MASTER

Unlabeled Multi-Robot Motion Planning with Tighter Separation Bounds

Slot, Stijn J.

Award date:
2020

Department of Mathematics and Computer Science
De Rondom 70, 5612 AP Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Unlabeled Multi-Robot Motion Planning with Tighter Separation Bounds

Author
Stijn Slot (S.J.)

Supervisors
Kevin Buchin
Irina Kostitsyna

Date
October 2020

Table of contents

Title
Unlabeled Multi-Robot Motion Planning with Tighter Separation Bounds

Table of contents

Title
Unlabeled Multi-Robot Motion Planning with Tighter Separation Bounds

Where innovation starts

Abstract

We consider the unlabeled motion planning problem of m unit disc robots moving in a simple polygonal workspace $\mathcal{W}$ of n edges. The goal is to find a motion plan that moves the discs to a given set of m target positions. For the unlabeled variant, it does not matter which robots reaches which target position as long as all target positions are occupied in the end. In this thesis we show that this problem is always solvable assuming some minimum separation between the start and target positions. Moreover, we describe an algorithm that can always find a solution in $O((m+n)\log(m+n)) + mn + m^2)$.

This result improves upon a previous work by Adler et al. [1] by showing that the problem is still efficiently solvable while assuming less separation between the start and target positions. Specifically, we have lowered the separation assumed between any pair of start and target positions from four to three, and show that it can even be dropped entirely when the free space consists of a single connected component. In addition, we prove that these separation assumptions are tight, showing that the problem does not always have a solution for lower bounds.

1 Introduction

In the multi-robot motion planning problem the goal is to plan the motion of several robots operating in a common environment, while avoiding collisions with obstacles and other robots. Stated simply, the robots need to move from starting positions to target positions as fast and efficient as possible without crashing. Although the problem might appear easy at first, it can quickly become more complex once there are multiple robots and the environment contains many obstacles.

Motion planning algorithms should take as input a description of the current state of the robots, the environment, and a target state. It should produce as output a motion plan for each robot to move from the current state to the target state. The robots are typically assumed to be simple two-dimensional shapes, like discs, squares, or polygons, moving in a two-dimensional environment, most often a polygon which may or may not contain holes (i.e. obstacles). Though real-life robots are of course three-dimensional objects, their movement is mostly two-dimensional. Therefore, it is reasonable to model the robots as two-dimensional objects in a two-dimensional environment. We can think of this as looking at the robots and their environment from a bird's-eye view.

Most often, we assume the robot can move freely around the two-dimensional environment, using actuators such as wheels. In some variants rotation or other manipulations of the robot's shape or size is also allowed to enable the robot to take paths it otherwise could not. In general, the number of independent parameters which are necessary to describe a robot's state are called the *degrees of freedom*, and the difficulty of the problem increases the higher degrees of freedom it has.

Robot motion planning has many applications and is highly relevant for a variety of fields. There is a growing need for path planning algorithms that can efficiently handle multiple robots and many new and exciting applications require multiple robots to work in a shared space to achieve a common goal. For instance, transportation robots driving in a shared storage room that need to move goods around as efficiently as possible without crashing or getting stuck. Or for example multiple manipulators, like robotic arms, handling objects traveling on a conveyor belt which need to coordinate their motion to achieve their desired goal. Outside of robotics, the problem also has applications for computer simulations (e.g. crowd simulation), artificial intelligence, biology, and many more.

The multi-robot motion planning problem is an extension of the single-robot motion planning problem, by the (obvious) inclusion of more robots. For the single-robot motion planning problem, we need to find a motion plan for a robot from its starting position to a target position while avoiding collisions with obstacles. Here, the environment is static, which reduces the complexity of the problem. Finding a solution to the single-robot motion planning prob-

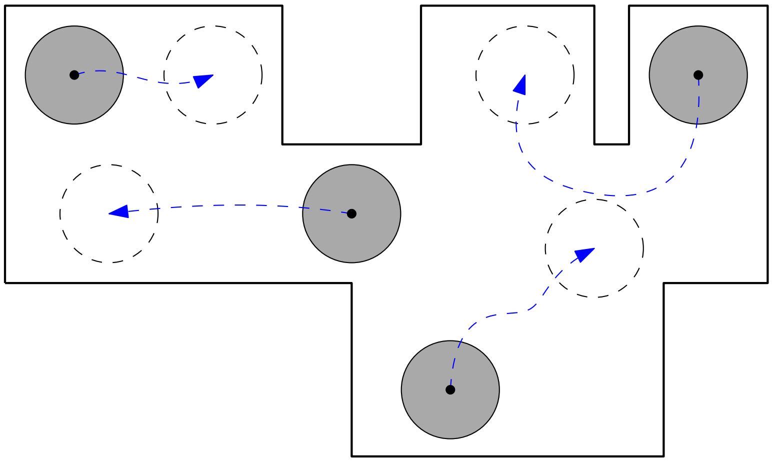


Figure 1.1: An example of a multi-robot motion planning problem. The robots, shown in gray, need to find paths to the target positions, shown as dashed circles. In blue a potential solution is given.

lem can therefore be done relatively easily by finding a path through the *configuration space*, which is the region of all valid positions where the robot is allowed to be (i.e. it does not collide with any obstacle).

However, the inclusion of multiple robots greatly increases the complexity of the problem due to the high degrees of freedom that are introduced. The multi-robot motion planning problem can be seen as planning a path through the *composite configuration space*, which is the combined product of the individual configuration spaces of the robots. Unfortunately, the composite configuration space grows exponentially with the number of robots, which means finding an optimal solution quickly becomes infeasible once we have more than a handful of robots. Thus, many motion planning algorithms instead focus on efficiently finding a valid solution that is preferably still close to optimal.

Multiple variants of the multi-robot motion planning problem are studied. Different assumptions can be made on the shape of the robots and its environment, as well as the type of movement (e.g. whether rotation is allowed). Furthermore, the most common variant is *labeled* robot motion planning, where each robot has a designated target position that it needs to reach. In contrast, in the *unlabeled* variant each robot only needs to reach some target position, such that at the end each target position is occupied by a robot. In this variant it does not matter which robot occupies which target position, as long as all target positions are occupied in the final state. There also exists a *colored* variant, which is a mix of the two where a target position needs to be occupied by a robot with a certain “color” (i.e. type). In this thesis we will study the unlabeled version of the problem, which has received considerably less attention than the labeled variant.

Unfortunately, the unlabeled variant has been proven to be PSPACE-hard for unit-square robots [16] and also for disc robots with two different radii [2]. Thus, it seems impossible for a computer to find a solution to the unlabeled problem both reliably and efficiently. These hardness proofs, however, rely on constructions where the robots are positioned very close together without much room for maneuvering, which is perhaps not a very realistic scenario. Surprisingly, when we assume some minimum spacing between the start and target positions the problem for robots moving in a simple polygon always has a solution, and the solution can be found in polynomial time, as shown by Adler, de Berg, Halperin, and Solovey [1]. In their paper they assume a minimum distance of four between the start and target positions for unit-disc robots. The *separation*, the minimum distance between the start and target positions, thus plays a key role in the difficulty of the problem. The separation bounds assumed by Adler et al. are not proven to be tight, however, so the question remains for what separation bounds the problem is always solvable. The goal of this project is to find the minimal separation necessary for which it is always possible to solve the motion planning problem, and to describe an algorithm that can do so efficiently.

1.1 Related work

The multi-robot motion planning problem has received much attention over the years. Already in 1983, the problem was first described in a paper on the *Piano Mover's problem* by Schwartz and Sharir [13]. Later that year, an algorithm for the case of two or three disc-robots moving in a polygonal environment was described, running in $O(n^3)$ and $O(n^{13})$ respectively [14]. This was then later improved by Yap [20] to $O(n^2)$ and $O(n^3)$ for two and three robots using the *retraction method*. A general approach using *cell decomposition* was later developed in 1991 by Sharir and Sifrony [15] that could deal with a variety of robot pairs in $O(n^2)$.

Unfortunately, when the number of robots increases beyond a fixed constant, the problem becomes hard. In 1984, a general (labeled) case of the multi-robot motion planning with disc robots and a simple polygonal workspace was shown to be strongly NP-hard [18]. This is a somewhat weaker result than the PSPACE-hardness for many other motion planning problems. For rectangular robots in a rectangular workspace, however, the problem was shown to be PSPACE-hard [7]. This result has later been refined to show that for PSPACE-completeness it is sufficient to have only 1x2 or 2x1 robots in a rectangular workspace [6].

Despite the hardness results for the general problem, various heuristic and/or practical path planners have been developed. *Sampling-based* techniques have shown to be reliable and effective at traversing the high dimensional configuration space of the multi-robot setting. In 1996, Kavraki, Svestka, Latombe, and Overmars [8] used a sampling approach based on constructing a *probabilistic roadmap* of the composite configuration space which could find a solution effectively with high probability. The probabilistic roadmaps can be widely applied to explore the high dimensional configuration space, such as settings with a large number of robots or robots with high degrees of freedom. However, in experiments by Sanchez and Latombe [12] already for 6 robots with 36 degrees of freedom the algorithm requires minutes to find the optimal solution. Thus, for large number of robots with high degrees of freedom, such centralized, coupled algorithms are not sufficiently scalable even when using a sampling-based approach.

Decoupled algorithms, where robots are first considered individually and issues are resolved locally, provide a scalable solution but at the cost of theoretical properties such as optimality and completeness. Therefore, many coupled algorithms have been proposed that combine individual (probabilistic) roadmaps for robots in a way that remains scalable yet keeps certain theoretical guarantees [4, 5]. In particular, Dobson et al. [3] show an algorithm called dRRT* that builds a roadmap for each robot and then implicitly searches the tensor product of these roadmaps in the composite space. They show that dRRT* is asymptotically-optimal, meaning the probability of finding the optimal solution asymptotically increases to 1 when the sampling size increases.

With respect to unlabeled motion planning, the problem was first considered by Kloder and Hutchinson [10] in 2006. In their paper they provide a sampling-based algorithm which is able to solve the problem. In 2016, Solovey and Halperin [16] have shown that for unit square robots the problem is PSPACE-hard using a reduction from *non-deterministic constraint logic* (NCL [6]). This PSPACE-hardness result also extends to the labeled variant for unit square robots. Just recently, the unlabeled variant for two classes of disc robots with different radii was also shown to be PSPACE-hard [2], with a similar reduction from NCL. In the reduction they use robots of radius $\frac{1}{2}$ and 1. In contrast, the earlier result for disc robots by Spirakis and Yap [18] used discs of many sizes with larger differences in radii.

Fortunately, efficient (polynomial-time) algorithm can still exist when some additional assumptions are made for the problem. Turpin, Michael, and Kumar [19] consider a variant of the unlabeled motion planning problem where the collection of free configurations surrounding every start or target position is star-shaped. This allows them to create an efficient algorithm for which the path-length is minimized. In the paper by Adler et al. [1], an $O(n \log n + mn + m^2)$ algorithm is given for the unlabeled variant, assuming the workspace is a simple polygon and the start and target positions are *well-separated*, which is defined as minimum distance of four between any start or target position. Their algorithm is based on creating a motion graph on the start and target positions and then treating this as an *unlabeled pebble game*, which can be solved in $O(S^2)$ where S is the number of pebbles [11]. Furthermore, in the paper by Adler et al. [1] the separation bound $4\sqrt{2}-2$ (≈ 3.646) is shown to be sometimes necessary for the problem to always have a solution. When the workspace contains obstacles, Solovey, Yu, Zamir, and Halperin [17] describe an approximation algorithm which is guaranteed to find a solution when one exists, assuming also that the start and target positions are *well-separated* and a minimum distance of $\sqrt{5}$ between a start or target position and an obstacle.

In this thesis, we will explore under what setting the unlabeled multi-robot motion planning problem always has a solution and provide an algorithm that can find such a solution in polynomial time. Specifically, in this thesis we will focus on the exact separation bounds between start and target position that are sometimes necessary and always sufficient for the problem to be solvable. The goal is to improve upon the separation bounds that were assumed by Adler et al. [1] and give an algorithm that relies on tighter separation assumptions.

1.2 Definitions and notation

We consider the problem of m indistinguishable unit-disc robots moving in a simple polygonal *workspace* $\mathcal{W} \subset \mathbb{R}^2$ with n edges. The *obstacle space* $\mathcal{O}$ is defined as the complement of the workspace $\mathcal{O} \triangleq \mathbb{R}^2 \setminus \mathcal{W}$. We will refer to points $x \in \mathcal{W}$ as *configurations*, and we will say that a robot is at configuration x when its center is positioned at point $x \in \mathcal{W}$. For a given $x \in \mathbb{R}^2$ and $r \in \mathbb{R}_+$, we define $\mathcal{D}_r(x)$ to be the open disc of radius r centered at x. For convenience, from this point we will use a green colored disc of radius one to denote a start configuration

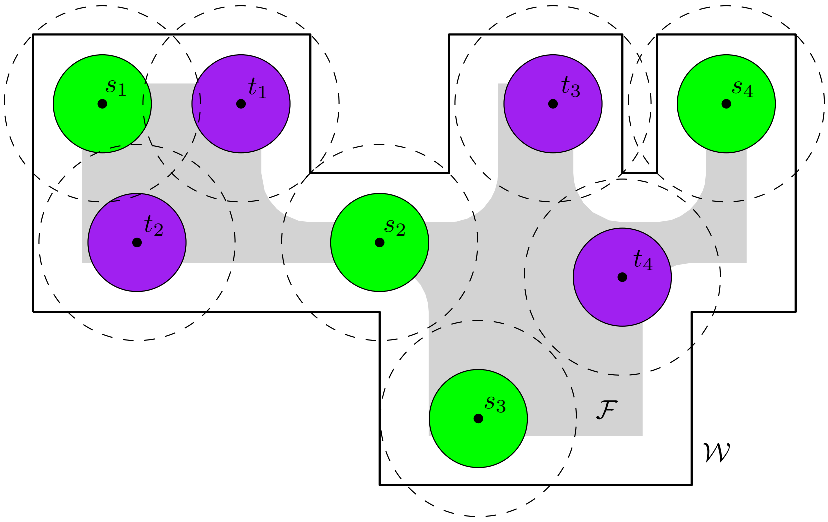


Figure 1.2: An illustration of some important definitions using the example from Figure 1.1. The workspace $\mathcal{W}$ is the outer polygon, the free space $\mathcal{F}$ is the inner gray area. The aura of the start and target configurations is shown as a dashed circle of radius two (for unit-disc robots).

in our illustrations and a purple disc of radius one for a target configuration.

The unit-disc robots are defined to be open sets, meaning a robot collides with the obstacle space $\mathcal{O}$ if and only if its center is at a distance less than one from $\mathcal{O}$. Thus, we can define the *free space* $\mathcal{F}$ to be all configurations that are at distance of at least one from the obstacle space, or formally $\mathcal{F} \triangleq \{x \in \mathbb{R}^2 \mid \mathcal{D}_1(x) \cap \mathcal{O} = \emptyset\}$. The free space is therefore a closed set of all configurations where a unit-disc robot does not collide with the obstacle space. Additionally, we require that robots do not collide with each other. Since the robots are open sets, no two robots are allowed to be less than a distance of 2 of each other. In other words, if a robot is at configuration x then no other robot can be at a configuration $y \in \mathcal{D}_2(x)$. For a configuration $x \in \mathbb{R}^2$ we will define the open disc $\mathcal{D}_2(x)$ to be the *collision disc*, or *aura*. Intuitively, the aura of a configuration x contains all configurations where a robot would collide with a robot at x. See Figure 1.2 for an illustration of the workspace $\mathcal{W}$, the free space $\mathcal{F}$ and the unit-disc robots with their auras. Furthermore, let $\delta(X)$ denote the boundary of some set $X \subset R^2$.

Besides the simple polygon $\mathcal{W}$ representing the workspace, we are also given the set of start configuration S and the set of target configuration T, such that $S, T \subset \mathcal{F}$. These represent the start and target positions for the m robots in our problem. We require that the robots do not overlap with one another when positioned at the start configurations for the problem to be valid (non-collision constraint), and similarly when the robots are on the target configurations. Formally, there should not exist two distinct start configurations $s_1, s_2 \in S$ such that $s_1 \neq s_2$ and $\mathcal{D}_1(s_1) \cap \mathcal{D}_1(s_2) \neq \emptyset$. And again, the same should be true for the set of target configurations T.

The goal of the problem is now to plan a collision-free motion for each of our m unit-disc robots from their starting configuration in S to some target configuration in T such that all target configurations T will be occupied by some robot. Since the robots are indistinguishable (i.e. unlabeled), it does not matter which robot ends up at which target configuration.

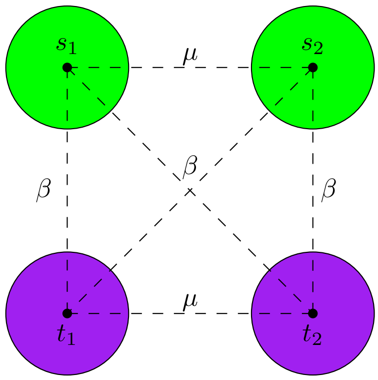


Figure 1.3: A visualization of the two types of separation, namely monochromatic separation denoted by μ and bichromatic separation denoted by β.

Formally, we wish to find continuous paths $\pi_i : [0,1] \to \mathcal{F}$, for $1 \le i \le m$, such that $\pi_i(0) = s_i$ and $\bigcup_{i=1}^{m} \pi_i(1) = T$. Additionally, we require that the robots do not collide with each other at any moment during their motion: For every $1 \le i \ne j \le m$ and every $\epsilon \in (0,1)$, we require $\mathcal{D}_1(\pi_i(\epsilon)) \cap \mathcal{D}_1(\pi_j(\epsilon)) = \emptyset$.

For a subset $Q \subset F$ of the free space, we will use $s(Q) = \{x \in S \mid x \in Q\}$ for the set of start configurations that reside in Q, and similarly let $t(Q) = \{x \in T \mid x \in Q\}$ be the set of target configurations in Q. We can then define the weight of Q as $w(Q) = |s(Q)| - |t(Q)|$. For the entire free space we have that $w(\mathcal{F}) = 0$, since there need to be an equal number of start and target configurations for the problem to have a solution.

Furthermore, we will distinguish between two types of separability bounds: monochromatic, namely between two start configurations or between two target configurations, which we denote by μ, and bichromatic,, namely between a start configuration and a target configuration, which we denote by β. See Figure 1.3 for an illustration of the two types of separability constraints.

1.3 Contributions

In this thesis we show that the unlabeled multi-robot motion planning problem is always solvable assuming monochromatic separation $\mu = 4$ and bichromatic separation $\beta = 3$ for unit-disc robots in a simple workspace. Moreover, we describe an algorithm that can always find a solution in polynomial time. Furthermore, we prove that the separability bounds assumed are tight, meaning it is sometimes necessary for the problem to have a solution. Additionally, if the free space consists of a single component then the bichromatic separation constraint can be dropped and the monochromatic separation is sufficient for the problem to always be solvable. The results described improve upon the results by Adler et al. [1] which described an algorithm that can always solve the problem assuming separation bounds of $\mu = \beta = 4$.

In Chapter 2 we show that the separation bounds assumed are sometimes necessary for the problem to have a solution. Namely, we show an instance where the problem is unsolvable when the monochromatic separation is less than $\mu = 4$, and similarly for when the bichromatic separation is less than $\beta = 3$. The latter proof relies on the free space consisting of multiple connected components. We will show that when the free space consists of a single connected component, bichromatic separation is not necessary for the problem to have a solution. The proofs for the separability bounds provide the lower bound to show that the assumed separation is tight.

Given the separation assumptions, we present two algorithms in Chapter 3 that can always solve the problem for a single free space component. The algorithms restrict the robots to be positioned on either a start or target configuration, and move one at a time between these configuration by using a *motion graph*. This simplifies the problem from an algorithmic perspective. The main difficulty lies in the lack of bichromatic separation ($\beta < 4$), since the collision discs of start and targets are allowed to overlap. This can cause difficult situations where a start and target configuration can together split the free space. Nonetheless, in this section we show a matching-based algorithm and a divide-and-conquer algorithm that are able to always solve the motion planning problem, and find a solution in polynomial time. In Chapter 4 these algorithms are extended to handle multiple free space components.

2 Tighter separation bounds

In this chapter we will explore the amount of *separation* between the start and target configurations which is sometimes necessary for the problem to always have a solution. As mentioned, separation refers to the minimum distance which is assumed to be present between any two start or target configurations. The minimum separation is a constraint we impose on the problem in order to make it easier to solve, since the base problem is likely PSPACE-hard. We will show that without a certain amount of separation there are instances of the problem which cannot be solved, thus the separation is sometimes necessary for the problem to be always solvable.

Adler et al. [1] describe an algorithm that can always find a solution to the unlabeled motion planning problem for unit-disc robots in a simple workspace, assuming a separation of four and that the number of start and target configurations is equal in every connected component of the free space. They do not make a distinction between monochromatic and bichromatic separation, so their result applies for $\mu = \beta = 4$. In the paper they refer to the configurations as *well-separated* to indicate this separation of four, however we will not be using this term. Importantly, the minimum separation of four is not shown to be tight, since only an example with separation of $\mu < 4\sqrt{2} - 2$ (≈ 3.646) is given for which a solution does not exist. In this chapter, we aim to tighten these separation bounds, as well as make a distinction between monochromatic and bichromatic separation. For both μ and β, a lower bound will be given for which a solution does not exists.

The lower bound instances given in this chapter were originally created by Bahareh Banyassady, Mark de Berg, Kevin Buchin, Karl Bringmann, Henning Fernau, Dan Halperin, and Yoshio Okamoto during the *Lorentz-Center Workshop on Fixed-Parameter Computational Geometry* in 2018. Their work was incredibly valuable as a starting point for this thesis, and for this chapter in particular.

2.1 Monochromatic separation

As described in Section 1.2, monochromatic separation μ refers to the minimum distance between any pair of start configurations or between any pair of target configurations. By the non-collision constraint, μ should always be at least two for the problem to be valid, otherwise the robots collide will collide at the initial state or at the goal state. We aim to find a tight lower bound for μ for which there always exists a solution.

Lemma 1. For $\mu < 4$ a solution does not always exist.

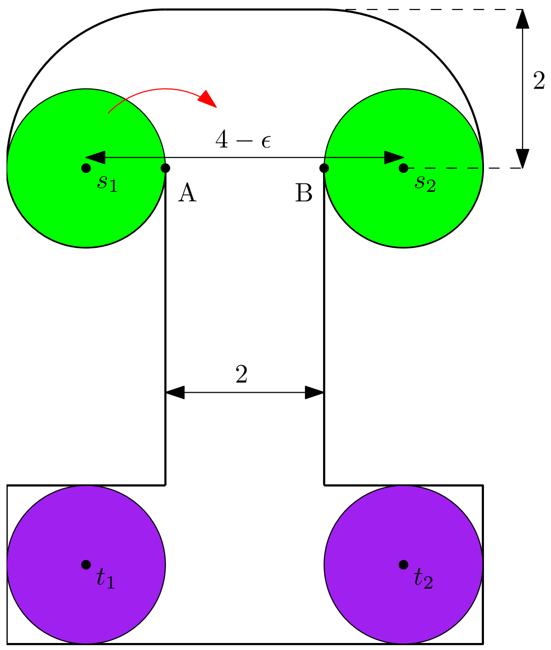


Figure 2.1: An instance to show that for $\mu < 3$ there does not always exist a solution. The two robots at the start configurations at the top have to move down the corridor to the two target configurations at the bottom. However, they always obstruct one another from moving into the corridor.

Proof. See Figure 2.1 for an instance where a solution does not exist when $\mu = 4 - \epsilon$ for some arbitrarily small $\epsilon > 0$.

In the example, two robots r_1, r_2 starting at start configurations s_1, s_2 need to move through a narrow corridor of width 2 to reach the target configurations t_1, t_2. The separation between s_1 and s_2 is the previously noted $4 - \epsilon$. Let points A, B be the endpoints of the corridor closest to s_1 and s_2, which in the example lie on the boundary of the robots at s_1 and s_2 respectively. Clearly, both robots cannot move into the corridor simultaneously, therefore assume w.l.o.g. that r_1 moves across the line segment $\overline{AB}$ first. Thus, for such a solution r_1 will need to rotate around point A and then move down the corridor.

We observe that points A and B, the end points of the corridor, must be below the line segment $\overline{s_1 s_2}$, given that the corridor has width 2 and the separation between s_1 and s_2 is less than 4. Note then by the triangle inequality we must have that the distance between A and s_2 is less than 3. This means that the aura of r_2 at s_2 intersects the movement of r_1 around A and into the corridor. Furthermore, there is no point in the free space where r_2 can move to give space to r_1, since any point obstructs the rotation of r_1 around A. Therefore, no solution exists for this instance. $\square$

Thus, for there to always exist a solution a monochromatic separation of $\mu = 4$ is necessary. Since we know an algorithm for when $\mu = \beta = 4$, the monochromatic separation is tight. Hence, we aim to reduce the bichromatic separation β.

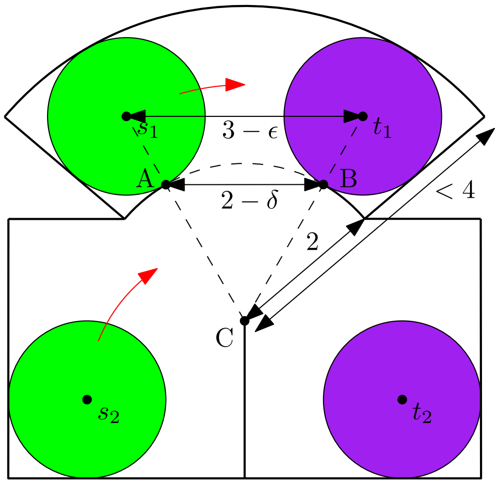


Figure 2.2: An instance to show that for $\beta < 3$ there does not always exist a solution. The free space consists of two components and in each of them a robot has to move from left to right. However, the robot in the top component always blocks the movement in the bottom.

2.2 Bichromatic separation

The bichromatic separation β refers to the minimum distance between any pair of a start and a target configuration, as explained in Section 1.2.

Lemma 2. For $\beta < 3$ a solution does not always exist.

Proof. See Figure 2.2 for an instance where a solution does not exist when $\beta = 3 - \epsilon$ for some arbitrarily small $\epsilon > 0$.

In the example, there are two connected components of the free space, both containing a start and target configuration (s_1, t_1 and s_2, t_2 respectively). The free space components containing the two robots are not connected, since points A and B have a distance of $2-\delta$ for some $\delta > 0$. Here, we define δ such that $\delta < \frac{2\epsilon}{3}$. In this example, A lies on the boundary of $\mathcal{D}_1(s_1)$ and B on the boundary of $\mathcal{D}_1(t_1)$. From these facts, it follows that A lies to the left of line segment $\overline{s_1C}$ and B lies to the right of line segment $\overline{t_1C}$. By the triangle inequality we know that the distance from s_1 to C must be less than 3, similarly for the distance from t_1 to C.

The key characteristic is that no matter where the robot in the top component is, it will block the movement from start to target of the robot in the bottom component. Since the top arc of the workspace is a semi-circle with center at C and radius less than 4, there is no point in the top component of the free space which does not block the movement in the bottom. Thus, the robot at s_2 can never reach t_2, which means no solution exists for this example. □

The non-existence of a solution when $\beta < 3$ stems from the interaction between start or goal position in one connected component of the free space with the motion of a robot in a neighboring component. However, when considering a single free space component, this type of interaction is no longer possible. In fact, we were not able to create a lower bound instance for a single free space component for which no solution exists. We therefore suspect that bichromatic separation is not necessary for there to always be a solution to the problem for a single free space component.

3 A single free space component

In this chapter we will consider the robot motion planning problem for a single maximal connected component F_i of the free space $\mathcal{F}$. Let $S_i \triangleq s(F_i)$ and $T_i \triangleq t(F_i)$ be the start and target configurations in F_i respectively. The conjecture is that, with the separation assumptions $\mu = 4$ (and $\beta = 0$), the problem is always solvable as long as F_i contains an equal number of start and target configurations.

Initially, we will assume the separation constraints of $\mu = 4$ and $\beta = 2$. This makes the problem somewhat easier since the separation constraints do not allow a configuration to be inside the aura of another configuration. In Section 3.7 we will then describe how to modify the algorithm(s) to handle the case when there is no bichromatic separation ($\beta = 0$). Furthermore, in Chapter 4 we will describe how to adjust the algorithm(s) to handle the entire free space $\mathcal{F}$, which might consist of multiple free space components.

The issue with having less than four bichromatic separation is shown in Figure 3.1. The narrow corridor at the top can only be traversed if neither the start configuration s nor the target configuration t is occupied, given that both auras intersect with the corridor. In particular, the red area of the t's aura blocks robots from moving through the corridor to the start s, but there is also no direct path from the red area to t that does not intersect with another configuration's aura (since any path from the red area to t will cross the aura of s). Thus, a robot at a start or target position can interfere with movement between other start/target configurations from a "remote" location.

As a result, the algorithm from Adler et al. [1] that uses the separation assumption $\mu = \beta = 4$ cannot be applied once we no longer assume $\beta = 4$. New algorithms will therefore have to be designed that can handle such "blocking" configurations efficiently in order to solve the motion planning problem.

We have considered multiple different algorithms for solving the single free space component. All approaches will use a motion graph, described in Section 3.3, in order to solve the problem. Very briefly, the motion graph captures "adjacencies" between the start/target configurations and the algorithms will use this to only move robots one at a time between the start/target configurations. This simplifies the problem, since after generating the motion graph the free space can be ignored. Before we describe the algorithms, we will first define some preliminaries in Section 3.1 and a useful graph data structure in Section 3.2. Then, the motion graph is defined and we describe how it can be constructed in Section 3.3

Afterwards we discuss three algorithms, namely a greedy algorithm, a matching algorithm, and a divide-and-conquer algorithm in Section 3.4, Section 3.5, and Section 3.6 respectively. Two of these algorithms, the matching algorithm and the divide-and-conquer algorithm, are proven to always be able to solve the motion planning problem for unit-disc robots in a sim-

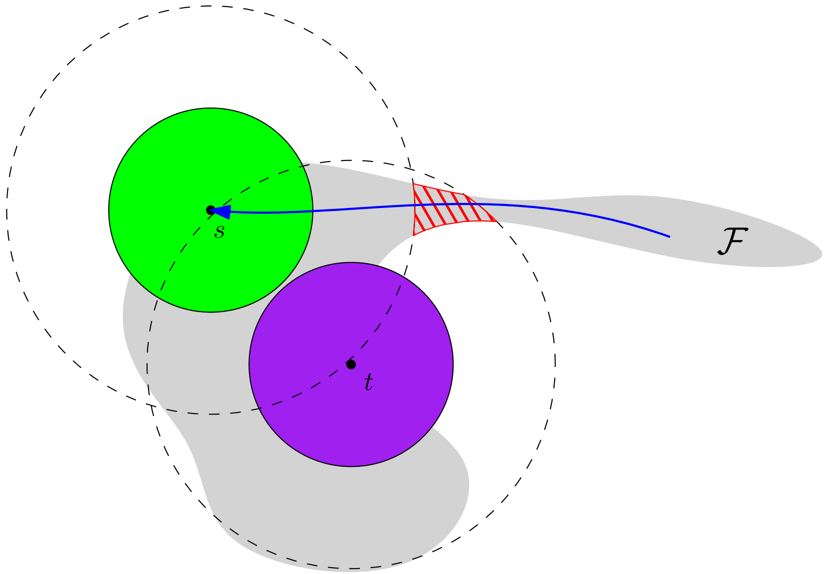


Figure 3.1: An instance where the lack of bichromatic separation poses a problem. No robot can pass through the narrow corridor if either start s or target t is occupied. In particular, the area shown in red blocks movement to the start s.

ple workspace. The reason we discuss multiple algorithms is that it shows different possible approaches for finding a solution. In this thesis we have tried many possible routes for solving the problem, both successful and unsuccessful, and in this chapter we present the results. We think that showing the different approaches will provide the reader with a better understanding of the problem. In the conclusion in Chapter 5 we will give a short comparison between the different algorithms.

3.1 Preliminaries

Before discussing the algorithms for solving the motion planning problem, it is useful to first introduce some concepts.

Let $F_i' = F_i \setminus \bigcup_{s \in S} \mathcal{D}_2(s)$ be the portion of the free space which does not intersect with the aura of any start configuration. In other words, F_i' is the portion of F_i after taking the complement with the auras of the start configurations. The subset F_i' can consist of different connected components, given that the aura around a start configuration might intersect the boundary of F_i in more than one connected component, thus splitting F_i into multiple components. The subset F_i' consists of two types of boundaries: the *free* boundary, which is the boundary it shares with the free space $\mathcal{F}$, and the aura boundary, which is the boundary of the aura around a start configuration. Recall that the free space is a closed set and the aura is an open set, thus both boundaries will be closed.

For each target configuration $t \in T$ we define $\mathcal{D}_2'(t) = \mathcal{D}_2(t) \cap F_i'$. In other words, the region $\mathcal{D}_2'(t)$ consists of the free space portion of the aura of t minus the aura of the start configurations in S_i. Recall that the bichromatic separation still allows the auras around start and target configurations to intersect, thus the region $\mathcal{D}_2'(t)$ can potentially consist of multiple

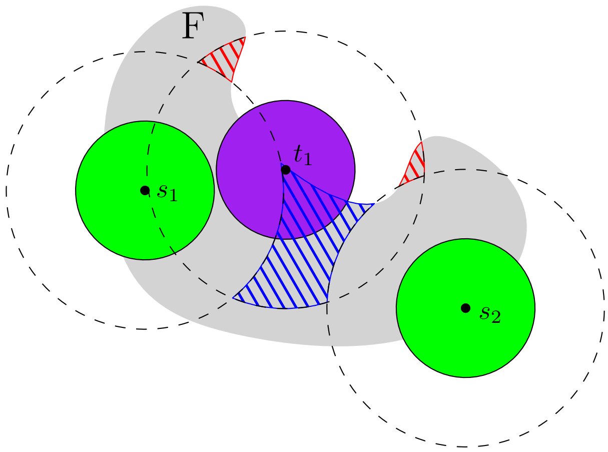


Figure 3.2: A target configuration t_1 is shown, where the components of $\mathcal{D}'_2(t_1)$ which contains t_1 is shown in blue and the remote components are shown in red.

connected components. One of these components of $\mathcal{D}'_2(t)$ will contain t itself. We will define the components of $\mathcal{D}'_2(t)$ that do not contain t as *remote components*. See Figure 3.2 for an illustration of a target whose aura contains multiple remote components.

The key characteristic of a remote component of t is that a path that stays within $\mathcal{D}_2(t)$ between t and a point in a remote component will have to pass through the aura of at least one start configuration. This can present issues when constructing the motion graph, which we will see in Section 3.3. Let R_i be the set of remote components for all target configurations T_i.

Furthermore, let a *blocking area* be a remote component that intersects the boundary of F'_i in more than one connected component. The blocking area cuts F'_i into multiple different connected components. Similarly, a target t which is associated with at least one blocking area will be referred to as a *blocker*. A blocker target might have multiple associated blocking areas, which is also illustrated in Figure 3.3. Let $B_i \subseteq R_i$ be the set of blocking areas for all target configurations.

For a blocking area $b_t \in B_i$, let the *blocking path* be any path $\pi \subset F_i$ which connects b_t to its associated blocker $t \in T_i$. By definition, this path will cross the aura of at least one start configuration, otherwise the blocking area would be connected to the component of $D'_2(t)$ which contains t.

Lemma 3. For a blocking area $b_t \in B_i$ and its associated blocker t, there exists some blocking path π such that $\pi \subset \mathcal{D}_2(t)$.

Proof. This follows from Lemma 2 from the paper by Adler et al. [1], which states that for any $x \in \mathcal{F}$ we have that $\mathcal{D}^*(x)$ is connected, where $\mathcal{D}^*(x)$ is the part of $\mathcal{D}_2(x)$ which is in the same free space component as x. By definition, this means that there exists a path within $\mathcal{D}^*(t)$ between any two points in $\mathcal{D}^*(t)$. The blocker configuration and any associated blocking area are inside $\mathcal{D}_2(t)$ and are both part of same free space component F_i. Thus, there must exist a path connecting the two inside $\mathcal{D}^*(t)$. □

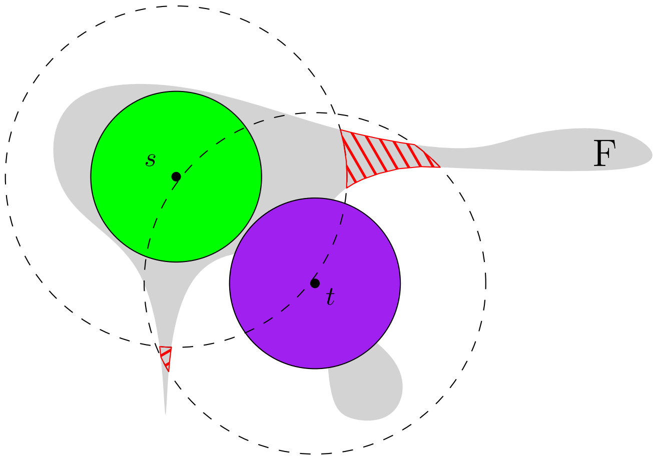


Figure 3.3: An example of a blocker t with multiple distinct blocking areas shown in red.

Lemma 4. There always exists some blocking path π between a blocker x and its blocking area b_x that stays within $\mathcal{D}_2(x)$ and does not intersect the blocking area b_y of some other blocker y.

Proof. This follows directly from Lemma 3 and the fact that $\mu = 4$, since the blocking areas b_x and b_y should lie inside $\mathcal{D}_2(x)$ and $\mathcal{D}_2(y)$ respectively, and $\mathcal{D}_2(x) \cap \mathcal{D}_2(y) = \emptyset$ with $\mu = 4$. Thus, there exists a blocking path π from x to b_x that stays within $D_2(x)$ and therefore cannot cross another blocking area b_y. □

See Figure 3.4 for another illustration of a blocker and its blocking area.

Let $\overline{F_i} = F_i \setminus R_i$ be the portion of the free space component F_i that does not intersect any remote components in R_i. By definition, a blocking area will intersect the boundary of F_i in multiple components. Since some remote components will be blocking areas, the region $\overline{F_i}$ can consist of multiple connected components. Let the maximal connected components of $\overline{F_i}$ be referred to as *residual components*. Note that $\overline{F_i}$ is different from F_i', which was the portion of the free space that does not overlap with the aura of start configurations.

Additionally, let $F_i^* = F_i' \cap \overline{F_i} = F_i \setminus (\bigcup_{s \in S_i} \mathcal{D}_2(s) \cup R_i)$ be the portion of the free space that does not intersect with either the aura of a start configuration or a remote component of a target configuration.

Lemma 5. The subsets of the free space F_i', $\overline{F_i}$, and F_i^*, the free space region of an aura $\mathcal{D}_2$ and the remote components R_i all have complexity $O(m + n)$ and can be computed in $O((m + n) \log(m + n))$.

Proof. The complement of the workspace polygon can be decomposed into $O(n)$ trapezoids by using a vertical decomposition. Let A be the union of these trapezoids and $O(m)$ unit discs centered at the start configurations. Note that all elements of A are pairwise disjoint. The union of the elements in A Minkowski-summed with a unit disc is linear in the number of elements plus the complexity of the elements [9]. Therefore, the region F_i' has complexity $O(m + n)$ and can be generated in $O((m + n) \log(m + n))$.

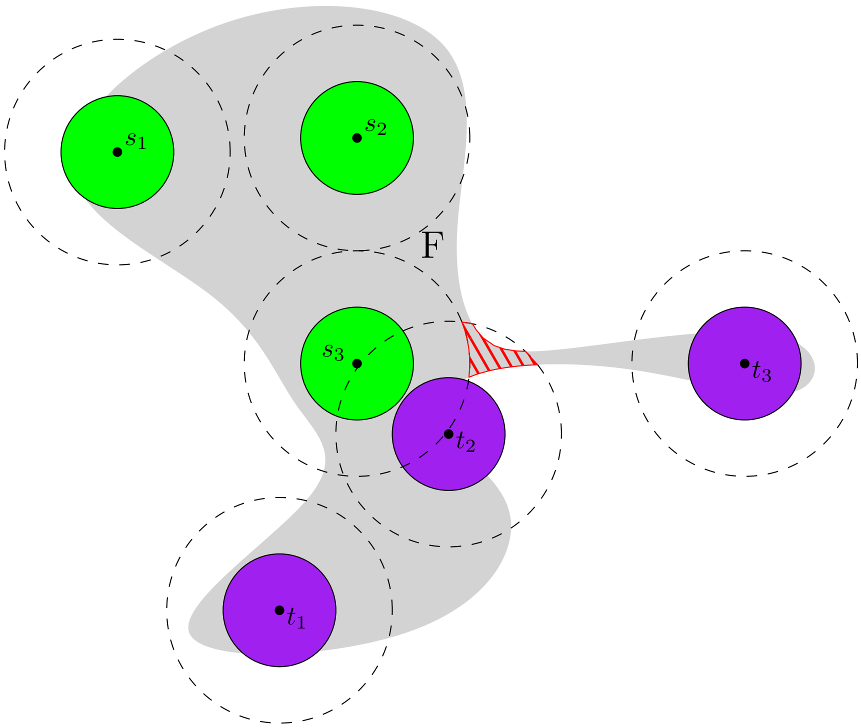


Figure 3.4: An example of a blocker t_2, with the blocking area shown in red. The start configuration s_3 separates the blocking area from t_2.

The free space region of an aura $\mathcal{D}_2$ consists of sections of the free space $F_i \setminus \bigcup_{x \in S_i \cup T_i} \mathcal{D}_2 x$. These sections have complexity $O(m+n)$, using a similar argument as for the free space region F_i'. Thus the free space region of an aura has complexity $O(m+n)$.

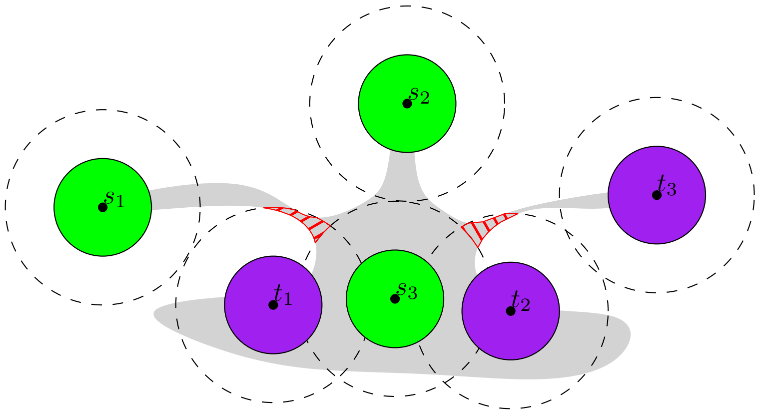
Using the same logic, the remote components R_i and the free space subsets $\overline{F_i}$ and F_i^* all use segments from existing sets with complexity $O(m+n)$, therefore their complexity is also bounded by $O(m+n)$ and can be computed in $O((m+n)\log(m+n))$. $\square$

3.2 The blocking area graph

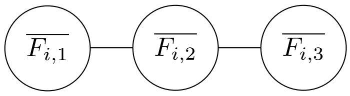
In this section we will introduce a graph structure based on residual components and blocking areas, as defined in Section 3.1.

Let $H_i = (V_i^H, E_i^H)$ be a graph whose vertices V_i^H equal the residual components of $\overline{F_i}$. Recall from Section 3.1 that $\overline{F_i} = F_i \setminus B_i$ is the portion of the free space component F_i that does not intersect any blocking areas in B_i. An edge is drawn between two distinct residual components $v_1, v_2 \in V_i^H$ if they are separated by a single blocking area $b_t \in B_i$ where the associated blocker t resides in either v_1 or v_2. See Figure 3.5 for an example of the blocking area graph.

Given the definition of a blocking area and the separability constraints, all configurations reside in residual components of $\overline{F_i}$. It is important to note that a single blocking area in B_i can divide $\overline{F_i}$ into more than two connected components, see Figure 3.6 for instance.



(a) An example with two blockers t_1 and t_2, with their associated blocking areas shown in red



(b) The corresponding blocking area graph H_i.

Figure 3.5: An example of the blocking area graph for an motion planning instance.

However, the definition of an edge in H_i requires the associated blocker to be in one of the two components. Therefore, such a blocking area will not result in a cycle in H_i.

Lemma 6. Any blocking area $b_t \in B$ shares a boundary with the residual component containing t.

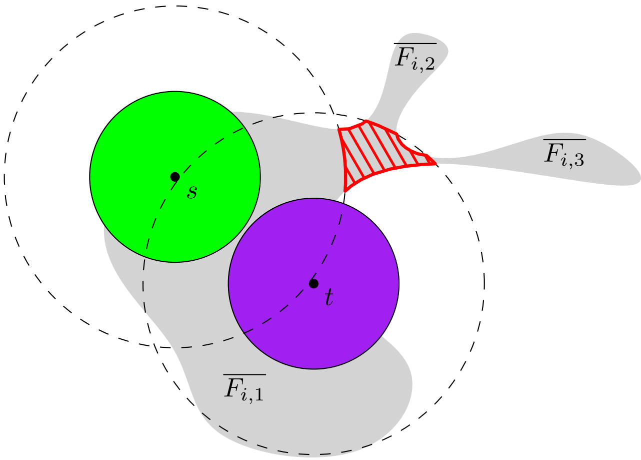
Proof. Follows directly from Lemma 4. There exist a blocking path π which connects b_t to t which does not cross any other blocking area. By definition of a residual component, the blocking area must therefore be adjacent to the component containing t. □

Lemma 7. The blocking area graph H_i is connected.

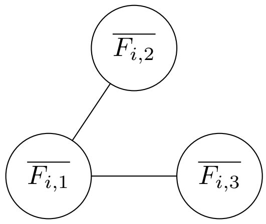
Proof. Assume for contradiction that H_i is not connected. Then there must be distinct residual components $x, y \in V_i^H$ which are not connected by a path in H_i. Take arbitrary points $p_x \in x$ and $p_y \in y$. Since both p_x and p_y lie in F_i and F_i is connected, there exists a path $\pi \subset F_i$ which connects p_x with p_y. Additionally, given the monochromatic separation $\mu = 4$, the blocking areas of distinct blockers do not intersect, therefore π will alternate between a blocking areas and residual components.

Take an arbitrary blocking area $b_t \in T$ that is traversed by π, which is associated with a blocker target t. Let v and w be the residual components adjacent to b_t that π traverses. We now argue that v and w are connected in H_i.

The blocker t must be in a residual component adjacent to b_t by Lemma 6. Let z be the residual component containing t. If z is equal to either v or w, meaning the blocker t resides in either v or w, then by definition there must be an edge between v and w as well, therefore they are connected. The other possibility is if z is not v nor w, meaning t in a third residual component not equal to v or w. In that case, there must be an edge between v and z and between z and w, thus v and w are connected through z. Applying this logic to all blocking



(a) The example with blocker t and the three residual components.



(b) The associated blocking area graph.

Figure 3.6: An instance where a blocker cuts the free space in more than two components.

areas along π between residual components x and y, we can conclude that x and y must be connected. □

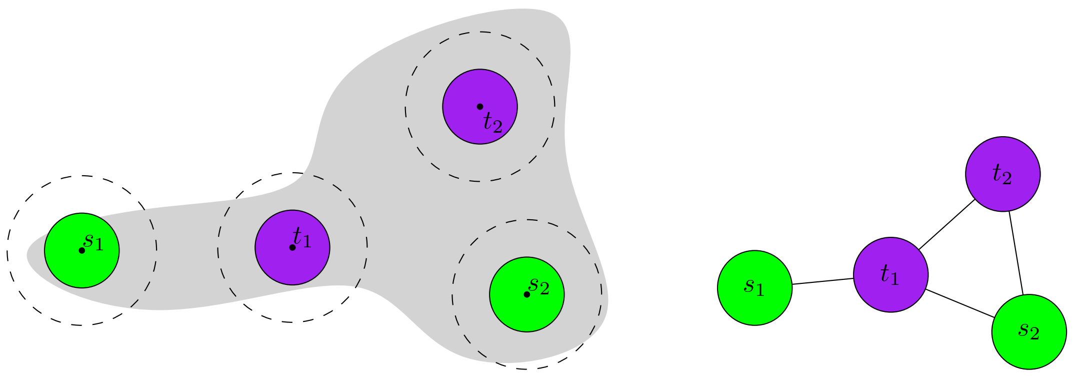
Lemma 8. The blocking area graph H_i is a tree.

Proof. Assume the graph $H_i = (V_i^H, E_i^H)$ is not a tree. By Lemma 7 we know that H_i is connected. For H_i not to be a tree it must therefore contain a cycle. Thus, there must exist some circular set $v_1, \ldots, v_k$ of distinct vertices in the graph, where $v_i \in V_i^H$ and $(v_i, v_{i+1}) \in E$ for $i \in 1, \ldots, k$ and $k > 2$. Given the cycle, there must exists some circular curve $\pi \subset F_i$ through the nodes $v_1, v_2, \ldots, v_k$, starting and ending in the same configuration and intersecting k blocking areas.

Let A be the area enclosed by π. Given that the workspace $\mathcal{W}$ is simple and $\pi \subset F_i$, we have that $A \subset F_i$. Given the monochromatic separation $\mu = 4$, the blocking areas π intersects are disjoint. since we assumed that $v_1, \ldots, v_k$ correspond to different residual components and π should cross each blocking area only once. However, in that case A cannot contain more than one residual components, since the disjoint blocking areas can only split A if π intersects the blocking area in more than one location. We arrive at a contradiction, thus the graph H_i must be a tree. □

3.3 The motion graph

The motion graph $G_i = (V_i^G, E_i^G)$ is a graph where the vertices represent the start and target configurations in $S_i \cup T_i$ and the edges represent a path between "adjacent" configurations. Edges between configurations should represent paths through the free space which only intersect the aura of the source and destination configurations, such that a robot can move from a source configuration along the path unobstructed to the destination configuration (as long as the destination is unoccupied). See Figure 3.7 for a visualization of a simple motion graph.



(a) Small motion planning instance. **(b)** The motion graph.

Figure 3.7: Example of a motion graph capturing the adjacencies of the configurations.

The idea is to always have robots positioned on either a start or target configuration and have them move one at a time between these configurations using the motion graph. This restricts robots to $2m$ possible positions, which greatly reduces the problem's complexity since we can ignore the free space once the motion graph is constructed. We would then like to show that the problem can still be solved efficiently using the motion graph. Ideally, we would create a (connected) motion graph for the start and target configurations, and then solve the problem by moving robots along this graph such that at the end all target configurations are occupied.

However, due to the lack of bichromatic separation, there can be cases where a start and target configuration are too close such that both cut the free space, see for example Figure 3.1 or Figure 3.3. In this case, any path through that part of the free space will have to intersect the aura of both configurations. This causes problems for creating a connected motion graph on all start and target configurations, since we require that an edge is always traversable as long as the destination vertex is unoccupied. In other words, we would like to create paths that do not cross the aura of other configurations except the source and destination vertices of an edge, but this is not always possible while keeping the motion graph connected.

See Figure 3.8 for an instance where a connected motion graph cannot be created. In the figure, for the configuration s_1 there does not exist a path to any other configuration which does not cross the aura of a third configuration. Therefore, if we want to create a motion graph we cannot connect s_1 to any other vertex.

To deal with this issue, we will differentiate between regular *unblockable* edges and *blockable* edges, the latter of which will be colored red. For blockable edges, we relax the constraint that an corresponding path through the free space is only allowed to cross the aura of its source and destination configuration. Instead, we allow these paths to also cross the blocking areas in B_i. We will show that, with this relaxation, we can construct a connected motion graph for all start and target configurations.

3.3.1 Construction

We will describe one method for constructing the motion graph $G_i = (V_i^G, E_i^G)$. The vertices V_i^G are the set of start and target configurations $S_i \cup T_i$. For the rest of this chapter we will

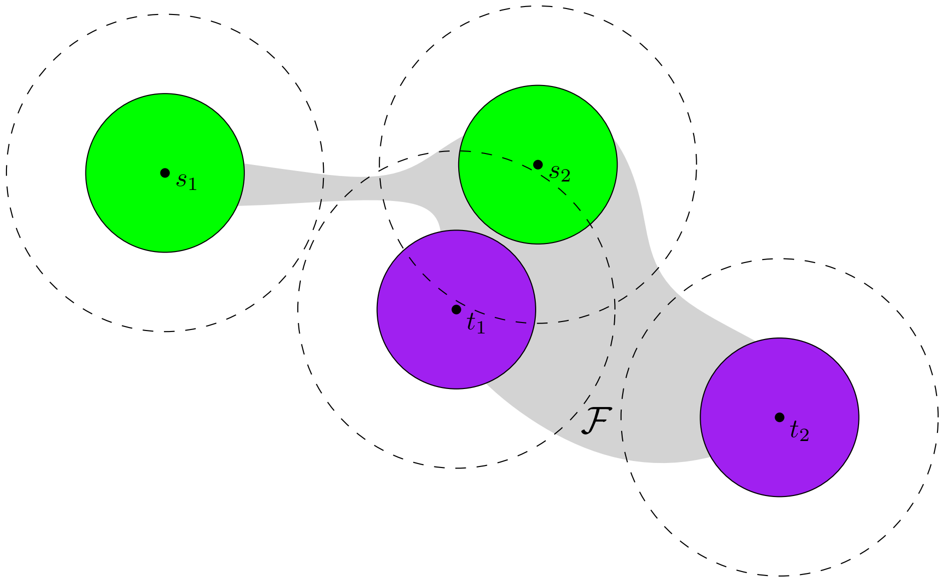


Figure 3.8: An instance where the lack of bichromatic separation ($\beta < 4$) makes it impossible to create a path that connects the start s_1 to any other configuration with a path that does not intersect any other aura.

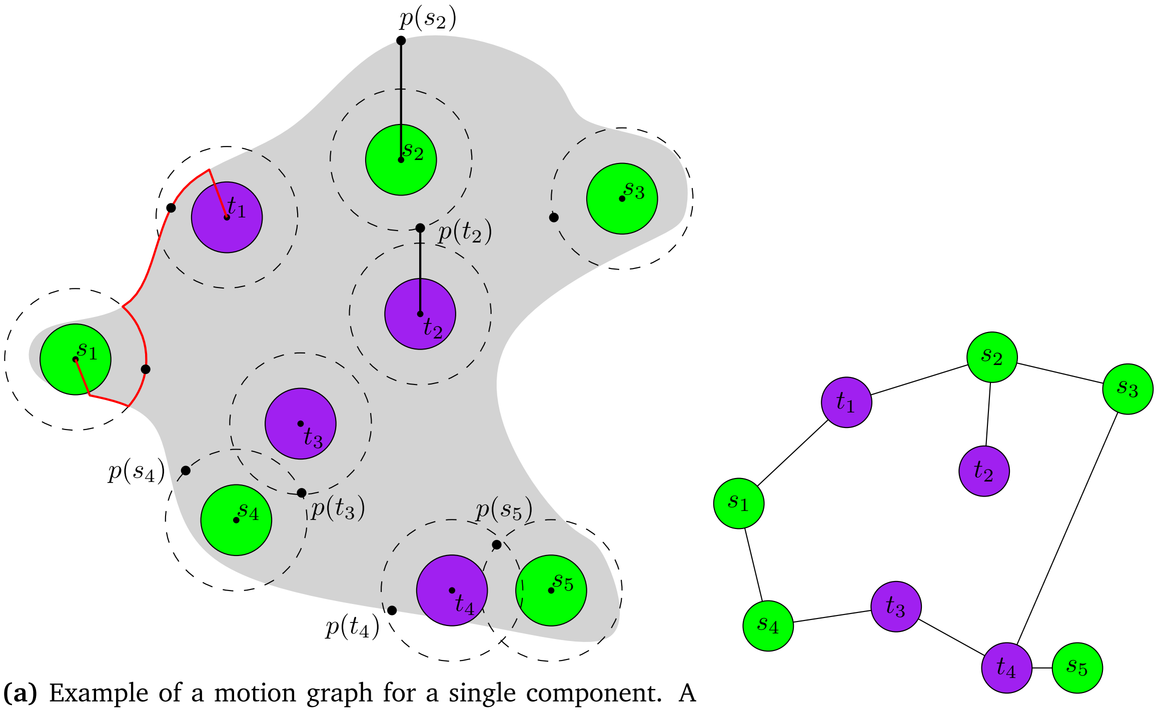
therefore use the terms configuration and vertex somewhat interchangeably, though they are not strictly the same. The edges E_i^G in the motion graph are generated as follows:

Recall from Section 3.1 that $F_i^* = F_i \setminus (\bigcup_{s \in S_i} \mathcal{D}_2(s) \cup R_i)$ is the portion of the free space that does not intersect with either the aura of a start configuration or a remote component of a target configuration. We assume that the subset F_i^* was split into k connected components, $F_{i,1}^*, \ldots, F_{i,k}^*$. These components contain all target configurations in T_i, since the aura of a start configuration cannot contain any target by the bichromatic separation $\beta = 2$, and a blocking area cannot either by definition. First, we will create edges for each connected component of F_i^*. Second, blockable edges are added that traverse blocking areas to make sure the motion graph is connected.

Take a single maximal connected component $F_{i,j}^* \subset F_i^*$. Although $F_{i,j}^*$ is connected, it can contain holes due to free-floating start configurations. For the boundary $\delta(F_{i,j}^*)$ we will create an ordered, circular list Λ_j. For each target configuration $t \in t(F_{i,j}^*)$ which intersects the outer boundary $\delta(F_{i,j}^*)$ we pick a set of representative points P_t on each connected component of $\delta(F_{i,j}^*) \cap \mathcal{D}_2(t)$.

By definition, the subset F_i^* does not contain any remote components, thus there must always exists a path from the target configuration t to each representative point $p \in P_t$ which stays within $\mathcal{D}_2(t)$ and does not intersect the aura of another start or target configuration. The representative points for each target which intersects the outer boundary of $F_{i,j}^*$ are stored in Λ_j based on their ordering along the boundary.

Next, we handle the target configurations in $F_{i,j}^*$ that do not intersect the outer boundary $\delta(F_{i,j}^*)$ as well as the start configuration that correspond to holes in $F_{i,j}^*$. For each such configuration x, we shoot a ray vertically upwards until it either hits the outer boundary of $F_{i,j}^*$ or the aura of another configuration. Let p_x be the first intersection point with either



(a) Example of a motion graph for a single component. A path between adjacent targets is shown in red.

(b) The associated motion graph.

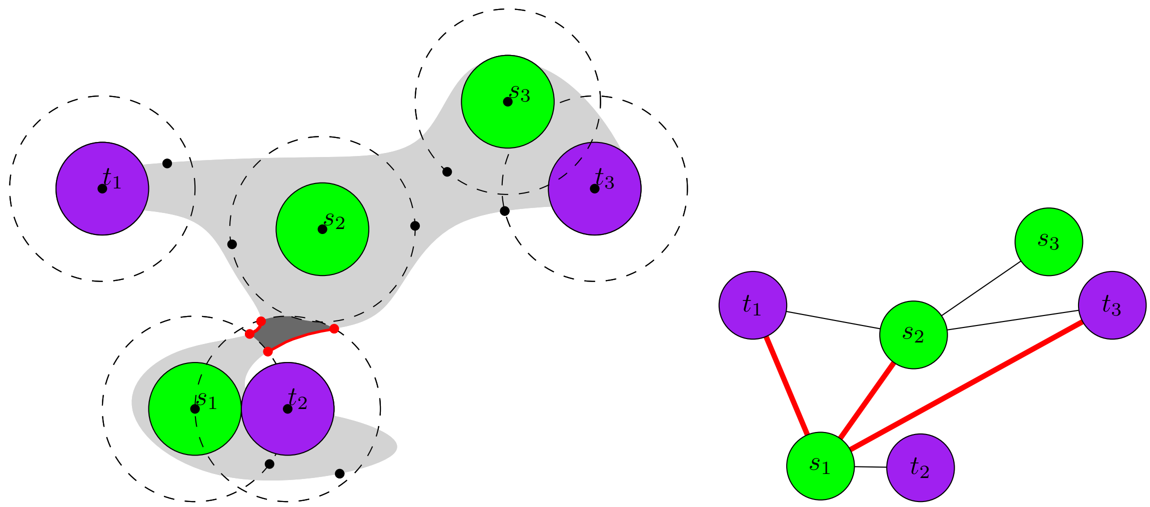
Figure 3.9: Example of the motion graph for a single component $F_{i,j}^*$.

the outer boundary or the aura of another configuration. If it hits the outer boundary, the intersection point p_x is added to Λ_j as a representative point of x.

Otherwise, if it hits the aura of another configuration y, we add an edge in the motion graph between x and y. Given the definition of F_i^*, if the ray from x does not hit the outer boundary then the configuration y is either a free-floating start or a target. Furthermore, if y is a target configuration, then we know that the intersection point p_x cannot lie on a remote component. For both cases, there is an unobstructed path from p to the configuration y.

Finally, the start configurations whose boundary touches the outer boundary of $\delta(F_{i,j}^*)$ are handled. Take such a start configuration s. Pick a single representative point p_s on the intersection of the outer boundary of $F_{i,j}^*$ and the aura of s. If there is an unobstructed path from s to p_s, we add the representative point p_s to Λ_j based on its ordering along the boundary. Otherwise, if the path must pass through the aura of a target t and the intersection point does not lie in a remote component, we can add an edge directly between s and t. In the last case, when a remote component obstructs any path from the configuration s to its representative point p_s, we will ignore s until we handle the blocking areas separately later.

Now, edges are added to E_i^G between any vertices whose representative points are adjacent in Λ_j. Since a configuration might have multiple representative points in Λ_j, self loops might be introduced but these can safely be ignored. In the same vein, multiple edges could be created between the same two vertices but we can remove all but one to decrease complexity. This concludes the construction for a single component $F_{i,j}^* \subset F_i^*$. See Figure 3.9 for an illustration on the motion graph for a connected component of F_i^*.



(a) Blocking area shown in gray, with free space boundary in red.

(b) The associated motion graph, with new edges in red.

Figure 3.10: Example of how a blocking area is handled with regards to the motion graph.

Finally, we have to incorporate paths through the blocking areas into our motion graph since, as mentioned, these blockable edges are necessary to ensure the motion graph is connected. Take a blocking area $b_t \in B_i$. We will add blockable edges that cross through b_t to the motion graph as follows:

Take the boundary $\delta(b_t)$ of the blocking area. The boundary is made up of free boundary, namely portions of the boundary of the free space, and two types of aura boundary, namely the boundary of the aura of t and the boundary of the aura of starts configurations in S_i. For each section of free boundary of $\delta(b_t)$ we take the two endpoints, call them x and y, and add edges to the motion graph based on the boundary type of the adjacent component. For each endpoint we pick "adjacent" configuration(s) and we will add an edge between the adjacent configuration of endpoints x and y.

If the endpoint lies on the boundary of a start configuration, this configuration is picked as adjacent. Otherwise, if the endpoint lies on the boundary of the blocker's aura, then the endpoint must lie on the boundary of a component of F_i^*, call it $F_{i,j}^*$. We can find the "adjacent" configuration(s) of the endpoint using the associated Λ_j of $F_{i,j}^*$. We can find up to two configuration that are adjacent in Λ_j with respect to the endpoint (one adjacent configuration in both directions of $\delta(F_{i,j}^*)$). We then add an edge pairwise between the adjacent configurations of x and those of y.

For each blocking area, the adjacent vertices of the endpoints of each connected portion of its free space boundary are found and are then connected with an edge. This procedure might result in multiple edges between the same two vertices, however we can safely remove all but one to reduce the complexity of G_i. See Figure 3.10 for an example of the procedure for blocking areas.

A special case is when Λ_j is empty for an adjacent component $F_{i,j}^*$. In that case, if b_t is the only blocking area adjacent to the outer boundary of $F_{i,j}^*$ we can safely ignore it since it does not contain any configuration in $S_i \cup T_i$. Otherwise, the component $F_{i,j}^*$ must share a boundary

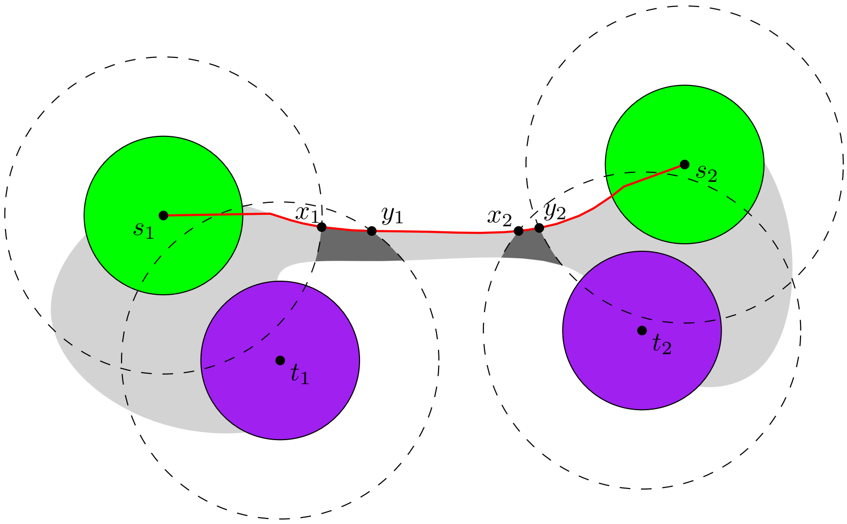


Figure 3.11: The special case when a blockable edge is added across two blocking areas.

with another blocking area $b_{t'}$. Following the free space boundary of $F^*_{i,j}$ and the blocking areas b_t and $b_{t'}$, we can add a blockable edge for the "outer" endpoints of the two blocking areas. This will result in a blockable edge added which has two associated blockers. See Figure 3.11 for a illustration of this special case.

3.3.2 Translating graph edges to free space paths

Edges in the motion graph should correspond to paths in the free space in order to generate a valid motion plan from edge traversals. The paths follow relatively straight-forward from the construction in subsection 3.3.1.

First, we describe the paths for the edges created for the single component $F^*_{i,j} \subset F^*_i$. Take an edge $e \in E^G_i$ between vertices x and y. Its path will consists of (up to) three parts: The motion from x to its representative point p_x, the motion from p_x to the representative point p_y of y, and the motion from p_y to y. The motion to and from representative point p_x consists of either the vertical ray with which p_x was generated or a path to the boundary of $\mathcal{D}_2(x) \cap F^*_{i,j}$ followed by a path along this boundary to p_x. Motion between representative points is along the boundary of $F^*_{i,j}$. If an edge was added directly (without using Λ), for example when a ray intersects a free-floating configuration, then the path only consists of the motion from x to the intersection point p_x and the motion from p_x to y.

For the blockable edges that traverse the blocking areas, the paths are constructed similarly. Take a blocking area $b_t \in B_i$. Each connected portion of free space boundary of b_t will contribute up to four edges between the adjacent vertices of the two endpoints. Take an edge between vertices x and y that was generated for a portion of the free space boundary of b_t with endpoints a and b. A path for these edges consists of the motion from the configuration x to the endpoint a, the motion between the endpoints a to b, and the motion from the other endpoint b to the configuration y. The paths between a configurations and an endpoint are generated similar to the edges above, while the motion between endpoints simply follows the free space boundary of b_t between a and b.

3.3.3 Correctness

In this section we will show that the motion graph constructed in subsection 3.3.1 is valid and has the properties we desire. For the algorithms discussed in the following sections it is crucial that the motion graph is connected, such that there exists a path in the motion graph between any two configurations. Additionally, it will be useful to show when two configurations are connected by only unblockable edges, since then the algorithm can ignore the influence of blockers.

Lemma 9. All paths in the motion graph created for a component $F^*_{i,j} \subset F^*_i$ are unblockable.

Proof. We argue that each portion of a path created in $F^*_{i,j}$ between two configurations x and y is unblockable, meaning it does not cross the aura of any configuration besides x and y.

The motion from a configuration x to its representative point p_x cannot be blocked by its construction. Given bichromatic separation $\beta = 2$, the configuration x itself is not inside the aura of another configuration. If the path from x to p_x crosses some other aura at any point, then the construction will update p_x to the intersection and connect x to this configuration. Thus, the path from a configuration to its representative point cannot be blocked by another configuration.

Given the monochromatic separation and the definition of F^*_i, a point on the boundary of a $F^*_{i,j}$ can only ever be in the aura of a single target configuration. Additionally, for each segment of the boundary that intersects a target's aura we choose a representative point. Therefore, the portion of the boundary between representative points that are adjacent on Λ_j can only intersect the aura of those two representative points. As a result, the motion along the boundary of $F^*_{i,j}$ between adjacent representative points p_x and p_y cannot intersect the aura of a third configuration not equal to x or y, and thus it is unblockable. □

Lemma 10. There always exists an unblocked path in the motion graph between two configurations that are inside the same residual component.

Proof. Take two configurations $x, y \in V^G_i$ that are inside the same residual component, as defined in Section 3.1. This means that there exists a path π through the free space which does not cross any blocking area. If x and y reside in the same component of F^*_i, then by Lemma 9 there exists an unblockable path and we are done. Otherwise, if x and y are in different components of F^*_i, then π must cross some start configurations in S_i that split F^*_i into multiple components. Let $s_1, \ldots, s_k$ be the start configurations that π intersects. Then all adjacent configurations in the sequence $x, s_1, \ldots, s_k, y$ will share a residual component that they either reside in or have a boundary with. By Lemma 9, there must therefore exist an unblockable path between each adjacent configuration in this sequence. Using those individual paths, the vertices x and y are connected with a path that only uses unblocked edges. □

Lemma 11. The motion graph G_i is connected.

Proof. By Lemma 10 we know there exists path in the motion graph between any configurations in the same residual component. By Lemma 7, the blocking area graph H_i is connected. The procedure done for each blocking area will ensure that two residual components that are adjacent in H_i also have edges in the motion graph between two configurations in either component. Combining these results, the motion graph G_i must be connected. □

3.3.4 Complexity analysis

Lemma 12. The number of edges $|E_i^G|$ in the motion graph G_i is bounded by $O(m)$.

Proof. The construction shown in subsection 3.3.1 will create edges for every connected component of F_i^* and then additional edges are added for each blocking area. We will argue that the edges added for both parts is bounded by $O(m)$.

For each component of $F_{i,j}^* \subset F_i^*$ the number of edges created is bounded by the target configurations that reside in it plus the start configurations that share a border with it. A start configuration can only add one or two edges per component of F_i^* it borders. The edges that a target configuration contributes is slightly more complicated to analyze, since for a target t we add a representative point for each connected component of $\delta(F_{i,j}^*) \cap \mathcal{D}_2(t)$.

However, the number of connected components of $\delta(F_{i,j}^*) \cap \mathcal{D}_2(t)$ is constant, since a connected component of $\delta(F_{i,j}^*) \cap \mathcal{D}_2(t)$ will have to intersect $\delta(\mathcal{D}_2(t))$ in two points. Each point $x \in F_i \cap \delta(\mathcal{D}_2(t))$ in the free space requires that $\mathcal{D}_1(x) \cap \mathcal{O} = \emptyset$. We can only fit a constant number of unit circles on $\delta(D_2(t))$, therefore there can only be a constant number of segments of $\delta(\mathcal{D}_2(t))$ that are in the free space F_i. Thus, the number of connected components of $\delta(F_{i,j}^*) \cap \mathcal{D}_2(t)$ is constant.

For a similar reason, a start configuration only borders a constant number of components of F_i^*. So the number of edges created for all components of F_i^* is bounded by $O(m)$.

A blocking area adds at most four edges to E_i^G per segment of free space boundary. By definition, a blocking area will intersect the free space boundary in at least two distinct connected components. However, a target can only intersect the free space boundary in a constant number of connected components. Therefore, a blocking area has a constant amount of free space boundary segments and all blocking areas in B_i will add $O(m)$ edges. □

Lemma 13. A path can be found in $O(m)$ between any two vertices through the graph G_i and the corresponding path in the free space will have complexity $O(m+n)$.

Proof. From Lemma 12, we know that the edges are bounded by the m robots. Using a simple path-finding algorithm, like a breadth-first search (BFS), a path can be found between two vertices in $O(|V_i^G| + |E_i^G|) = O(m)$.

Any path between two vertices in the motion graph will take at most $m-1$ edges. The corresponding path in the free space, after the translation discussed in subsection 3.3.2, will consist of the free space boundary and the portion between a configuration and a representative point. Crucially, a section of the free space boundary will only be taken once in any path, while the path between a configuration and its representative point is taken either once or twice (e.g. potentially in both directions). Therefore, using Lemma 5 the entire path through the free space will have complexity $O(m+n)$. □

Lemma 14. The motion graph G_i can be created in $O(mn + m^2)$.

Proof. As argued in Lemma 12, the number of edges in the motion graph is bounded by $O(m)$. Each edge can be calculated using simple procedures described in subsection 3.3.1, that are dependent on the components of F_i^* and the blocking areas B_i. By Lemma 5, the subset F_i^*

of the free space and the set of blocking areas B_i both have complexity $O(m + n)$. Thus, the entire procedure is bounded by $O(mn + m^2)$. □

3.4 Greedy algorithm

In this section, we discuss a greedy algorithm for solving the motion planning problem. The idea is to handle any issues with blocked edges locally by moving the robot at the blocker configuration away to make traversal of a blocked edge possible. The approach uses the motion graph, discussed in Section 3.3, and treats it as an unlabeled pebble game similar to how this is done in Adler et al. [1].

In the unlabeled pebble game there are indistinguishable "pebbles" that occupy vertices in the motion graph G_i, and which can move one-at-a-time along an edge if the destination is not occupied. At the start of the pebble game each start configuration contains a pebble. A solution to the pebble game is then a series of moves along edges such that at the end all target vertices contain a pebble. Kornhauser [11] proved that the unlabeled pebble-motion problem is always solvable and can be found in $O(S_i^2)$ time.

However, if there are blockable edges in the motion graph then extra care is necessary to ensure the edges are only traversed if the associated blocker configuration is not occupied by a robot. The $O(S_i^2)$ running time algorithm will therefore need to be adapted to handle blockable edges properly. One way of this issue is, whenever a blockable edge needs to be traversed, to move away an associated blocker target to another position. The greedy approach taken is thus to try to handle blockers "on-the-fly" by moving the robot at the blocker position to another unoccupied vertex in the motion graph. In this section we explore whether there is a simple and efficient method that can always handle blockers.

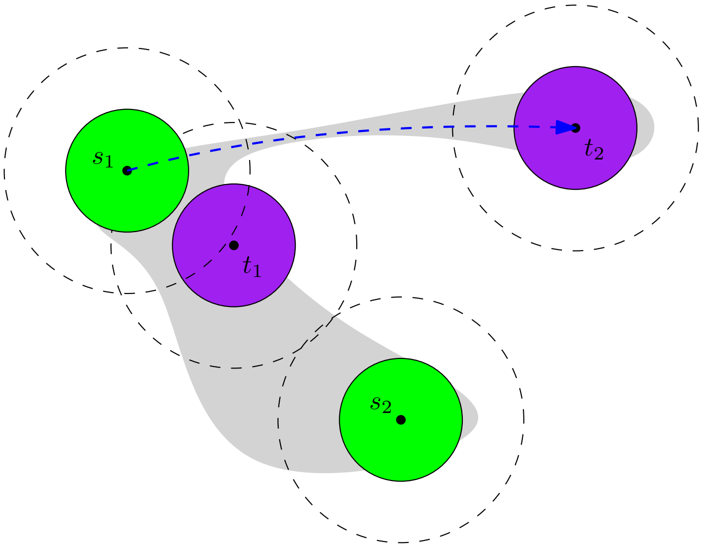
3.4.1 Handling blocked edges

The pebble moves along edges in our motion graph are not as straight-forward as in the standard pebble game. As mentioned, the blockable edges need extra attention when moving pebbles along the motion graph. In this section we try to show how to handle blockers on-the-fly.

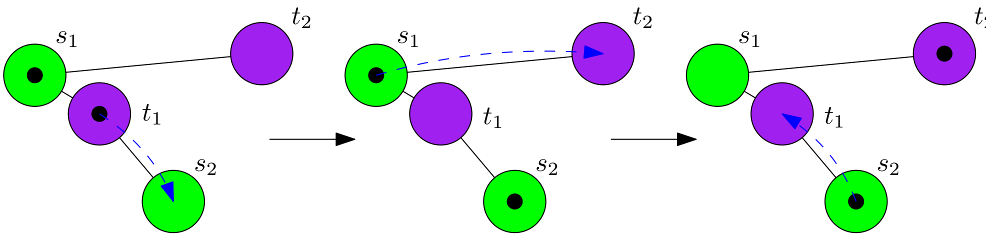
Assume that we need to make a series of single pebble moves along a path in our pebble graph. Let s be the start vertex of our path and t the target vertex. The path from s to t might have to use blockable paths in the motion graph. The associated blockers will be handled one by one. We describe how to solve the blocking in two cases:

If the residual component of H_i that the blocker t_b resides in contains at least one unoccupied vertex x, we can make a series of pebble moves along the path from t to x. Since both the blocker and the unoccupied configuration are in the same residual component, by Lemma 10 there is an unblocked path between the two vertices. The vertex x might also be a blocker, so we have to be careful that we do not move the blocking robot to another blocker of the s-t path. See Figure 3.12 for an illustration of this procedure.

Otherwise, if the residual component of H_i containing the blocker does not contain an unoccupied node, there is unfortunately little more to do than to reverse pebble moves up to



(a) A simple instance of a blocker t_1 that obstructs movement between s_1 and t_2.



(b) The procedure for moving a robot at blocker t_1 to allow movement across the edge from s_1 to t_2. Afterwards, the robot at s_2 moves back to t_1.

Figure 3.12: An example of how a robot at a blocker target configuration can be moved to another node in the same residual component of H_i to allow traversal of a blocked edge.

that point until the blocker is no longer occupied. Moving the blocker to a vertex outside its residual component might run into additional blocked edges. Then perform the motion along the s-t path, and finally redo the previous pebble moves in order.

3.4.2 The algorithm

See Algorithm 1 for pseudocode on the greedy algorithm. The algorithm will continuously pick an unoccupied target and try to occupy it by finding a nearby occupied start. Any blocked edges on the path between the start and target will be handled as described in subsection 3.4.1.

Algorithm 1 Pseudo code for the greedy algorithm for a single free space component F_i.

```
procedure MotionPlanningGreedy(W, S_i, T_i)
    Calculate free space F_i
    Compute blocking areas B_i and blocking area graph H_i = (V_i^H, E_i^H)
    Create the motion graph G_i = (V_i^G, E_i^G)
    while exists unoccupied target t do
        Find nearest occupied start s (BFS)
        Calculate path π from s to t in motion graph G_i
        for each blocker b corresponding to a blocked edge in π do
            Handle robot at blocker b by either:
                • If possible, move b to unoccupied vertex x in same residual component.
                  Make sure x does not block π as well.
                • Else, reverse moves until b no longer occupied.
        end for
        Make pebble move(s) from s to t
        for each pebble move reversed do
            Execute pebble move, handling blockers same as above.
        end for
    end while
end procedure
```

3.4.3 Analysis

The correctness of the greedy algorithm is dependent on its ability to handle blockers locally. In the first case, we deal with an occupied blocker by moving the robot to another configuration in H_i. However, the pebble graph can be in a state where there is no clear procedure to unblock. For such instances, the second case tries to resolve the blocking by reversing the pebble moves done up to that point. Since in the initial state all target configurations are empty, this will eventually get to a state where the blocker is no longer occupied. Once such a state is found the pebble move is executed, after which all moves which have been reversed have to be redone.

However, the destination t of the pebble move(s) might also block some of the moves that were reversed. As a result, this procedure has the potential to cause an endless cycle, where

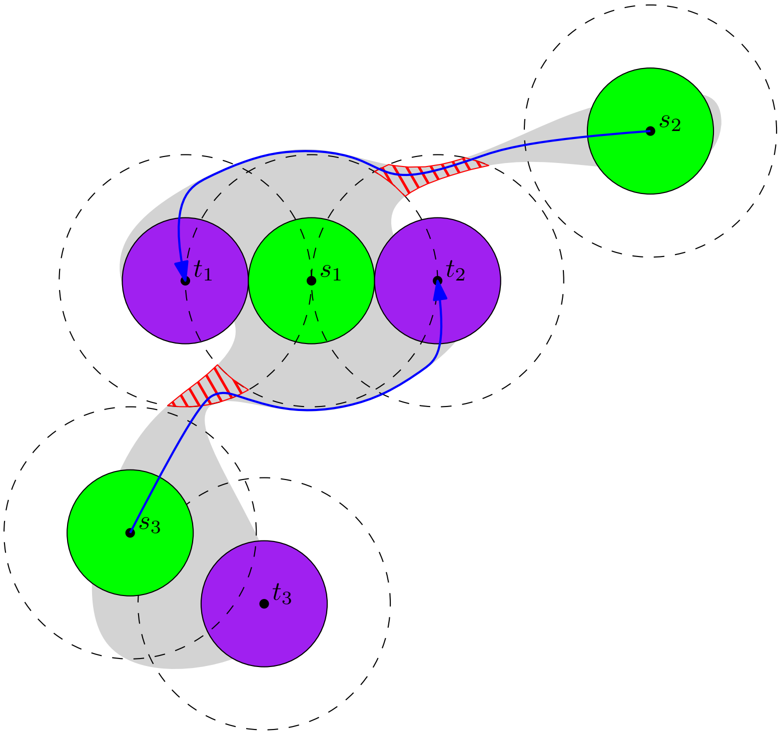


Figure 3.13: A instance where there can be a potential cyclical blocking, if s_2 is matched to t_1 and s_3 is matched to t_2. In red the blocking area and in blue potential paths that cause a cyclical blocking.

moves are continuously reversed and redone in different orderings. If there exists an instance where the greedy algorithm uses a start-target matchings whose paths form a *cyclical blocking*, meaning each target blocks the next path in the cycle, the algorithm might not terminate. Formally, there are pairs $(s_1, t_1), \ldots, (s_k, t_k)$ fin the matching or some k, where a target t_{j-1} blocks the path in the motion graph between (s_i, t_i) cyclically for some k. An example can be seen in Figure 3.13.

Theorem 1. For a single connected component $F_i \subset \mathcal{F}$ containing an equal number of start and target configurations, the greedy algorithm will always find a solution to the unlabeled motion planning problem for unit-disc robots in a simple workspace, when assuming $\mu = 4$ and $\beta = 2$ and there do not exist any start-target matching which contain a cyclical blocking.

Proof. If no start-target matching exists which contains a cyclical blocking, that means that for each matching the algorithm picks there must exist some (topological) ordering which avoids blockings. Thus, when executing the matching in the topological ordering the solution will never encounter pebble moves across blocked edges where the blocker is occupied.

The greedy algorithm solves an arbitrary start-target pair, which means it does not necessarily solve a matching in this topological ordering. In the worst case, no blockings it encounters can be locally resolved and the algorithm will always have to perform the second case. This requires reversing/redoing all previous moves for each start-target pair until they are in topological ordering. However, by always moving a start-target move to the first position whenever we detect a blocking, this procedure is guaranteed to eventually find the topological ordering. □

Thus, the greedy algorithm is only guaranteed to find a solution when there does not exists a matching for the problem with a cyclical blocking. However, instances exist where matchings contain cyclical blockings, see Figure 3.13. This issue was the reason for us to abandon the greedy approach in favour of dealing with the matching and blockers more robustly. In Section 3.5 it is shown how to calculate a matching which does not have any cyclical blockings.

It could be that there exists some greedy method of matching start/target vertices that would avoid any cyclical blockings. There can potentially be made a case that you would never get into a cyclical blocking, for example by always picking the closest start target pairs. Perhaps there also exists a more clever way of resolving local blockings which does not run into this issue. However, we decided to not pursue this approach any further, and instead focus on less greedy methods. An additional factor is that even without cyclical blockings, the running time of the greedy algorithm is not very efficient since the algorithm might need to reverse and redo many robot moves whenever it tries to make pebble moves for a new start-target pair. Nonetheless, we hope this section shows the difficulty of the problem and the issues one can encounter when trying to solve the motion graph greedily.

3.5 Matching algorithm

In this section we explore an algorithm based on finding a matching between the start and target configurations $S_i \cup T_i$ and then calculating an ordering for solving each start-target pair. The idea is to order the pairs such that no pebble moves will be blocked when executing the start-target paths in the motion graph. The approach follows from the issue of cyclical blockings in Section 3.4, where it was shown that an arbitrary matching might not always provide a valid solution no matter the order in which it is executed. Thus, in this section we explore whether it is always possible to find a matching where no cyclical blockings occur.

3.5.1 The algorithm

We take a minimal matching M on the start and target configuration $S_i \cup T_i$, where the weight is equal to the number of blocking areas that are minimally traversed for paths between the matched pairs. The idea behind the distance function is to ensure that the matching minimizes the amount of blockings that occur due to blockable edges. Formally, the distance function is defined as follows:

$$w(M) = \sum_{(s,t)\in M} w(s,t) = \sum_{(s,t)\in M} \#\text{blocked areas traversed in path } s \text{ to } t \tag{3.1}$$

After calculating a minimal matching, the algorithm will handle each cyclical blocking by performing a swap. The algorithm finds a start configuration that is located “inside” the cycle. The start configuration is then swapped with an arbitrary target of the cycle. An example of

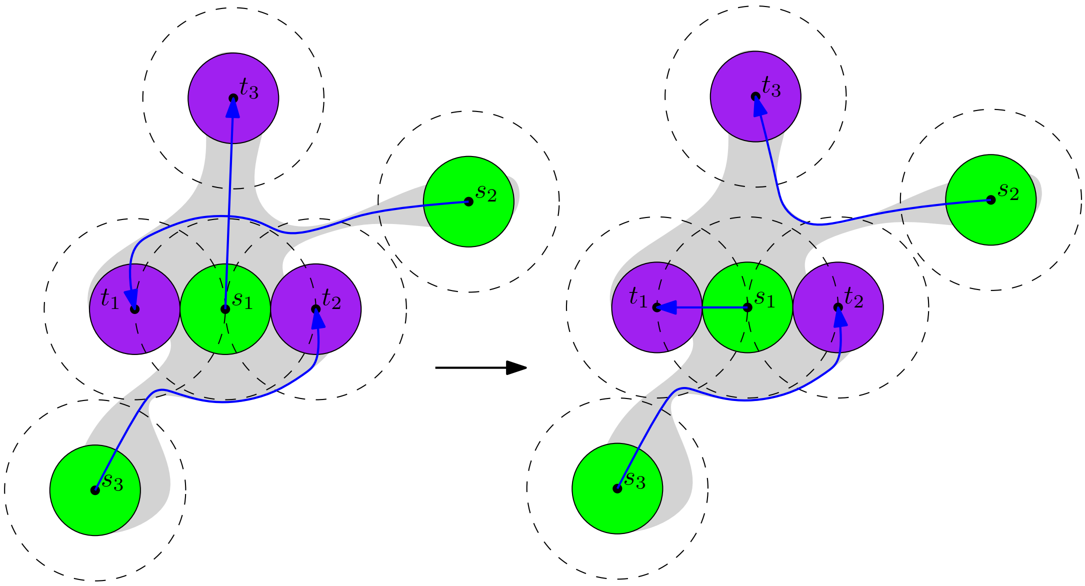


Figure 3.14: An example where a cyclical blocking can be resolved by making a swap in the matching. In blue the matched pairs are shown (the blue lines give an indication of the motion plans, but are not entirely accurate with the actual motion graph construction).

such a swap is shown in Figure 3.14. See subsection 3.5.2 for the proof why this removes the cyclical blocking. Once all cyclical blockings are removed, there exists a topological ordering to solve each matched pair which will not cause any blockings.

See Algorithm 2 for the pseudo-code.

3.5.2 Correctness

The matching algorithm attempts to find a matching which does not contain any cyclical blockings. Initially, the minimal matching is calculated according to the weight function in Equation 3.1, which minimizes the blocking areas traversed. This minimal matching might still contain cyclical blockings, which will need to be removed by making swaps between pairs of the matching. We will argue that the algorithm can always make a valid swap which will remove a cyclical blocking and therefore the algorithm will find a matching that does not contain any cyclical blockings.

Lemma 15. The minimal matching M cannot have paths that cross a blocking area between the same residual components in opposite directions.

Proof. Assume for contradiction that M does contain two pairs s_1, t_1 and s_2, t_2 whose paths cross two components $\overline{F_{i,j}}, \overline{F_{i,k}} \in V_i^H$ in opposite order. Let b_t be the blocker between $\overline{F_{i,j}}$ and $\overline{F_{i,k}}$. Additionally, let the paths in the motion graph between the pairs have the shape $s_1, \ldots, x_1, y_1, \ldots, t_1$ and $s_2, \ldots, x_2, y_2, \ldots, t_2$ such that $x_1, x_2 \in F_i$ and $y_1, y_2 \in F_j$ and the pair x_1, y_1 is connected in the motion graph by an edge which intersects b_t, similarly for x_2, y_2. We can then make a swap in the matching such that we instead have pairs s_1, t_2 and s_2, t_1. By Lemma 10, there exists an unblocked path between x_1 and x_2, similarly for y_1, y_2. Thus the

Algorithm 2 Pseudo code for the matching algorithm for a single free space component F_i.

procedure MOTIONPLANNINGMATCHING($\mathcal{W}, S_i, T_i$)
  Calculate free space F_i
  Find blocking areas B_i
  Create the motion graph $G_i = (V_i^G, E_i^G)$
  Create initial minimal matching M on start/target pairs $S_i \cup T_i$,
    where the weight: $w(s,t) = $ #blocking areas in $s - t$ path
  Calculate paths for each pair in M and detect blockings
  for each blocking cycle **do**
    Find start configuration s_b inside cycle
    Perform swap with s_b and arbitrary target t of cycle
    Recalculate paths
  end for
  Execute matching in topological order of blockings
end procedure

paths $s_1, \ldots, x_1, x_2, \ldots, t_2$ and $s_2, \ldots, y_2, y_1, \ldots, t_1$ have a strictly smaller weight, since they no longer intersect the blocking area of b_t. □

Lemma 16. For a cyclical blocking $(s_1, t_1), \ldots, (s_k, t_k)$, all targets of the cycle lie in the same residual component.

Proof. Assume for contradiction that the targets do not all lie inside the same residual component. Then there must exist some adjacent pairs (s_j, t_j) and (s_{j+1}, t_{j+1}) in the cycle whose targets lie inside distinct residual components. Let b_t be a blocking area that separates these distinct residual components. Since the blocking area graph H_i is a tree and a targets blocking area is adjacent in H_i by Lemma 6, the path between s_{i+j} and t_{i+j} will traverse b_t. Since we have a cyclical blocking there exists a sequence of blockings which eventually return to block the path between s_j and t_j. Thus, there thus must be some path that returns through b_t. This is a contradiction with Lemma 15, which states that b_t cannot be traversed in opposite direction in the matching M. □

By Lemma 16, we know that all targets $t_1, \ldots, t_k$ lie inside the same residual component, call it $\overline{F_{i,j}}$. Since all paths between the start-target pairs are blocked, all start configurations in the cycle must therefore lie outside $\overline{F_{i,j}}$. No blocking area of the targets $t_1, \ldots, t_k$ is traversed by two different paths, since the blocking areas are adjacent $\overline{F_{i,j}}$ by Lemma 6. Thus, a minimal path only needs to traverse one blocking area to enter $\overline{F_{i,j}}$.

Lemma 17. For a cyclical blocking $(s_1, t_1), \ldots, (s_k, t_k)$, there exists a swap between two start-target pairs which will remove the cycle without creating additional cycles.

Proof. By definition, a target and their blocking area are separated by a start configuration. Thus, there should exist at least one start configuration inside $\overline{F_{i,j}}$ which separates the targets from their blocking areas. Call this start s_x, and its matched target configuration t_x. Since s_x and the target configurations in our cycle are inside $\overline{F_{i,j}}$ (Lemma 16), there is an unblocked path from s_x to any target configuration in the cycle. The path from s_x to t_x also cannot cross the blocking area of any target configuration in the cycle, since that would mean the blocking area would be crossed in opposite directions, which cannot happen due to Lemma 15. Similarly, t_x cannot be a blocker for any path in the cycle.

We can take some start-target pair (s_i, t_i) from the cycle and swap it with (s_x, t_x) in the matching. Given that both s_x and t_i are in $\overline{F_{i,j}}$, by Lemma 10 we know there exists an unblocked path in the motion graph connecting the two. This removes the current cycle, since t_i can block other paths but its path cannot be blocked. What remains to be shown is that the new path from s_i to t_x does not create any additional cycles.

Given the original cycle, the path from s_i to t_x will be blocked by the start-target pair (s_{i-1}, t_{i-1}) (among potential other blockers). If t_x does not reside in $\overline{F_{i,j}}$ than the fact that t_{i-1} is a blocker does not matter, since we have established that for a blocking cycle to exist all targets should reside in the same residual component. Thus, it will not lead to any additional blockings.

The remaining case is if t_x resides in $\overline{F_{i,j}}$. Assume for contradiction that because of the additional blocking from t_{i-1} there now exists a new cycle. Since t_x is in $\overline{F_{i,j}}$ and it must be part of the new cycle, all targets in the new cycle must reside in $\overline{F_{i,j}}$. The target t_{i-1} must be part of the new cycle. Since it was also part of the previous cycle, there must exist at least one start-target pair in the new cycle which is blocked by more than one blocker in $\overline{F_{i,j}}$. However, this is impossible, since all blocking areas are adjacent to their residual component by Lemma 6 and it is always possible to cross only a single adjacent blocking area to enter $\overline{F_{i,j}}$. □

Theorem 2. For a single connected component $F_i \subset \mathcal{F}$ containing an equal number of start and target configurations, the matching algorithm always finds a solution to the unlabeled motion planning problem for unit-disc robots in a simple workspace, assuming monochromatic separation $\mu = 4$ and bichromatic separation $\beta = 2$.

Proof. By Lemma 17, whenever there exists a blocking cycle a swap can be made between a target configuration in the cycle and a start configuration "inside" the cycle, which removes the cycle and creates no additional cycles. This procedure can thus be repeated until all cycles are removed from the minimal matching. A start-target pair can only be part of one cycle, since all targets of a cycle are in the same residual component and a start-target path can only be blocked by a single blocker in one residual component. Therefore, the algorithm will eventually terminate. The algorithm will thus find a matching which does not have any cyclical blockings, and therefore there exists an ordering of executing the matching which never encounters a blocked edge. In other words, when solving the start-target pairs in this ordering no path will be blocked. □

3.5.3 Complexity analysis

Lemma 18. The matching algorithm finds a solution to the unlabeled motion planning problem in $O((m+n)\log(m+n) + mn + m^3)$.

Proof. Calculating the free space components F_i and the blocking areas B_i is bounded by $O((m+n)\log(m+n))$ by Lemma 5. The motion graph can then be calculated in $O(mn+m^2)$ by Lemma 14.

An initial minimal weight matching can be found using a naive algorithm in $O(m^3)$ by taking an arbitrary matching and swapping matched pairs until there is no longer any improvements to be made. Likely there is a less naive solution to finding this minimal weight matching, perhaps by taking advantage of the tree structure of the blocking areas. However, for our purpose a pessimistic upper bound is sufficient to show the algorithm can find a solution in polynomial time.

For each matched pair we have to calculate a path through the free space and store which blockers, if any, the path passes. The paths can be calculated in $O(m)$, shown in Lemma 13.

The procedure for removing the cyclical blockings from the matching is bounded by $O(m^2)$. A cycle can be found in $O(m)$, for example by starting from a matched pair and following all blockings (similar to a depth-first search). The cycle is then removed by performing a local swap with an adjacent start configuration and two new paths have to be calculated, which is also bounded by $O(m)$, see Lemma 13. Since each start-target pair can initially only be part of a single cycle, there can be at most $O(m)$ cyclical blockings which need to be removed. Therefore, removing cyclical blockings can be done in $O(m^2)$.

The final topological ordering can be found in $O(m)$ once we have a matching without cyclical blocking by performing a simple in-order tree walk of the matched pairs and their blockings. The resulting paths for each robot will have complexity $O(m+n)$, therefore the entire motion plan will have complexity $O(mn)$. Thus, the algorithm can find a solution in $O((m+n)\log(m+n) + mn + m^2)$. □

3.6 Divide-and-conquer algorithm

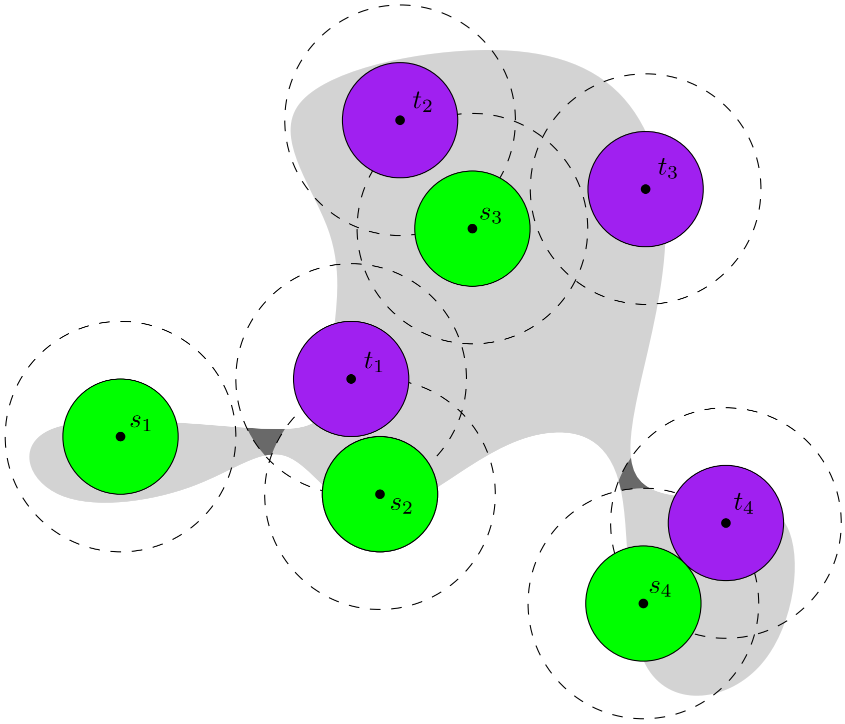
In this section a divide-and-conquer approach is described for solving the motion planning problem for each robot. The general idea is to use the blocking area graph H_i, which is a tree, in order to split the problem into smaller subproblems. The procedure will pick a specific residual component of the graph, occupy its target positions, and then remove the nodes and its edges from the graph. Afterwards, we recurse on the remaining subtrees.

3.6.1 The algorithm

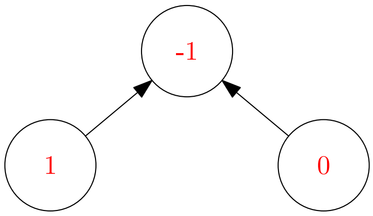
The algorithm will use the blocking area graph $H_i = (V_i^G, E_i^G)$, described in Section 3.2, to recursively solve the motion planning problem. The reason why the blocking area graph is used is that, by Lemma 10, the robots inside a residual component of H_i can always be moved to targets without being blocked. The main idea is then to pick a residual component such that after solving it no robots need to move across its adjacent blocking areas, meaning we do not have to deal with any blockers in solved residual components.

Initially, if there is an edge which splits H_i into zero weight subtrees, meaning the two remaining trees have an equal number of robots and target configurations, then the edge is removed and the algorithm will recurse on the two remaining subproblems.

However, if no such edge is found, then we pick a vertex of H_i which requires robots to move across each edge in H_i to balance its weight. For this purpose, let $D_i = (V_i^D, E_i^D)$ be a directed graph where V_i^D is equal to V_H. Let there be an edge between two residual



(a) An instance with two blocker targets t_1 and t_4, with blocking areas shown in gray.



(b) The corresponding directed graph D_i, with the weight of each residual component in red.

Figure 3.15: An example of the directed graph D_i for an instance with two blockers and three residual components.

components u and v if there exists an undirected edge in H_i between the components. The edge is directed from u to v if, after removing the edge between u and v in H_i, the subtree containing u has a positive weight, and directed from v to u if it has a negative weight. See Figure 3.15 for an illustration of this directed graph.

The algorithm will then pick a sink node of D_i, which always exists (see Lemma 19). Let σ be this sink node. This component is now solved as follows: First, all robots inside σ are moved to unoccupied target configurations. Given that σ is a sink, all edges adjacent to σ are pointing inwards, therefore each edge requires at least one robot to move into σ. For each edge the required number of robots are moved into σ across the blocking area. If the blocker which is associated with blocking area is occupied (thus the blocker has to reside in σ), we move this robot to our original target and the (now unoccupied) blocker becomes the new target.

Once all target configurations of σ are filled, the component and its edges are removed from

H_i and we recurse on the remaining subgraphs. Given the way σ was selected and how the robots were moved into σ, each subgraph should have an equal number of robots and target configurations. Note that certain start configurations might now be unoccupied.

Algorithm 3 Pseudo code for the divide-and-conquer algorithm for a single free space component F_i.

```
procedure MotionPlanningDivideConquer(𝒲, S_i, T_i)
    Calculate free space F_i
    Compute blocking areas and create blocking area graph H_i = (V_i^H, E_i^H)
    Create the motion graph G_i = (V_i^G, E_i^G)
    if there exists edge e ∈ E_i^H which divides H_i into zero weight parts then
        Cut H_i on e and recurse on parts
    else
        Create a directed graph D_i = (V_i^D, E_i^D) from H_i
        Find sink node σ ∈ D_i
        for each start s ∈ σ do
            Find nearest unoccupied target t ∈ σ
            Make pebble move from s to t
        end for
        for each edge e ∈ E_i^D that points to σ do
            for each robot at start s that needs to move in across e do
                find unoccupied target t ∈ σ
                if A blocker t_b ∈ σ blocks movement from s to t then
                    Make pebble move(s) from t_b to t
                    Make pebble move(s) from s to t_b
                else
                    Make pebble move(s) from s to t
                end if
            end for
        end for
        remove σ from H_i and D_i and recurse on subproblems
    end if
end procedure
```

See Algorithm 3 for pseudocode.

3.6.2 Correctness

In this section we argue that the divide-and-conquer algorithm always finds a solution. The algorithm uses the blocking area graph H_i to divide the problem into subproblems. If there is an edge that splits the problem evenly, then these are first removed to split H_i into as many smaller subproblems as possible. Otherwise, the directed graph $D_i = (V_i^D, E_i^D)$ is created by directing all edges in H_i from positive weight subgraphs to negative weight. Here, a sink node σ is then selected and solved.

Lemma 19. There always exists a sink vertex in directed graph D_i

Proof. Since there are no edges that split H_i into zero weight subgraphs, every edge in D_i has a defined direction. The blocking area graph H_i is a tree, by Lemma 8, and D_i is created by directing all edges. Thus, D_i is a directed acyclic graph (DAG) and must contain at least one sink vertex □

Theorem 3. For a connected free space component $F_i \subset \mathcal{F}$ containing an equal number of start and target configurations, the divide-and-conquer algorithm always finds a solution to the unlabeled motion planning problem for unit-disc robots in a simple workspace, assuming monochromatic separation $\mu = 4$ and bichromatic separation $\beta = 2$.

Proof. We argue correctness using induction on the number of vertices of H_i:

If H_i consists of a single residual component $\overline{F_{i,j}}$, then the algorithm will treat $\overline{F_{i,j}}$ vacuously as a sink. All target configurations will then be filled greedily by finding the nearest unoccupied target for each start, and by Lemma 10 there is an unblocked path in the motion graph between any two such configurations. Thus, it always finds a valid solution.

Otherwise, assume that H_i consists of more than one residual component. We assume that the algorithm is correct for any blocking area graph H' with fewer vertices than H_i (induction hypothesis). We now show that the algorithm will reduce H_i into smaller subproblems which can be solved.

If there exists edge $e \in E_H$ which divides H_i into zero weight subtrees the algorithm cuts e from H_i and recurse on both subproblems. Since both subtrees have strictly fewer vertices and an equal number of start and target configurations, both subtrees can be solved according to the induction hypothesis.

If no such edge exists, the algorithm finds a "sink" residual component σ where all adjacent edges in H_i require robots to move into σ. By Lemma 19, there always exists such a sink component. Since σ has only edges pointed inwards, it means that all edges require one or more robots to move into σ in order to fill its target configurations. This means the free space associated with σ will have at least one more target configuration compared to start configurations for every adjacent edge.

Before moving robots into σ, the robots that already reside in σ will be moved to target configurations. By Lemma 10, there exists a path in the motion graph that cannot be blocked and thus this is always possible. Afterwards, the required number of robots are moved in across every edge adjacent to σ in H_i. These paths can cross blocking areas, however, only targets inside σ can be occupied. Thus, if the robots have to cross the blocking area of an occupied blocker b_t inside σ, the algorithm will first move the robot at b_t to the original target configuration (which is always possible by Lemma 10) and then move the robot outside σ to t_b instead.

Once σ is completely solved, the algorithm will remove σ and all its adjacent edges from H_i. All remaining connected components are strictly smaller and have an equal number of start and target configurations, thus we have assumed the algorithm is able to solve them. □

3.6.3 Complexity Analysis

Lemma 20. The matching algorithm finds a solution to the unlabeled motion planning problem in $O((m+n)\log(m+n)+mn+m^2)$.

Proof. Similar to before, calculating the free space component F_i and the blocking areas B_i is bounded by $O((m+n)\log(m+n))$ by Lemma 5. The motion graph can then be calculated in $O(mn+m^2)$ by Lemma 14. The blocking area graph follows from the calculated components, and can similarly be generated within $O((m+n)\log(m+n))$.

In total, the divide-and-conquer algorithm will calculate a path for each target configuration, which is bounded by $O(m)$ by Lemma 13, thus calculating all paths can be done in $O(m^2)$. A path sometimes requires the additional movement of a robot at a blocker position, but this does not influence the $O(m)$ bound. The resulting paths for each robot will have complexity $O(m+n)$, therefore the entire motion plan generated will have complexity $O(mn)$.

The number of iterations for the recursive algorithm is bounded by $O(m)$, since it always either occupies a target configuration or removes an edge from H_i. Since H_i is a tree by Lemma 8, the number of edges is bounded by $O(m)$. Detecting an edge in H_i to remove can be done efficiently in $O(m)$ by rooting the graph H_i and for each vertex storing the weight of the subtree rooted at that vertex. An edge that splits H_i into zero weight parts will then correspond to an edge where the subtree rooted at the "child" vertex has a weight of zero. Similarly, storing the weight for each subtree of H_i allows us to find the sink vertex σ efficiently. After σ is solved and removed, updating the weight can be done in $O(m)$.

Thus, the algorithm can find a solution in $O((m+n)\log(m+n)+mn+m^2)$. ☐

3.7 No bichromatic separation

So far, we have given two algorithms that can solve the unlabeled robot motion planning problem for unit-disc robots in a simple workspace, for a single connected component of the free space and assuming monochromatic separation of $\mu = 4$ and bichromatic separation of $\beta = 2$.

However, we have only shown in Chapter 2 that $\mu = 4$ is sometimes necessary for the problem to have a solution. Thus, we would like to further extend the algorithms such that they no longer require any bichromatic separation, as we have theorized that bichromatic separation is not necessary given a the free space consists of a single connected component. Recall that in Chapter 2 the lower bound $\beta = 3$ was only shown necessary for an instance with multiple free space components.

Fortunately, the extension to $\beta = 0$ is relatively straight-forward. The monochromatic separation restricts the configurations such that configurations closer than a distance of two to each other only come in pairs of one start and one target configuration.

Lemma 21. For any configuration $u \in S_i \cup T_i$, there is at most one other configuration $v \in S \cup T$ that resides inside its aura $\mathcal{D}_2(u)$, where $u \neq v$.

Proof. Assume for contradiction that there exists some configuration u which has more than

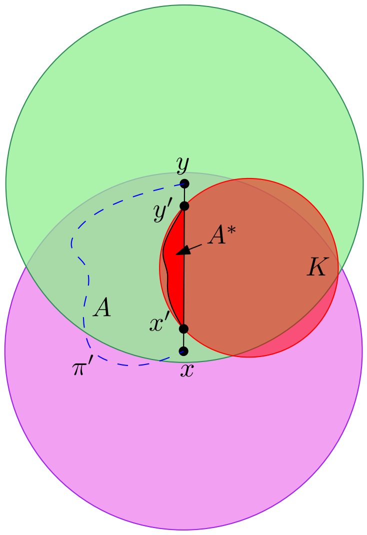


Figure 3.16: Illustration for Lemma 22. The light green and pink discs represent the auras of y and x respectively, while A^* is contained within the red unit-disc K.

one start/target configuration inside its aura. Assume w.l.o.g. that u is a start configuration. Since $\mu = 4$, we know that the configurations inside the aura of u must then be target configurations. Let v, w be two of such target configurations inside $\mathcal{D}_2(u)$. Using the triangle inequality, we find that $d(v, w) \leq d(v, u) + d(u, w) < 2 + 2 < 4$, given that the aura is defined as an open set. Since v and w are both target configurations, this contradicts with the monochromatic separation $\mu = 4$. □

In addition to the fact that configurations closer than a distance two together only come in pairs, such pairs also have an unobstructed path between them.

Lemma 22. For any two configurations $x, y \in S_i \cup T_i$ such that $y \in \mathcal{D}_2(x)$ and $x \neq y$, there exists a path $\pi \subset F_i \cap \mathcal{D}_2(x) \cap \mathcal{D}_2(y)$ connecting x and y.

Proof. The proof is similar to Lemma 2 in the paper by Adler et al. [1]. Take $x, y \in S_i \cup T_i$ such that $y \in \mathcal{D}_2(x)$ and $x \neq y$. By this lemma, there exists a path $\pi' \subset F_i \cap \mathcal{D}_2(x)$ connecting x and y. Assume the line segment $\overline{xy}$ does not lie in the free space, since otherwise the path $\pi = \overline{xy}$ stays within $F_i \cap \mathcal{D}_2(x) \cap \mathcal{D}_2(y)$. Take configurations x', y' on $\overline{xy}$ such that $x', y' \in F_i$ and the distance $||x - y||$ is minimized. Let A be the area enclosed by $\pi \cup \overline{xy}$ and $A^* = A \setminus F_i$ be the part of A which lies in the obstacle space. We claim that $A^* \subset \mathcal{D}_2(x) \cup \mathcal{D}_2(y)$.

Assume w.l.o.g. that y lies directly above x and that π to the left of $\overline{xy}$. Let K be the unit circle which intersects both x and y. Note that K must lie to the right of $\overline{xy}$. The region A^* must then be entirely enclosed within $K \cap A$, which must be within $\mathcal{D}_2(x) \cup \mathcal{D}_2(y)$. Thus, a path consisting of $\overline{xx'}$, $\delta(A^*)$, and $\overline{y'y}$ connects x and y through F_i and stays within $\mathcal{D}_2(x) \cap \mathcal{D}_2(y)$.

See Figure 3.16 for an illustration of this proof. □

The bichromatic separation was previously only used for the creation of the motion graph, where it is used to show that target configuration all reside in residual components of the free space after taking the complement of the start configurations. The algorithms only make use of the motion graph and the bichromatic separation is not needed for those. Therefore, we can make use of the fact that start-target configurations within distance two only come in pairs to adjust the motion graph and the algorithms.

Theorem 4. For a single connected free space component $F_i \in \mathcal{F}$ containing an equal number of start and target configurationss, there always exists a solution to the unlabeled motion planning problem for unit-disc robots in a simple workspace, assuming monochromatic separation $\mu = 4$.

Proof. By Theorem 2 and Theorem 3, the matching and divide-and-conquer algorithms both find a solution assuming also bichromatic separation $\beta = 2$. We will now show how to adjust the algorithms and the construction of the motion graph once this assumption no longer holds.

Let $Q \subseteq S$ be the set of start configurations for which there exists a target configuration in their aura. By Lemma 21, we know that each $s \in Q$ has a unique target configuration $t \in T$, which we will denote with t_s. Let $T(Q) = \bigcup_{s \in Q} t_s$ be the set of target configurations residing in the aura of some start in Q. For the construction of the motion graph we ignore all target configurations in $T(Q)$. The start configurations in Q will be handled the same as regular start configurations. This will leave all start configurations and all targets with bichromatic separation of $\beta = 2$, meaning the motion graph can be generated as before.

For the algorithms the only adjustment is that the start configurations in Q need to be treated as both a start and target. For the matching algorithm (see subsection 3.5.1), this simply means the initial matching M should match each node associated to start configuration $s \in Q$ to itself. Removing cyclical blockings remains unchanged, since the path between a node matched to itself can never be blocked thus also not be part of a cyclical blocking (note that a swap which such a node can still be made).

For the divide-and-conquer algorithm, the start configurations only play a role when solving a residual component of the blocking area graph. However, we can again simply treat the start configuration as a target when solving a component. The start configuration can never act as a blocker by definition, and thus no additional problems arise.

After the initial algorithms are finished, the robots will be at target configurations $T_i \setminus T(Q)$ and at all start configurations in Q. What remains is then to move the robots from the start configurations in Q to their matched targets in $T(Q)$ such that at the end each target in T is occupied. By Lemma 22 for each start configuration $s \in Q$ there exists a path π from s to t_s which stays within $F_i \cap \mathcal{D}_2(s) \cap \mathcal{D}_2(t)$. Given the monochromatic separation $\mu = 4$, π does not cross the aura of another configuration in $S_i \cup T_i$ besides s and t_s. Therefore, we can move the robot at s to t_s across π without interference from another configuration. Doing this for all starts in Q will result in all target configurations in T_i being occupied. □

4 Multiple free space components

In this chapter we will consider the case where the free space $\mathcal{F}$ consists of multiple connected components. In Lemma 2 of Chapter 2, it was already shown that when $\beta < 3$ a solution does not always exist if the free space contains more than one connected component. Thus, we will assume the separation bounds of $\mu = 4$ and $\beta = 3$. The algorithm(s) discussed in Chapter 3 can be applied to find motion plans for single free space components $F_i \in \mathcal{F}$. Thus, the remaining difficulty lies in the interaction between free space components. Even though the free space is disconnected, robots in one free space component might still block a valid path in another component. An example can be seen in Figure 4.1. In this chapter, we will describe how to order the motion plans for each single connected component such that we can solve the unlabeled multi-robot motion planning problem for multiple free space components.

In the paper by Adler et al. [1], a procedure is shown for defining an order of solving the free space components. This procedure can be described as follows:

Let F_i, F_j be two distinct components of $\mathcal{F}$, and let $x \in F_i$ be such that $\mathcal{D}_2(x) \setminus F_j \neq \emptyset$. We then call x an *interference configuration* from F_i to F_j, and define the *interference set* from F_i to F_j as $I_{(i,j)} \triangleq \{x \in F_i : D_2(x) \setminus F_j \neq \emptyset\}$. We also define the *mutual interference set* of F_i, F_j as $I_{\{i,j\}} \triangleq I_{(i,j)} \cup I_{(j,i)}$. Intuitively, an element of the interference set from F_i to F_j is a point in F_i which, when a robot occupies it, could block a path in F_j, and the interference set is the set of all such points. The mutual interference set of F_i, F_j is the set of all single-robot configurations in either component which might block a valid single-robot path in the other component.

To obtain the ordering of free space components, a directed graph representing the structure of $\mathcal{F}$ is defined, called the directed-interference forest $G = (V, E)$, where the nodes in V correspond to the components F_i. We add the directed edge (F_i, F_j) to E if either there exists a start configuration $s \in S$ such that $s \in I_{(i,j)}$, or there exists a target configuration $t \in T$ such that $t \in I_{(j,i)}$. Intuitively, a directed edge (F_i, F_j) shows that if F_i is not solved before F_j, interference will occur.

In lemma 3 of the paper by Adler et al. [1], it is shown that for any mutual interference set $I_{\{i,j\}}$ and any two configurations $x_1, x_2 \in I_{\{i,j\}}$ we have $\mathcal{D}_2(x_1) \cap \mathcal{D}_2(x_2) \neq \emptyset$. Together with the separation constraints $\mu = \beta = 4$ that is assumed in the paper, there cannot be more than one start of target configuration in $I_{\{i,j\}}$. This avoids loops of size 2. Since W is simple, loops of size 3 or larger are also impossible. Thus, G is a DAG and a topological ordering can be found that respects interference between components.

However, with a tighter separation of $\beta = 3$, the claim that the mutual interference set $I_{\{i,j\}}$ can at most contain a single start or target configuration is no longer valid. Since μ remains 4, it is still true that the mutual interference set cannot hold two or more start configurations

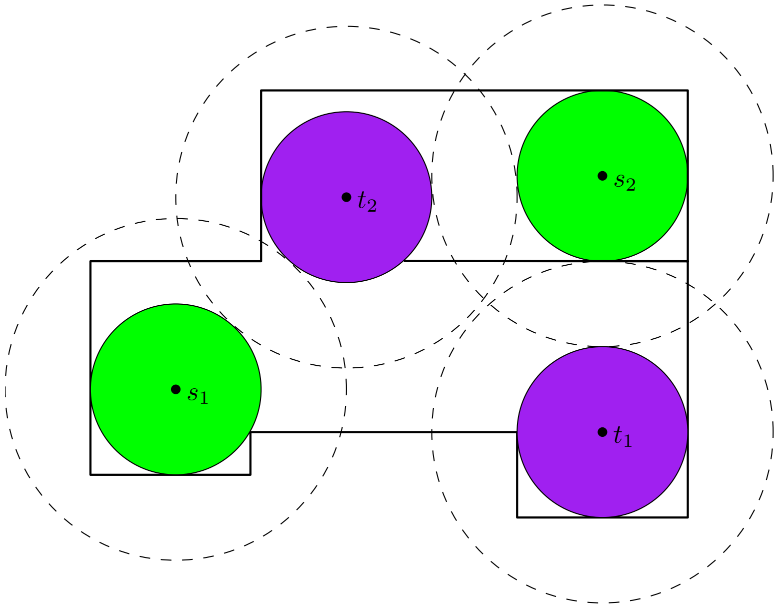


Figure 4.1: An example of a configuration (t_2) blocking movement (s_1 to t_1) in another free space component.

or two or more target configurations. However, it does allow the mutual interference set to contain both a start **and** a target configuration. See Figure 4.2 for an example of this interference. If both start and target configuration in the mutual interference set belong to the same free space component, the directed graph G contains a loop of size 2. This breaks the topological ordering.

Thus, for $\beta = 3$ the ordering breaks for the case where a start and a target configuration in one free space component both interfere with the another component. In this case no direct ordering can be made since the component interferes when its start configurations are occupied as well as when its target configurations are occupied.

However, interference does not always affect the connectivity of the affected free space component. Therefore, we define a *remote blocker* as a target *or* start configuration that intersects the boundary of a free space component, other than the one it resides in, in more than one connected component. By definition, all remote blockers between two free space components are in the mutual interference set.

Lemma 23. If the unlabeled motion planning problem has no remote blockers, then there is always a solution.

Proof. We can create a motion plan for each free space component separately, using (one of) the algorithm(s) in Chapter 3.

Let x be an element of the interference set $I_{(i,j)}$ between free space components F_i and F_j. Given the separation bounds $\mu = 4$ and $\beta = 3$, the aura of a x cannot contain another start/target configuration, thus any robot path π that intersect the aura of x does so in at least two points. In other words, any path through the aura of x must also leave the aura. Since x is not a remote blocker, its aura intersects the boundary of F_j in one connected component.

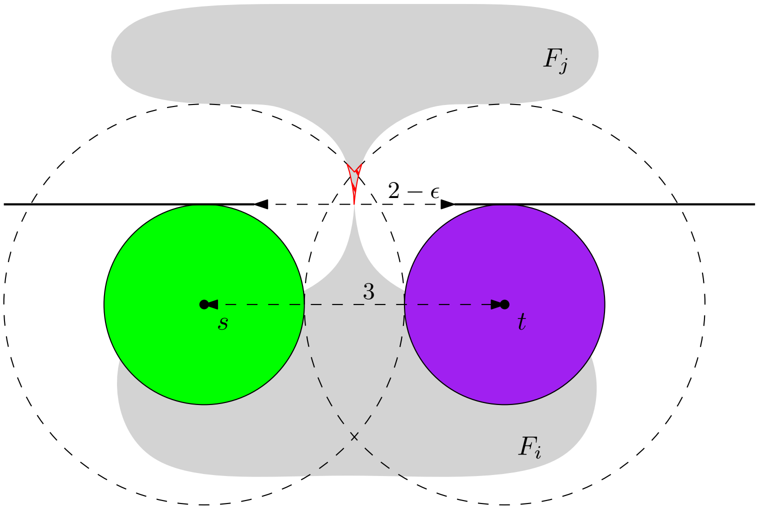


Figure 4.2: Example of start and target configuration both interfering with another free space component.

Therefore, every path π that crosses the aura of x can be modified to use $\delta(\mathcal{D}_2(x))$ instead. After modifying the paths for each element in the interference sets we will arrive at a valid solution. □

Identifying the remote blockers in the mutual interference set shows us which configurations can break the connectivity and therefore potentially the solvability of the destination free space component. Looking at remote blockers, we can again try to find an ordering to the components that will satisfy the direction of the blockings.

Lemma 24. The interference set $I_{(i,j)}$ of free space components F_i, F_j cannot contain a start configuration x_1 and target configuration x_2 that are both remote blockers of F_j.

Proof. Let F_i be a free space component containing a start configuration x_1 and a target configuration x_2 and let F_j be another free space component such that $x_1, x_2 \in I_{(i,j)}$. For a contradiction, we assume that both configurations are remote blockers for F_j. From Lemma 3 from the paper by Adler et al. [1], we know that $\mathcal{D}_2(x_1) \cap \mathcal{D}_2(x_2) \neq \emptyset$. Together with $\beta = 3$, we conclude that $3 \leq d(x_1, x_2) < 4$, where $d()$ is the euclidean distance function.

We take $y_1 \in \delta(\mathcal{D}_2(x_1)) \cap \delta(F_j)$ and $y_2 \in \delta(\mathcal{D}_2(x_1)) \cap \delta(F_j)$ such that the $d(y_1, y_2)$ is maximized. We assume w.l.o.g. that x_1 lies to the left of x_2 and that the line segments $\overline{x_1y_1}$ and $\overline{x_2y_2}$ do not intersect (in such a case we could choose x_1 and x_2 differently such that their distance does not decrease and the line segments do not intersect). See Figure 4.3 for an illustration.

Since $y_1 \in \mathcal{F} \cap \mathcal{D}_2(x_1)$, we know that $\overline{x_1y_1} \in \mathcal{W}$, similarly for $\overline{x_2y_2}$. Given that $x_1, x_2 \in F_i$, there exists a path $\pi_1 \subset F_i$ between x_1 and x_2. Similarly, there exists a path $\pi_2 \subset F_j$ between y_1 and y_2. We now define the curve $\lambda \triangleq \pi_1 \cup \overline{x_1y_1} \cup \pi_2 \cup \overline{x_2y_2}$ for which we can conclude that $\lambda \subset \mathcal{W}$. Let A be the area enclosed by λ. Since $\mathcal{W}$ is simple and $\lambda \in \mathcal{W}$, we have $A \subset \mathcal{W}$.

We now define points x_1', y_1' to be points on $\overline{x_1y_1}$ such that $x_1' \in F_i$ and $y_1' \in F_j$ and the distance $d(x_1', y_1')$ is minimized. We do similar for x_2' and y_2' for the line segment $\overline{x_2y_2}$. Take a unit-disc K_1 such that K_1 lies to the left of $\overline{x_1y_1}$ and passes through x_1' and y_1'. Similarly, take

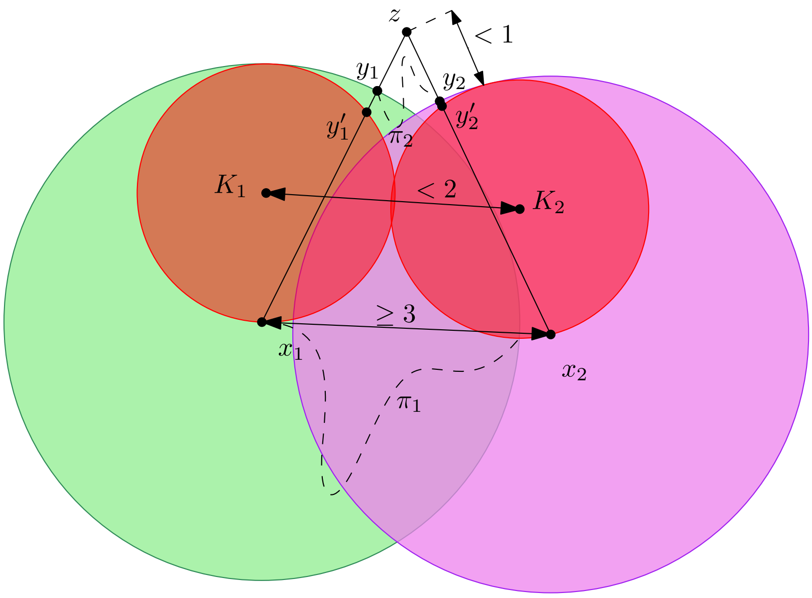


Figure 4.3: Illustration of Lemma 24. The collision disk $\mathcal{D}_2(x_1)$ is shown in green, $\mathcal{D}_2(x_2)$ in purple. The disks K_1, K_2 are shown in red. For simplicity we have shown when $x_1 = x'_1$ and $x_2 = x'_2$.

a unit-disc K_2 for x'_2 and y'_2 that lies to the right of $\overline{x_2 y_2}$. We define $A^* \triangleq A \setminus \mathcal{F}$. Then we have that $A^* \subset K_1 \cup K_2$. From this we can conclude that $K_1 \cap K_2 \neq \emptyset$, otherwise F_i and F_j are connected. Thus, we know that $d(K_1, K_2) < 2$.

Since $d(x_1, x_2) \geq 3$ and $d(K_1, K_2) < 2$, we can conclude that $d(y_1, y_2) < 1$. Let z be the intersection between the line through x_1 and y_1 and the line through x_2 and y_2. Then, $d(y_1, z) < 1$, and similarly $d(y_2, z) < 1$.

By how we defined y_1, we know there exists $w_1 \in \mathcal{O} \cap \mathcal{D}_1(y_1)$ and that w_1 lies on or to the right of the line $x_1 y_1$ (otherwise there exists a point $y''_1 \in \delta(\mathcal{D}_2(x_1) \cap \delta(F_j)$ to the left of y_1 which would mean $d(y_1, y_2) < d(y''_1, y_2)$). Similarly, for y_2 there exists $w_2 \in \mathcal{O} \cap \mathcal{D}_1(y_2)$ which lies to the left of line segment $\overline{x_2 y_2}$. However, this contradicts with the fact that intersection z lies within distance 1 of both y_1 and y_2. □

Since the interference set cannot contain more than one remote blocker by Lemma 24, we can find a topological ordering of solving the free space components such that the motion plan for a free space component F_i is never blocked by a robot in a component $F_j \neq F_i$. Intuitively, since we cannot have both a start and target configuration as a remote blocker from one free space component to the other, there is a well-defined direction regarding remote blockers between two components.

Note that a configuration that is not a remote blocker can still interfere, however any path in the motion plan that crosses the aura of an interfering configuration can be modified to use the boundary of the aura instead, see Lemma 23.

Theorem 5. Given m unit-disc robots in a simple polygonal workspace $W \in \mathbb{R}^2$, with start and target configurations S, T and separation constraints $\mu = 4$ and $\beta = 3$. Assuming each maximal connected component F_i of the free space $\mathcal{F}$ for a single unit-disc robot in W contains an equal number of start and target configurations, there exists a collision free motion plan for the robots starting at S such that all target configurations in T will be occupied after execution.

Proof. We can create a motion plan for each connected components $F_i \subset \mathcal{F}$ of the free space using one of the algorithm discussed in Chapter 3 (Theorem 2, Theorem 3). We can then use the fact that only a single configuration can be a remote blocker between two components of $\mathcal{F}$ by Lemma 24 to find an ordering for solving the free space components that respects remote blockers.

To obtain the ordering of free space components, a directed forest $G = (V, E)$ is created such that the nodes in V correspond to the free space components $F_i \in \mathcal{F}$. We add the directed edge (F_i, F_j) to E if either there exists a start configuration $s \in F_i$ such that s is a remote blocker for F_j, or there exists a remote blocker target configuration $t \in F_j$ such that t is a remote blocker for F_i.

Given the fact that an interference set cannot contain two remote blocker by Lemma 24 and the workspace W is simple, the graph G is a DAG. Therefore, G has a topological ordering that respects remote blocking between components.

Lastly, the motion plan of one free space component might still encounter interference from other free space components. But since these are not remote components, we are able to modify all paths that pass the aura of an interfering configuration to take the boundary instead, as explained in Lemma 23. □

5 Conclusion

In this thesis we have shown that the unlabeled multi-robot motion planning problem for unit-disc robots in a simple workspace can always be solved using monochromatic separation of four and bichromatic separation of three, as long as each connected component of the free space contains an equal number of start and target configurations. When the free space consists of a single component, the bichromatic separation constraint can even be dropped and only monochromatic separation of four is necessary for the problem to have a solution. These separation bounds are tight, which improves upon what was previously shown by Adler et al.[1]. In Chapter 2 it was shown that monochromatic separation of four is sometimes necessary for the problem to always have a solution, and similarly that bichromatic separation of three is sometimes necessary when the free space consists of multiple components.

The lack of bichromatic separation bounds when $\beta < 4$ makes the problem significantly more difficult, since the auras of start and target configurations can overlap. This results in situations where start/target configurations can block part of the free space together, see blocking areas in Section 3.1. We have looked at several ways of handling blockers to still always be able to find a solution to the motion planning problem.

We explored different algorithms to solve the multi-robot motion planning problem for a single free space component in Chapter 3, first assuming the monochromatic separation of four and a bichromatic separation of two. In Section 3.7 it was then shown how to adapt the algorithm to handle the case without any bichromatic separation.

In Chapter 3 we discussed three separate approaches for solving the problem. A greedy algorithm which tries to handle blockers “on-the-fly”, a matching algorithm which calculates a matching without cyclical blockings, and a divide-and-conquer algorithm that recursively solves the problem using the blocking area graph. In the end, the matching and the divide-and-conquer algorithm proved to always be able to solve the problem in polynomial time.

Though the greedy approach showed some promise, there were potential issues with cyclical blockings and the running time. For the matching algorithm it was proven that cyclical blockings could always be removed, and thus a topological ordering for solving start-target pairs can be found that does not result in any blockings. For the divide-and-conquer algorithm, the blocking area graph could be used to split the instance into subproblems and solve the subproblems recursively. Thus, we have presented two algorithms that can always find a solution to the unlabeled robot motion planning problem for unit-disc robots in a simple workspace. Moreover, they can find such a solution in polynomial time.

Comparing the algorithms, we proved a better theoretical upper bound for the complexity of the divide-and-conquer algorithm, though the upper bound for the matching algorithm is likely overly pessimistic. The total complexity of both algorithms is also for a large part determined by the construction of the free space, its derivative components, and motion graph plus the complexity of the final paths generated, which they have in common. Nonetheless, we suspect the divide-and-conquer algorithm will perform better on average than the matching algorithm, simply by its recursive nature. The divide-and-conquer has the potential to split the problem into many smaller subproblems, which will be more efficient to solve. For a better comparison, both algorithms would have to be implemented to provide some experimental evidence.

In Chapter 4 we described how to extend the algorithms for a single free space component to handle multiple free space components. The bichromatic separation of three was assumed since it was shown that this is sometimes necessary for the problem to be solvable. With these separation bounds, we showed that there can never be a cyclical blocking between separate free space components and thus a topological ordering of solving these components always exists.

Overall, the work done in this thesis is a small but important step in the field of multi-robot motion planning. It gives us more knowledge under what assumptions the problem is efficiently solvable and what properties make the problem hard. Given the many applications of robot motion planning, for example in robotics, a good theoretical understanding is crucial for the development of efficient algorithms.

5.1 Future work

In this thesis we have focused on the separation constraints for a particular variant of the unlabeled motion planning problem, namely with disc-shaped robots in a simple workspace. Assuming the free space components contain an equal number of start and target configurations, we have given algorithms that can always solve this problem for stricter separation bounds than was previously shown [1],

Additionally, the separation bounds are strict in the sense that a solution does not always exist for less separation. This result is in contrast with the general unlabeled motion planning problem, which was shown to be PSPACE-hard [2] for disc-shaped robots, though this proof uses two classes of robots with different radii. Thus, the question still remains what the exact complexity is for this variant of unlabeled robot motion planning with just a single class of disc-shaped robots, in case we do not assume any separation.

It would be interesting to see if it is possible to create efficient algorithms that can solve the problem with less separation. We have shown that the separation is necessary for the problem to always have a solution, but an algorithm could still be used to find a solution, in case it exists. If the problem is shown to be NP-hard or PSPACE-hard, perhaps an algorithm can be designed whose running time depends in some way on the separation that is assumed.

Another direction would be to apply the separation assumptions to other variant of the motion planning problem. Perhaps we can use a certain separation constraint to design a polynomial algorithm for other variants, for example when the robots are square or polygonal. What challenges arise when the workspace is no longer simple and contains obstacles? Intuitively, obstacles seem to pose an issue when defining an ordering for solving multiple free space components, since configurations can interfere between components at multiple locations.

From a practical side, an implementation of the algorithm(s) proposed could provide additional insight as well as a good comparison based on their performance. We suspect that the recursive algorithm would perform better than the matching algorithm in many cases, since it has the possibility to split the problem up efficiently into small subproblems. Additional difficulties or edge cases could present themselves that were not considered in this thesis.

Bibliography

[1] Aviv Adler, Mark De Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. In *Algorithmic foundations of robotics XI*, pages 1–17. Springer, 2015.

[2] Thomas Brocken, G Wessel van der Heijden, Irina Kostitsyna, Lloyd E Lo-Wong, and Remco JA Surtel. Multi-robot motion planning of k-colored discs is PSPACE-hard. *arXiv preprint arXiv:2004.12144*, 2020.

[3] Andrew Dobson, Kiril Solovey, Rahul Shome, Dan Halperin, and Kostas E Bekris. Scalable asymptotically-optimal multi-robot motion planning. In *2017 international symposium on multi-robot and multi-agent systems (MRS)*, pages 120–127. IEEE, 2017.

[4] Mokhtar Gharbi, Juan Cortés, and Thierry Siméon. Roadmap composition for multi-arm systems path planning. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2471–2476. IEEE, 2009.

[5] Fabien Gravot and Rachid Alami. A method for handling multiple roadmaps and its use for complex manipulation planning. In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, volume 3, pages 2914–2919. IEEE, 2003.

[6] Robert A Hearn and Erik D Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343:72–96, 2005.

[7] John E Hopcroft, Jacob Theodore Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the “warehouseman’s problem”. *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[8] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] Klara Kedem, Ron Livne, János Pach, and Micha Sharir. On the union of jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete & Computational Geometry*, 1(1):59–71, 1986.

[10] Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multi-robot formations. *IEEE Transactions on Robotics*, 22(4):650–665, 2006.

[11] Daniel Martin Kornhauser, Gary Miller, and Paul Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. Master’s thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[12] Gildardo Sanchez and J-C Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 2112–2119. IEEE, 2002.

[13] Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–351, 1983.

[14] Jacob T Schwartz and Micha Sharir. On the piano movers' problem: Iii. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983.

[15] Micha Sharir and Shmuel Sifrony. Coordinated motion planning for two independent robots. *Annals of Mathematics and Artificial Intelligence*, 3(1):107–130, 1991.

[16] Kiril Solovey and Dan Halperin. On the hardness of unlabeled multi-robot motion planning. *The International Journal of Robotics Research*, 35(14):1750–1759, 2016.

[17] Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion planning for unlabeled discs with optimality guarantees. Robotics: Science and Systems Foundation, 2015.

[18] Paul Spirakis and Chee K Yap. Strong np-hardness of moving many discs. *Information Processing Letters*, 19(1):55–59, 1984.

[19] Matthew Turpin, Nathan Michael, and Vijay Kumar. Concurrent assignment and planning of trajectories for large teams of interchangeable robots. In *2013 IEEE International Conference on Robotics and Automation*, pages 842–848. IEEE, 2013.

[20] C Yap. Coordinating the motion of several discs. *Courant Institute of Mathematical Sciences*, 1984.