [24], to yield an efficient algorithm for the fully-colored multi-robot motion planning problem.

1.2 Contribution

In this paper we present a sampling-based algorithm for the k-color problem (for any k). This algorithm is aimed to solve the most general cases of this problem and does not make any assumptions regarding the workspace or the structure of the robots.

Our algorithm for the k-color problem—the KPUMP algorithm—reduces the k-color problem to several discrete pebble problems. Specifically, a sample generated by KPUMP represents a local k-color problem that is embedded in a variant of the pebble motion problem. Those pebble problems are constructed in a manner that enables the algorithm to transform movements of pebbles into valid motions of the robots. This allows KPUMP to generate a wide range of motions and placements for the robots with minimal

investigation of the configuration space, thus reducing the dependence of the algorithm on costly geometric tools such as the collision detector.

As reflected in the experiments reported below for the case of disc robots in the plane, KPUMP proves to be efficient, even on challenging scenes, and is able to solve problems involving a large number of robots using a modest number of samples. Interestingly, it performs well even on inputs of the standard (fully-colored) multi-robot problem.

This algorithm is simple to implement and does not require special geometric components beyond single-robot local planners and single-robot collision detectors. We compare the performance of our algorithm with a simplified version of KPUMP that can be considered as a variant of the PRM algorithm for the same problem. We note that the the latter performs much slower than KPUMP and fails to solve even problems that are considered to be simple for KPUMP. Moreover, concentrating on the fully-colored case, KPUMP outperforms a state-of-the-art implementation of the PRM algorithm. Our discussion will mainly focus on UPUMP—an algorithm for the unlabeled case, since its extension for the k-color case, namely KPUMP, is almost straightforward. The experiments though will demonstrate the power of KPUMP for various values of k.

The organization of the paper is as follows. In Section 2 we give formal definitions of the unlabeled and k-color problems. In Section 3 we present a variant of the pebble problem and discuss its properties which will be exploited by our algorithms. In Section 4 we present UPUMP. In the following section (Section 5) we describe a subroutine that is used by UPUMP, which we call the *connection generator*. In Section 6 we describe the changes that are necessary to transform UPUMP into KPUMP. We present experimental results for the case of disc robots moving among polygonal obstacles in the plane in Section 7 and discuss certain properties of our techniques in Section 8, as well as further work.

2 Preliminaries and Terminology

Let r be a robot operating in the workspace W. We denote by $\mathcal{F}(r)$ the *free space* of a robot r—the collection of all collision-free *single-robot configurations*.[2] Given $s, t \in \mathcal{F}(r)$, a *path* for r from s to t is a continuous function $\pi : [0, 1] \to \mathcal{F}(r)$, such that $\pi(0) = s, \pi(1) = t$.

Unlabeled Multi-Robot Motion Planning. We say that two robots r, r' are *geometrically identical* if $\mathcal{F}(r) = \mathcal{F}(r')$ for the same workspace W. Let $R = \{r_1, \ldots, r_m\}$ be a set of m geometrically identical robots, operating in a workspace W. We may use $\mathcal{F}$ to denote $\mathcal{F}(r_i)$ for any $1 \leq i \leq m$. Let

[2] We assume that $\mathcal{F}(r)$ is an open set. This is not critical in the algorithms below as we assume that the robot never moves in contact with the obstacles.

$C = \{c_1, \ldots, c_m | c_i \in \mathcal{F}\}$ be a set of m single-robot configurations. C is a *configuration* if for every $c, c' \in C$, with $c \neq c'$, the robots $r, r' \in R$, placed in c, c', do not collide. Notice that we reserve the unqualified term *configuration* to refer to a set of m collision-free single-robot configurations. Other types of configurations will be qualified: single-robot configurations and pumped configurations.

Given two configurations $S = \{s_1, \ldots, s_m\}, T = \{t_1, \ldots, t_m\}$, named *start* and *target*, respectively, we define $\mathcal{U} = (R, S, T)$ as the *unlabeled problem*, which is shorthand for the *unlabeled multi-robot motion planning problem*. Our goal is to find an *unlabeled path* $\pi_{\mathcal{U}}$, defined as follows. Firstly, $\pi_{\mathcal{U}}$ is a collection of m paths $\{\pi_1, \ldots, \pi_m\}$ such that for every i, π_i is a collision-free path for the robot r_i from s_i to *some* $t \in T$. Secondly, the robots have to remain collision-free while moving on the respective paths, i.e., for every $\theta \in [0, 1]$, $\pi_{\mathcal{U}}(\theta) = \{\pi_1(\theta), \ldots, \pi_m(\theta)\}$ is a configuration. Notice that this also implies that $\pi_{\mathcal{U}}(1)$ is some permutation of T.

Throughout this paper, we use the notation $r(c) \subset \mathcal{C}$, for $c \in \mathcal{F}$, to represent the portion of the configuration space covered by a robot $r \in R$ placed in the single-robot configuration c. Note that two robots from R collide, when placed in $c, c' \in \mathcal{F}$, if $r(c) \cap r(c') \neq \emptyset$.

k-Color Multi-Robot Motion Planning. The k-color problem $\mathcal{L}$ is defined by the set of unlabeled problems $\{\mathcal{U}_1, \ldots, \mathcal{U}_k\}$, where $\mathcal{U}_i = (R_i, S_i, T_i)$ and $|R_i| = m_i$. The definition of the solution to this problem, namely a *k-color path*, immediately follows. A special case of this problem, usually named simply *multi-robot motion planning*, is a k-color problem where for every $\mathcal{U}_i$ it holds that $|R_i| = 1$. In our context we call this special case *fully-colored*.

3 The Pebble Motion Problem

In preparation for the algorithm presented in the next section, we discuss a variant of the problem of *pebble motion on graphs*. This problem is a discretization of the unlabeled problem. This discretization is defined in a manner that will allow us to transform local unlabeled problems into pebble problems such that a movement of the pebbles can be transformed back into valid robot motions. We explain below where our formulation is different from the standard presentation of the pebble-motion problem.

3.1 Formal Definition

A pebble problem [14] $\mathcal{P}(G, S, T, m)$ is defined by an undirected graph $G = (V, E)$, and two sets of vertices $S, T \subseteq V$, where $|S| = |T| = m$. A *pebble placement* is an ordered set of m distinct vertices of V. Initially, m identical

pebbles $\tau_1, \ldots, \tau_m$ are placed in S. We wish to find a chain of placements $\pi^* = P_1, \ldots, P_\ell$, called a *pebble path*, which obeys the following set of rules. Firstly, we demand that $P_1 = S$. Secondly, for every two consecutive placements $P = \{p_1, \ldots, p_m\}, P' = \{p'_1, \ldots, p'_m\}$ and every $1 \leq i \leq m$ it holds that $(p_i, p'_i) \in E$ or $p_i = p'_i$, i.e., the pebble τ_i is allowed to stay in its current vertex or move to a neighboring vertex in the graph.

Next we depart from the problem definition in [14]. We demand that P_ℓ is some permutation of the elements of T. (The original formulation [14] specified which pebble will reside on which specific vertex of T.) We do, however, impose an additional requirement—the *separation rule*—which requires that the pebbles will move separately, i.e., for every two consecutive placements P, P', as defined above, exactly one pebble τ_i makes a move on an edge, while the other pebbles remain stationary. More formally, there exists $1 \leq i \leq m$ such that $(p_i, p'_i) \in E$ and for every $j \neq i$ it holds that $p_j = p'_j$. The reason for this restriction will become clear later on.

3.2 Solvability

We provide a simple test to identify whether a given pebble problem has a solution. We start with a pair of basic definitions.

Definition 1. Let V' be a pebble placement of $\mathcal{P}(G, S, T, m)$ and let $\{G_1, \ldots, G_h\}$ be the set of maximal connected subgraphs of G, where $G_i = (V_i, E_i)$. The *signature* of V' is defined as $\mathrm{sig}(G, V') = \{|V' \cap V_i|\}_{i=1}^{h}$.

Namely, the signature of a placement is the number of pebbles in every connected component. Using this definition we define an equivalence relation between placements.

Definition 2. Let V', V'' be two placements of $\mathcal{P}(G, S, T, m)$. We say that the two placements are equivalent if $\mathrm{sig}(G, V') = \mathrm{sig}(G, V'')$ and denote this property by $V' \equiv V''$.

We note that this equivalence relation is defined between placements of the same graph. The variant of the pebble problem used in this paper possesses the following property, which states that there exists a pebble path between every two equivalent pebble placements. This property plays a cental role in the design of the UPUMP algorithm, presented in the next sections.

Lemma 1. *For every pebble problem $\mathcal{P}(G, S, T, m)$ such that $S \equiv T$, there exists a pebble path from S to T.*

Proof. This lemma is a generalization of [13, Section 3, first Lemma] where an algorithm for the case of a connected graph is given. We mention that this algorithm constructs a spanning tree of G and restricts the movements of the pebbles to the edges of the tree.

From now on, we will refer to the algorithm that solves the pebble problem as *pebble solver*, which given a pebble problem returns a pebble path.

4 Algorithm for the Unlabeled Case: Pumped Configurations

In this section we present our main contribution — a sampling-based algorithm for the unlabeled problem. The algorithm, UPUMP, generates a collection of geometrically-embedded graphs. These are called *pebble graphs* and enable to map valid movements of pebbles from one pebble placement to the other on these graphs, into motions of robots between configurations. The vertices of such pebble graphs are single-robot configurations while the edges represent single-robot paths. We generate a pebble graph by sampling a set of single-robot configurations, called *pumped configurations*, of size larger than the actual number of robots, to seemingly accommodate an increased number of robots.

This technique makes use of the fact that our problem does not involve one complex robot, but rather a collection of robots operating in the same configuration space. This is in contrast with a popular sampling-based technique that considers the group of robots as one *composite robot*. In our opinion, the latter suffers from an acute disadvantage compared to our technique. We will demonstrate this claim experimentally and discuss the benefits of UPUMP and KPUMP in depth later on.

After discussing the construction of pebble graphs and exploring their various properties we show that they can be connected to generate more complex paths where the robots not only move within a single pebble graph but also between different pebble graphs on collision-free paths. We conclude this section with a description of the sampling-based algorithm.

4.1 Construction of Pebble Graphs

We now define more formally some of the aforementioned structures. Recall that a configuration is a collection of m single-robot configurations, where m is the actual number of robots, i.e., $|R|= m$, for which the m robots are collision-free.

Definition 3. Let $V = \{v_1, \ldots, v_n\}$ for $n \geq m$ be a set of single-robot configurations such that for every $v \in V$ it holds that $v \in \mathcal{F}$, where $\mathcal{F} = \mathcal{F}(r)$ for some $r \in R$. V is a *pumped configuration* if every $C \subseteq V$, such that $|C|= m$, is a configuration.

This implies that a pumped configuration can accommodate an increased number of robots, n to be exact. A possible implementation of a procedure that generates a pumped configuration is given in Algorithm 1.

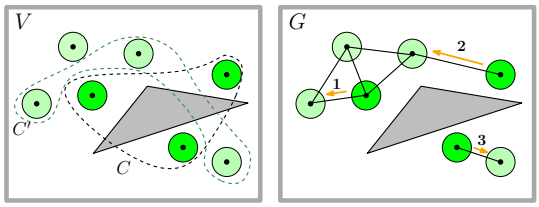
Given a pumped configuration V we construct the graph $G = (V, E)$ where the edges represent paths in $\mathcal{F}$ for individual robots. We call it a *pebble graph*, and view it as embedded in the free configuration space. To generate the edges of G, and the respective paths, we utilize the *edge planner* mechanism that is described below. This, in turn, relies on the *local planner* component, that traditionally attempts to connect two single-robot configurations with a straight-line path, although a more sophisticated technique can be used.

Given $v, v' \in V$ such that V is a pumped configuration and $v \neq v'$, and let π be a path for $r \in R$ from v to v' that was generated by the local planner. If for every $u \in V$, where $u \neq v, v'$, the robot r, while moving on π, does not collide with a (geometrically identical) robot placed in u, then the *edge planner* returns π. Otherwise, it reports failure.

A procedure for the creation of a pebble graph is described in Algorithm 2. The edge planner is applied on every pair $v \neq v'$ in V. Upon successful generation of a path $\pi_{v,v'}$ the edge (v, v') is added to G. An example of a pumped configuration, as well as its underlying graph, are given in Figure 2.

4.2 Properties of Pebble Graphs

We now discuss the various properties of this special graph. We first note that every configuration $C \subset V$ is also a pebble placement for some pebble problem that is defined on G (and vice versa). A less obvious property of the pebble graph G, which is described in the following proposition, allows us to transform pebble paths into robot paths.



(a) (b)

Fig. 2: (a) Pumped configuration V with $m = 3, n = 7$, for the problem of disc robots in the plane. C, C' are two configurations such that $C, C' \subset V$. (b) The pebble graph G is induced by V using an edge planner that tries to connect pairs of single-robot configurations with a straight-line path. In addition, a path induced by a pebble path, from C to C', is described, where the arrows describe the movements of the robots from one single-robot configuration to its neighbor, and the numbers indicate the order in which those movements occur.

Proposition 1. *Let $G = (V, E)$ be a pebble graph and let $C, C' \subset V$ be two configurations such that $C \equiv C'$. Then there exists a path $\pi_{\mathcal{U}'}$ for $\mathcal{U}' = (C, C')$.*

Proof. By Lemma 1 there is a pebble path π^* for the pebble problem $\mathcal{P}(G, C, C', m)$. We transform the movements of the pebbles into π^* to a valid motion of the robots in the following manner. A movement of the pebble τ_i on the edge $(v, v') \in E$ is transformed to the the motion of the robot r_i along the path $\pi_{v,v'}$. Notice that a collision between a robot and an obstacle cannot occur since the path was generated by the edge planner. Additionally, a moving robot cannot collide with another "stationary" robot that resides in some other vertex $u \in V$. Finally, a collision between two moving robots cannot occur since the pebble path π^* must respect the separation rule (Section 3), which states that at most one pebble is allowed to move at a given time. □

4.3 Connecting Pebble Graphs

Proposition 1 implies that certain unlabeled problems can be solved using a single pebble graph. However, this statement does not hold for many other instances of the unlabeled problem. As an example, consider an unlabeled problem $\mathcal{U} = (S, T)$ in which there exists at least one pair $s \in S, t \in T, s \neq t$,

Algorithm 1 PUMPED_CONFIGURATION(n)

```
1: V ← ∅
2: while |V| ≠ n do
3:     v ← RANDOM_SAMPLE()
4:     valid ← TRUE
5:     for all v' ∈ V, v ≠ v' do
6:         if r(v) ∩ r(v') ≠ ∅ then
7:             valid ← FALSE
8:     if valid then
9:         V ← V ∪ {v}
10: return V
```

Algorithm 2 PEBBLE_GRAPH(n)

```
1: V ← PUMPED_CONFIGURATION(n)
2: E ← ∅
3: for all v, v' ∈ V, v ≠ v' do
4:     π_{v,v'} ← EDGE_PLANNER(V, v, v')
5:     if π_{v,v'} ≠ ⊥ then
6:         E ← E ∪ {(v, v')}
7: return G = (V, E)
```

such that a robot $r \in R$ placed in s overlaps with another robot $r' \in R$ placed in t. Thus, s, t cannot be in the same pumped configuration.

Fortunately, we can combine several graphs in order to find paths for more general unlabeled problems. For instance, robots may move from a pebble graph $G_S = (V, E)$ where $S \subset V$, through several other pebble graphs until they will finally reach $G_T = (V', E')$ where $T \subset V'$.

We first show that given two pebble graphs and an unlabeled path connecting two configurations, one from every graph, the robots can move from the first pebble graph to the second. This path serves as a "bridge" between the two graphs and connects not only the two configurations but many other configurations from the two graphs as well. Before describing a mechanism to generate such paths we provide a concrete description of the property discussed here in the form of the following lemma. We omit its proof, which is straightforward.

Lemma 2. *Let $C \subset V, C' \subset V'$ be two configurations of the pebble graphs $G = (V, E), G' = (V', E')$ and let $\pi_{C,C'}$ be a path for the unlabeled problem $\mathcal{U}' = (C, C')$. In addition, let D, D' be two configurations such that $D \subset V, D' \subset V'$ and $D \equiv C, D' \equiv C'$. Then there exists a path $\pi_{\mathcal{U}''}$ for $\mathcal{U}'' = (D, D')$.*

Paths similar to $\pi_{C,C'}$ described above are generated using the following component which generalizes the component *local planner* used in standard sampling-based algorithms. We postpone a detailed description of this component to Section 5.

Given two pumped configurations V, V' and an integer q, the *connection generator* returns q unlabeled paths such that every returned path $\pi_{C,C'}$ is a solution for some unlabeled problem $\mathcal{U}' = (C, C')$ where C, C' are configurations such that $C \subset V, C' \subset V'$.

By Lemma 2, a single connection implicitly connects a collection of configurations with a specific signature from the first graph with a similar collection in the second graph. We require from the connection generator to create several such connections in order to connect a variety of signatures between the two graphs.

4.4 Description of UPUMP

Next, we extend Lemma 2 to describe still more complex paths. The UPUMP algorithm has a preprocessing phase and a query phase. In the first phase it samples a collection of pebble graphs and connects them using the connection generator. Those connections represent edges in a roadmap $\mathcal{H}$ whose vertices are configurations from the different pebble graphs. Additional edges, that represent paths between configurations within the same pebble graph, are

added to $\mathcal{H}$ afterwards. In the query phase, given start and target configurations S, T, UPUMP generates two pebble graphs that contain them. These two graphs are connected to other previously sampled pebble graphs. We give a more formal description below, along with the description of the parameters used by UPUMP.

Parameters. g is the number of sampled pebble graphs; n represents the size of a sampled pumped configuration; q is the maximal number of connections between two pebble graphs.

Preprocessing (Algorithm 3). UPUMP samples a collection of g pebble graphs $\mathcal{G}$ (line 3). For every pair of sampled pebble graphs $G = (V, E), G' = (V', E')$ we apply the CONNECT procedure (line 6) that generates several connections between the two pebble graphs, and updates the roadmap accordingly. Then, we add edges to $\mathcal{H}$ that represent connections that follow from Proposition 1 (line 8). We remind that two configurations are equivalent only if they were taken from the same pebble graph, and their signatures are identical. We draw an edge between them but do not generate the respective paths at this point, as only some of them will eventually participate in a path returned in the query phase (an economical "lazy" approach).

Connect (Algorithm 4). This is an auxiliary method that uses the connection generator component to connect two given pebble graphs (line 1). For every path $\pi_{C,C'}$ returned by the connection generator, where C, C' are configurations of G, G', respectively, C and C' are added as vertices to the roadmap $\mathcal{H}$ together with an edge between them. To this edge the information $\pi_{C,C'}$ is attached.

Query (Algorithm 5). In this phase, UPUMP is given the start and target configurations. As S, T can be considered as pumped configurations (containing m single-robot configurations) we generate the respective pebble graphs G_S, G_T (line 1). We then connect G_S, G_T to previously sampled pebble graphs using the connection generator and add relevant vertices and edges to $\mathcal{H}$ (CONNECT procedure described in Algorithm 4). Finally, if S, T are connected in $\mathcal{H}$ a path retrieval is carried out.

Path Retrieval (Algorithm 6). Using a graph search algorithm, a path is found between S and T in $\mathcal{H}$ (line 2). Then, it is transformed into a solution to the unlabeled problem $\mathcal{U} = (R, S, T)$. If two consecutive configurations C_{i-1}, C_i on the path are equivalent, then the respective pebble path is produced (line 6) and converted to a path for the unlabeled problem $\mathcal{U} = (R, C_{i-1}, C_i)$, following the process described in Proposition 1. If on the other hand $C_{i-1} \not\equiv C_i$, then the path π_{C_{i-1},C_i}, that was generated by the connection generator, is used.

5 Algorithm for the Connection Generator

We describe an algorithm for the connection generator component (CONGEN), used by UPUMP. Recall that the connection generator is given two pumped configurations V, V' and an integer q that represents the number of desired connections. Throughout this section we will use the *local planner* mechanism, that was used in the implementation of the *edge planner* (Section 4.1). Recall that given two single-robot configurations $v, v' \in \mathcal{F}$ the local planner attempts to construct a path $\pi_{v,v'}$ for a robot $r \in R$ from v to v'.

The algorithm transforms the problem of finding paths between pumped configurations into the problem of finding an *independent set* in an undirected graph. We generate the set of pairs $\mathcal{L} = \{(v, v') | v \in V, v' \in V',\ \pi_{v,v'} \neq \perp\}$. Namely, these are pairs of elements from V, V' for which the local planner successfully generated a path. We say that two pairs $(v, v'), (u, u') \in \mathcal{L}$ *interfere* if there exists $\theta \in [0, 1]$ such that a robot $r \in R$ placed in $\pi_{v,v'}(\theta)$ collides with another robot $r' \in R$ placed in $\pi_{u,u'}(\theta)$. Notice that every two pairs $(v, v'), (u, u') \in \mathcal{L}$, such that $v = u$ or $v' = u'$, interfere by definition. We construct the *interference graph* $\mathcal{I}$ whose vertices are the elements of $\mathcal{L}$, i.e., every vertex of $\mathcal{I}$ represents a path. We connect a pair of vertices of $\mathcal{I}$ by an edge if they interfere.

Notice that by definition, every independent set of size m of the vertices of $\mathcal{I}$ represents a collection of m non-colliding single-robot paths. Note that the problem of finding an independent set is known to be NP-Hard.

We use the following heuristic to find several independent sets: vertices of the graph are examined one by one, when the order is determined by a random permutation of the vertices. A new vertex is added to the set only if it is not connected to other vertices that are already in the set.

Algorithm 3 PREPROCESS(g, q, n)

```
V ← ∅; E ← ∅; H = (V, E)
G ← ∅
for i = 1 → g do
    G ← PEBBLE_GRAPH(n)
    G ← G ∪ {G}
for all G, G' ∈ G do
    CONNECT(G, G', q)
for all C, C' ∈ V where C ≡ C' do
    E ← E ∪ {(C, C')}
```

Algorithm 4 CONNECT$(G = (V, E), G' = (V', E'), \mathcal{H} = (\mathcal{V}, \mathcal{E}), q)$

```
{(C_1, C'_1), ..., (C_q, C'_q)} ← CONGEN(V, V', q)
for i = 1 → q do
    V ← V ∪ {C_i, C'_i}
    E ← E ∪ {(C_i, C'_i)}
```

Remark. We concede that our approach to finding an independent set is not guaranteed to find a solution. This may impede attempts to prove the completeness of the UPUMP algorithm. We address this issue in Section 8 and state the modification that could lead to a probabilistic completeness proof of UPUMP.

6 Algorithm for the k-Color Case

We describe the changes required to transform UPUMP into KPUMP—an algorithm for the k-color problem. We stress that the extension to the k-color case is straightforward and we provide it here only for the completeness of presentation. KPUMP simultaneously samples several pumped configurations—each corresponds to a different color and hence to a different unlabeled problem. The resulting pebble graphs are constructed in a manner that prevents collision between robots of different colors. This calls for the redefinition of the edge planner mechanism (Section 4) as well as other components.

Algorithm 5 QUERY(S, T, q)

$G_S = (S, \emptyset)$; $G_T = (T, \emptyset)$
for all $G \in \mathcal{G}$ **do**
    CONNECT$(G, G_S, \mathcal{H}, q)$
    CONNECT$(G, G_T, \mathcal{H}, q)$
if S, T not connected in $\mathcal{H}$ **then**
    return FAILURE
return RETRIEVE_PATH$(\mathcal{H}, S, T)$

Algorithm 6 RETRIEVE_PATH$(\mathcal{H}, S, T)$

$\Pi \leftarrow \emptyset$
$\{C_0, \ldots, C_\ell\} \leftarrow$ GRAPH_PATH$(\mathcal{H}, S, T)$
for $i = 1 \rightarrow \ell$ **do**
    if $C_{i-1} \equiv C_i$ **then**
        $G \leftarrow$ pebble graph of C_i
        $\pi^* \leftarrow$ PEBBLE_SOLVER(G, C_{i-1}, C_i)
        $\pi \leftarrow$ TRANSFORM_PATH(π^*)
        Π.append(π)
    else
        Π.append(π_{C_{i-1}, C_i})
return Π

6.1 Composite Pebble Graphs

We begin with several definitions that extend the primitives presented in the description of UPUMP. Recall that the k-color problem $\mathcal{L}$ is defined by $\mathcal{U}_1, \ldots, \mathcal{U}_k$ where each $\mathcal{U}_i = (R_i, S_i, T_i)$ is an unlabeled problem and $|R_i| = m_i$.

Definition 4. Let $\mathbb{C} = \{C_1, \ldots, C_k\}$ be a collection of k configurations, where C_i is a configuration of $\mathcal{U}_i$. $\mathbb{C}$ is a *composite configuration* if for every $c \in C_i, c' \in C_j$, where $i \neq j$, it holds that $R_i(c) \cap R_j(c') = \emptyset$.

Definition 5. Let $\mathbb{V} = \{V_1, \ldots, V_k\}$ be a collection of pumped configurations, where V_i is a pumped configuration for $\mathcal{U}_i$. $\mathbb{V}$ is a *composite pumped configuration* if every $\mathbb{C} = \{C_1, \ldots, C_k\}$, such that $|C_i| = m_i$ and $C_i \subset V_i$, is a composite configuration.

Let $\mathbb{V}$ be a composite pumped configuration, as defined above. We construct a pebble graph for every pumped configuration V_i of $\mathbb{V}$. The edges of every graph are generated in a similar manner to the unlabeled case, although here we impose more restrictions to avoid the collision between robots of different colors.

Next, we generate the *composite pebble graph* $\mathbb{G} = \{G_1, \ldots, G_k\}$, where G_i is the pebble graph that resulted from the pumped configuration V_i. We now define an equivalence relation between composite configurations of the same composite pebble graph. Recall that two configurations are equivalent, if their signatures are identical (Definition 2). We generalize this notion for the case of composite configurations.

Definition 6. Let $\mathbb{G} = \{G_1, \ldots, G_k\}$ be a composite pebble graph, where $G_i = (V_i, E_i)$. Let $\mathbb{C} = \{C_1, \ldots, C_k\}, \mathbb{C}' = \{C'_1, \ldots, C'_k\}$ be two composite configurations, where $C_i, C'_i \subset V_i$. We say that $\mathbb{C}$ and $\mathbb{C}'$ are *equivalent*, and denote this relation by $\mathbb{C} \equiv \mathbb{C}'$, if for every $1 \leq i \leq k$ it holds that $C_i \equiv C'_i$, where the latter "$\equiv$" symbol represents the equivalence relation between configurations.

The following proposition is a generalization of Proposition 1 and its proof is omitted as it is similar to the proof for the unlabeled case.

Proposition 2. *Let $\mathbb{C} = \{C_1, \ldots, C_k\}, \mathbb{C}' = \{C'_1, \ldots, C'_k\}$ be two composite configurations of the same composite pebble graph. If $\mathbb{C} \equiv \mathbb{C}'$ then there exists a solution to the k-color problem $\{\mathcal{U}'_1, \ldots, \mathcal{U}'_k\}$, where $\mathcal{U}'_i = (R_i, C_i, C'_i)$.*

6.2 Description of KPUMP

We describe the sampling-based algorithm for the k-color case. KPUMP constructs a roadmap $\mathcal{H}$ whose vertices are composite configurations (recall that

in UPUMP, the vertices of this roadmap were configurations). The edges of $\mathcal{H}$ represent valid paths between composite configurations. These paths either connect equivalent composite configurations, as described in Proposition 2, or composite configurations from different composite graphs, where the latter paths are generated using the following mechanism, which is a generalization of the connection generator (Section 4.3). Given two composite pumped configurations, and an integer q, the *composite connection generator* returns a collection of q paths, for the k-color problem, between the two composite pumped configurations.

We now return to the description of KPUMP. KPUMP samples composite pumped configurations $\mathbb{V}_1, \mathbb{V}_2, \ldots, \mathbb{V}_g$ and generates the respective composite pebble graphs $\mathbb{G}_1, \ldots, \mathbb{G}_g$. Given a path between two composite configurations $\mathbb{C}, \mathbb{C}'$ returned by the composite connection generator we add the vertices $\mathbb{C}, \mathbb{C}'$ to $\mathcal{H}$ and the respective edge. Finally we connect the start and target composite configurations $\mathbb{S}, \mathbb{T}$, respectively, to previously sampled composite pebble graphs.

7 Experimental Results

We describe experimental results for the case of disc robots in the plane moving amidst polygonal obstacles. We show results for five challenging scenarios and compare the performance of KPUMP with two other sampling-based algorithms. Specifically we compare KPUMP with the PRM implementation of the OOPSMP package [20] on inputs of the fully-colored problem. For other inputs we use a basic sampling-based algorithm for the k-color problem called KBASIC, described later on.

KPUMP was implemented in C++ using CGAL Arrangements [7] and the Boost Graph Library (BGL) [23]. The code was tested on a PC with Intel i7-2600 3.40GHz processor with 8GB of memory, running a Windows 7 64-bit OS. For the implementation of the *local planner* a straight-line connection strategy [1] was used. This strategy attempts to move the robot along a straight line drawn between two positions.

Parameters of KPUMP. The algorithm has three parameters that affect its performance: g describes the number of the sampled pebble graphs in the UPUMP algorithm, or the number of composite pebble graphs in KPUMP; q is the number of connections produced by the connection generator between two samples; μ is the maximal number of single-robot configurations that one sample comprises, i.e., for every sampled pumped configuration $\mathbb{V} = \{V_1, \ldots, V_k\}$ it holds that $\sum|V_i| \leq \mu$. The value of the latter parameter depends on the input problem. For unlabeled problems, increasing μ results in increased connectivity of the resulting pebble graphs. Thus, it will be beneficial that the pumped configurations will be as large as possible (limited by the topology of the scenario). On the other hand, in k-colored problems where

$k > 1$ the value μ has to be set more carefully as an excessively high value of μ will reduce the connectivity of the pebble graphs. This stems from the fact that a single-robot path produced by the edge planner has to avoid collision with robots from other groups. Consequently, as the value of μ grows it becomes harder to connect single-robot configurations using an edge planner.

Test Scenarios. The scenarios are illustrated in Figure 3 and represent a variety of challenging problems. The unlabeled problem in (a) involves the motion of a large collection of robots. Scenarios (b) and (e) describe 2-color and 4-color problems comprising a large number of robots as well. Although scenarios (c), (d) do not involve as many robots, they are nevertheless challenging. This range of problems demonstrate the work of the various components of the KPUMP algorithm. In the first three scenarios the resulting pebble graphs have a low number of connected components due to the low value of k (as in scenario (a)) or high clearance from the obstacles (as in (e)). Therefore, large portions of the resulting paths involve the motions of the robots on paths induced by pebble problems. While the generated graphs in scenarios (c) and (d) have low connectivity, KPUMP still performs well—due to the use of the connection generator component.

Table 1: Results for selected scenarios.

	Properties			Parameters			Time
	k	m	M	g	q	μ	
(a)	1	25	25	2	5000	150	23.2
(b)	2	8	16	50	1000	40	20.3
(c)	8	1	8	100	150	32	213.7
(d)	5	1	5	50	100	25	1.9
(e)	4	3	12	40	250	28	32.9

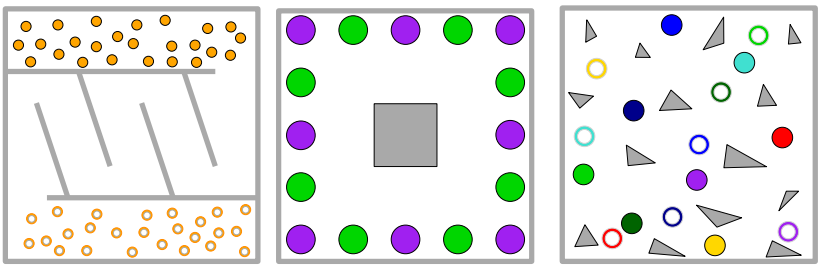
The results of running KPUMP for specific parameters are given in Table 1. In addition to the parameters mentioned above, the table contains the values k for the number of colors, m the number of robots in every color and M the total number of robots. The running times are given in seconds and represent the overall duration of the preprocessing and query phases, for a single query. The parameters used by KPUMP and other algorithms, mentioned later on, were manually optimized over a concrete set. A failure was declared when an algorithm was unable to solve a scenario for more than three runs out of five.

Comparison with Other Algorithms. The first part of the comparison involves solely inputs of the fully-colored problem. We compare KPUMP with the implementation of PRM provided by OOPSMP, which, by our experience, is very efficient. This algorithm is designed for solving fully-colored multi-robot motion planning problems. While OOPSMP required 100 seconds to solve scenario (d), KPUMP managed to solve it in 1.9 seconds. Scenario (c) proved to be even more challenging for OOPSMP, which failed to solve it, even when was given 5000 seconds of preprocessing time, whereas KPUMP solved in 213.7 seconds.

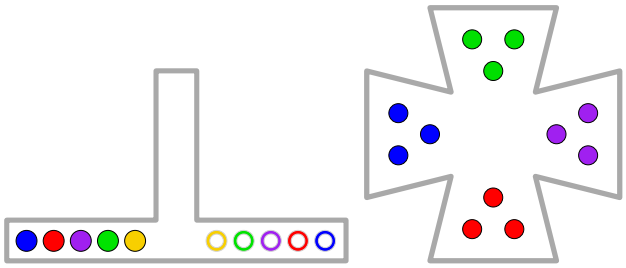
In order to provide a more informative comparison, we ran both algorithms on scenarios (c),(d), only that now we increased the difficulty of these sce-

narios gradually—incrementally introducing the robots, i.e., starting with a single robot and adding the others one by one, as long as OOPSMP succeeded solving the new inputs in reasonable time. In this case OOPSMP was able to solve scenario (c) with five robots, while the case of six robots was out of its reach (when given 5000 seconds of preprocessing time). The speedup of KPUMP compared to OOPSMP for this new range of scenarios is depicted in Figure 4 along with an additional test case ("decoupled-simple"), which is a simpler variant of scenario (c) with some of the obstacles removed and the radius of the robots is decreased. The latter was designed to test the performance of OOPSMP on problems involving a higher number of robots.

As we are not aware of any other algorithms for the k-color problem, we designed a basic algorithm to compare KPUMP with. This algorithm, which we call KBASIC, os a special case of KPUMP that samples configurations, instead of pumped configurations, and can be viewed as an extension of PRM for the k-color case. The entire set of scenarios (a)-(e)(in their original form) proved to be too challenging for KBASIC, which spent at times more than ten minutes. Similarly to the previous comparison we designed a set of simple



(a) Unlabeled (b) 2-Color (c) Fully-Colored: Decoupled



(d) Fully-Colored: Coupled (e) 4-Color

Fig. 3: [Best viewed in color] Start positions of the robots are indicated by discs while target positions are illustrated as circles in respective colors (unless otherwise indicated). (a) Unlabeled scene with twenty five robots. (b) 2-Color scene; the two groups are required to switch positions. (c) Fully-colored scene with eight robots. (d) Fully-colored scene with five robots. (e) 4-Color scene; every group has to move in a clockwise manner to the next room, e.g., the blue group should move to the top room.

test scenarios. Specifically, scenario (e) was converted into five k-color problems for $1 \leq k \leq 5$ by partitioning the robots into k groups such that a robot number i was assigned to the group i mod k. Then, as in the previous comparison, the robots were introduced incrementally. Figure 4 depicts the speedup of KPUMP compared with KBASIC for each of the k-color problems. This shows that KPUMP outperforms KBASIC in every possible setting, be it a k-color, unlabeled or fully-colored problem.

8 Discussion and Further Work

In this section we discuss the various properties of the KPUMP algorithm and novelties it encompasses, as well as directions for future research.

8.1 Shortcomings of the Composite Robot Approach

The traditional composite robot approach to the multi-robot problem treats the group of robots as one composite robot whose configuration space is the Cartesian product of the configuration spaces of the individual robots. With this approach, single-robot tools, such as sampling-based algorithms, can be used to solve multi-robot problems. For instance, this technique is used in the software packages OOPSMP and OMPL [12, 20] where PRM is applied to the fully-colored problem, and in the KBASIC algorithm discussed above. Paths generated by this approach usually force the robots to move simultaneously from one placement to the other, where none of the robots remains in the same position while the others are moving.

We believe that such paths are unnatural in the multi-robot setting and are more difficult to produce than paths that involve motion of only few robots at a time. Given collision-free placements for all the robots it is usually possible to move some of the robots to different placements without altering the placements of the rest of the robots, i.e., those robots remain still. For instance, consider a configuration $C = \{c_1, \ldots, c_m\}$ for some unlabeled prob-

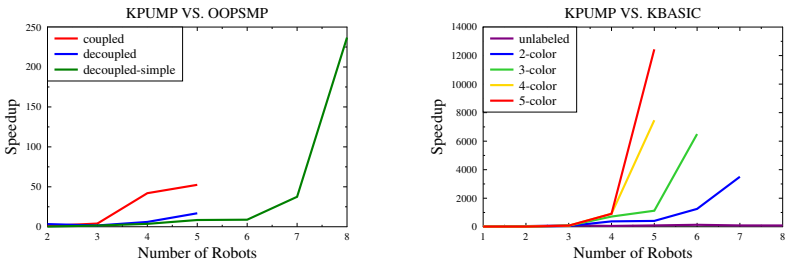


Fig. 4: [Best viewed in color] Comparing KPUMP with OOPSMP/PRM and KBASIC

lem $\mathcal{U}$. Unless the workspace is extremely tight, another configuration C' can be derived from C where only c_1 is moved to c_1'. Moreover, connecting two such configurations by a path requires only a single-robot collision-free path for which the moving robot does not collide with the other robots placed in $c_2, \ldots, c_m$. In contrast, the connection of two "unrelated" configurations by a path imposes much harder constraints—m single-robot collision-free paths have to be created and in addition, robots moving along those paths must not collide with each other.

KPUMP utilizes this observation by restricting the movements of the robots along certain path sections—induced by pebble problems—to motions of individual robots. We emphasize that KPUMP does not preclude simultaneous movements of robots when necessary, specifically on path sections where the robots move from one pebble graph to the other along paths generated by the connection generator.

8.2 Amplification of Samples

Pumped configurations that are sampled by KPUMP, and the resulting pebble graphs, are fairly simple structures which require only little effort to generate. Yet, using the transformation to pebble problem, these samples are amplified to describe not only placements and paths for single robots, but also to represent an incredible amount of paths and positions for all the robots in a given problem. However, this information is not represented explicitly and only little storage space is required to represent a pebble graph. In addition, a small number of configurations must be stored. Specifically, these are configurations through which the pebble graphs connects to other graphs. Such configurations are selected by the *connection generator*. Similarly, this component does not require an explicit representation of all the configurations represented by the pebble graph. Furthermore, continuing the theme presented here that one action leads to a large number of outcomes, namely, a sample of a pumped configuration results in many configurations, a path generated by a connection generator not only connects two configurations from the two pebble graphs, but also a large number of configurations from them, which are not necessarily directly connected. Thus, these properties enable KPUMP to generate a variety of configurations and motions of the robots, using only few samples. To reproduce this variety by KBASIC one must generate far more samples.

An additional advantage of the use of pebble graphs lies in the fact that they can be connected more easily than two configurations, when a powerful component as the connection generator is at hand. Using this component, KPUMP succeeds in solving difficult scenarios even when the generated pebble graphs suffer from low connectivity, as in scenarios (c) and (d).

8.3 Completeness

Our immediate future goal is to investigate the completeness of our approach. We propose the following line of attack to show probabilistic completeness. We start with a simple observation.

Observation 1 *Let $\mathcal{U} = (R, S, T)$ be an unlabeled problem, where $R = \{r_1, \ldots, r_m\}, S = \{s_1, \ldots, s_m\}, T = \{t_1, \ldots, t_m\}$. There is a solution to $\mathcal{U}$ if and only if there exists a permutation σ_T, such that there is a solution to the fully-colored problem $\mathcal{L} = \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$, where $\mathcal{U}_i = (r_i, s_i, t'_i)$ and $\sigma_T(T) = (t'_1, \ldots, t'_m)$.*

At the heart of the new idea is the trivial observation, that is interesting in the context of many robots, that the preprocessing stage of the PRM algorithm is oblivious to the queries. Thus, using a single PRM roadmap for the fully-colored case, that is constructed from a sufficient amount of samples, one expects to find a solution to an unlabeled problem, by applying queries of fully-colored problems, as described in Observation 1.

This leads to the following course of action. Suppose that it can be shown that UPUMP simulates a run of the PRM algorithm for a solvable fully-colored problem that is an instance of the unlabeled problem. Then, by the completeness of the PRM it can be deduced that UPUMP is probabilistically complete. This approach to the completeness proof, although it looks promising, requires the modification of the connection generator, which currently attempts to find a connection between two pumped configurations using a greedy heuristic (for this reason the new idea is not immediate to carry out).

To overcome this hurdle, we may substitute the connection generator algorithm by an Integer Programming formulation of the independent set problem.

Even if, as we hope, these observations will lead to a probabilistic completeness proof, they leave at least one big open problems. Can one formally prove the efficiency gain in using U/KPUMP over a naive application of PRM to the multi-robot case as a composite robot?

References

1. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: Choosing good distance metrics and local planners for probabilistic roadmap methods. IEEE Transactions on Robotics **16**(4), 442–447 (2000)
2. Aronov, B., de Berg, M., van der Stappen, A.F., Svestka, P., Vleugels, J.: Motion planning for multiple robots. Discrete & Computational Geometry **22**(4), 505–525 (1999)
3. van den Berg, J., Overmars, M.: Prioritized motion planning for multiple robots. In: International Conference on Intelligent Robots and Systems (IROS), pp. 430 – 435 (2005)

4. van den Berg, J., Snoeyink, J., Lin, M., Manocha, D.: Centralized path planning for multiple robots: Optimal decoupling into sequential plans. In: Robotics: Science and Systems (RSS) (2009)
5. Calinescu, G., Dumitrescu, A., Pach, J.: Reconfigurations in graphs and grids. SIAM Journal on Discrete Mathematics **22**(1), 124–138 (2008)
6. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, G., Kavraki, L., Thrun, S.: Principles of Robot Motion: Theory, Algorithms, and Implementations. MIT Press (2005)
7. Fogel, E., Halperin, D., Wein, R.: CGAL Arrangements and Their Applications: A Step-by-Step Guide. Geometry and Computing. Springer (2012)
8. Goraly, G., Hassin, R.: Multi-color pebble motion on graphs. Algorithmica **58**(3), 610–636 (2010)
9. Hirsch, S., Halperin, D.: Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane. In: Workshop on the Algorithmic Foundations of Robotics (WAFR), pp. 239–255. Springer (2002)
10. Hopcroft, J., Schwartz, J., Sharir, M.: On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "Warehouseman's problem". International Journal of Robotics Research **3**(4), 76–88 (1984)
11. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
12. Kavraki Lab: The open motion planning library (OMPL) (2010). ompl.kavrakilab.org
13. Kornhauser, D.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. M.Sc. thesis, Department of Electrical Engineering and Computer Scienec, Massachusetts Institute of Technology (1984)
14. Kornhauser, D., Miller, G., Spirakis, P.: Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In: Foundations of Computer Science (FOCS), pp. 241–250. IEEE Computer Society (1984)
15. Kuffner, J.J., Lavalle, S.M.: RRT-Connect: An efficient approach to single-query path planning. In: International Conference on Robotics and Automation (ICRA), pp. 995–1001 (2000)
16. LaValle, S.M.: Planning Algorithms. Cambridge University Press (2006)
17. Leroy, S., Laumond, J.P., Simeon, T.: Multiple path coordination for mobile robots: A geometric algorithm. In: International Joint Conference on Artificial Intelligence, pp. 1118–1123 (1999)
18. Loyd, S.: Mathematical Puzzles of Sam Loyd. Dover (1959)
19. Luna, R., Bekris, K.E.: Efficient and complete centralized multi-robot path planning. In: International Conference on Intelligent Robots and Systems (IROS) (2011)
20. Plaku, E., Bekris, K.E., Kavraki, L.E.: OOPS for motion planning: An online open-source programming system. In: International Conference on Robotics and Automation (ICRA), pp. 3711–3716. IEEE (2007)
21. Sanchez, G., Latombe, J.C.: Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In: International Conference on Robotics and Automation (ICRA) (2002)
22. Schwartz, J.T., Sharir, M.: On the piano movers' problem: III. Coordinating the motion of several independent bodies. International Journal of Robotics Research **2**(3), 46–75 (1983)
23. Siek, J., Lee, L.Q., Lumsdaine, A.: Boost graph library. http://www.boost.org/libs/graph/ (2000)
24. Svestka, P., Overmars, M.H.: Coordinated path planning for multiple robots. Robotics and Autonomous Systems **23**, 125–152 (1998)
25. Wagner, G., Choset, H.: M*: A complete multirobot path planning algorithm with performance bounds. In: International Conference on Intelligent Robots and Systems (IROS), pp. 3260–3267 (2011)

26. Wagner, G., Kang, M., Choset, H.: Probabilistic path planning for multiple robots with subdimensional expansion. In: International Conference on Robotics and Automation (ICRA), pp. 2886–2892 (2012)