SIAM J. Comput.
Vol. 48, No. 6, pp. 1727–1762

COORDINATED MOTION PLANNING: RECONFIGURING A SWARM OF LABELED ROBOTS WITH BOUNDED STRETCH[*]

ERIK D. DEMAINE[†], SÁNDOR P. FEKETE[‡], PHILLIP KELDENICH[‡], HENK MEIJER[§], AND CHRISTIAN SCHEFFER[‡]

Abstract. We develop constant-factor approximation algorithms for minimizing the execution time of a coordinated parallel motion plan for a relatively dense swarm of homogeneous robots in the absence of obstacles. In our first model, each robot has a specified start and destination on the square grid, and in each round of coordinated parallel motion, every robot can move to any adjacent position that is either empty or simultaneously being vacated by another robot. In this model, our algorithm achieves *constant stretch factor*: if every robot starts at distance at most d from its destination, then the total duration of the overall schedule is $O(d)$, which is optimal up to constant factors. Our result holds for distinguished robots (each robot has a specific destination), identical (unlabeled) robots, and most generally, classes of different robot types (where each destination specifies a required type of robot). We also show that finding the optimal coordinated parallel motion plan is NP-hard, justifying approximation algorithms. In our second model, each robot is a unit-radius disk in the plane, and robots can translate continuously in parallel subject to not intersecting, i.e., having disk centers at L_2-distance at least 2. We prove the same result—constant-factor approximation algorithm to minimizing execution time via constant stretch factor—when the pairwise L_∞-distance between disk centers is at least $2\sqrt{2} = 2.8284\ldots$. On the other hand, for N densely packed disks at distance at most $2 + \delta$ for a sufficiently small $\delta > 0$, we prove that a stretch factor of $\Omega(N^{1/4})$ is sometimes necessary (when densely packed), while a stretch factor of $\mathcal{O}(N^{1/2})$ is always possible.

Key words. robot swarms, coordinated motion planning, parallel motion, makespan, bounded stretch, complexity, approximation

AMS subject classifications. 68Q25, 05C21, 68T40

DOI. 10.1137/18M1194341

1. Introduction.

How do we coordinate the motion of many robots, vehicles, aircraft, or people? If each mobile agent has a destination in mind, how can it find an efficient route that avoids collisions with other agents as they simultaneously move to their destinations? These basic questions arise in many application domains, such as ground swarm robotics [55, 56], aerial swarm robotics [76, 13], air traffic control [15], and vehicular traffic networks [26, 61].

Multi-robot coordination has been studied since the early days of robotics and computational geometry. As far back as the 1980s, the groundbreaking work by Schwartz and Sharir [62] gave algorithms for coordinating the motion of several disk-shaped objects among obstacles. Their algorithms run in time polynomial in the complexity of the obstacles, but exponential in the number of disks. Around the same

[*]Received by the editors June 14, 2018; accepted for publication (in revised form) August 5, 2019; published electronically November 26, 2019. An extended abstract containing major parts of this paper appeared in the Proceedings of the Thirty-Fourth International Symposium on Computational Geometry (SoCG 2018) [17]. There is a video [6] motivating, visualizing, and demonstrating the concepts of this paper, reachable via https://www.youtube.com/watch?v=_2CsL_vaQTo.
https://doi.org/10.1137/18M1194341

Funding: This work was partially supported by the DFG Research Unit *Controlling Concurrent Change*, funding number FOR 1800, project FE407/17-2, Conflict Resolution and Optimization.

[†]MIT Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 (edemaine@mit.edu).

[‡]Department of Computer Science, TU Braunschweig, Mühlenpfordtstr. 23, 38106 Braunschweig, Germany (s.fekete@tu-bs.de, p.keldenich@tu-bs.de, c.scheffer@tu-bs.de).

[§]Science Department, University College Roosevelt Middelburg, 4331 CB Middelburg, The Netherlands (h.meijer@ucr.nl).

time, Hopcroft, Schwartz, and Sharir [39] and Hopcroft and Wilfong [40] showed that it is PSPACE-complete to decide whether multiple robots can reach a given target configuration, illustrating the significant theoretical challenge of coordinating many individual robots.

Most other previous work on coordinated motion planning (see section 1.2) has largely focused on *sequential* schedules, where one robot moves at a time, with objectives such as minimizing the number of moves. In practice, however, robots usually move simultaneously, so we desire a *parallel* motion schedule, with a natural objective of minimizing the time until completion, called *makespan*.

How well can we exploit parallelism in a robot swarm to achieve an efficient schedule? A simple lower bound for the time required for *all* robots to reach their destinations is the time it takes to move just *one* robot to its destination in the absence of other robots, i.e., by the maximum distance between a robot's origin and destination. Moving a dense arrangement of robots to their destinations while avoiding collisions may require substantially more time than this lower bound. We define the *stretch factor* to be the ratio of the time taken by a parallel motion plan divided by the simple lower bound; see Figure 1.

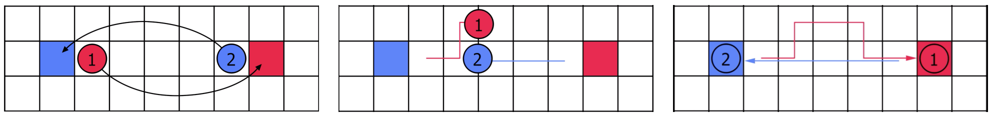


FIG. 1. *An instance from our video* [6] *for the grid case causing stretch larger than* 1.

1.1. Our results. In this paper, we provide several fundamental insights into coordinated motion planning. Our main results show that, under mild assumptions on the separation between robots, it is possible to achieve a *constant* stretch factor, i.e., a stretch factor independent of the number of robots. These algorithms therefore provide an absolute performance guarantee on the makespan of the parallel motion schedule, which implies that the schedule is a constant-factor approximation of the best possible schedule. For densely packed arrangements of robots (without our separation assumptions), we prove that constant stretch factor is no longer possible, and give upper and lower bounds on the worst-case stretch factor.

More precisely, we study two models of coordinated motion planning. In the *grid model*, robots occupy cells of the (2D) square grid and, in each round of motion, each robot can move to any horizontally or vertically neighboring position that is either empty or is simultaneously being vacated by a moving robot. Thus, each round of motion can be defined by a disjoint collection of directed cycles in the dual grid, where each robot simultaneously moves to the next position in its cycle. In the *disk model*, each robot is a unit disk in the plane, and they can all move in parallel up to unit speed so long as the disks remain disjoint. In both cases, each robot has a destination, and we want to move all robots to their destination in the minimum possible time (makespan).

We prove the following results:

1. It is strongly NP-complete to find the optimal makespan in the grid model; see Theorem 3.1.
2. We give a polynomial-time $\mathcal{O}(1)$-approximation algorithm for minimizing the makespan in the grid model. In fact, the algorithm achieves the stronger guarantee of a constant stretch factor; see Theorem 3.5.

3. For our approach, we introduce a technique to separate planar (cyclic) flows into so-called *subflows* whose thickness can be controlled by the number of subflows; see Definition 3.8 and Lemma 3.9. This result is of independent interest for network packet routing with bounded memory. Indeed, our Theorem 3.5 implies that $\mathcal{O}(D)$ steps suffice to route any permutation of dilation D on the grid, even with a buffer size of 1, resolving a 20-year-old open question of Scheideler [60].
4. We extend our approach to establish constant stretch for the generalization of colored robot classes, where each robot has a color, and each destination specifies only the desired color of a robot that must arrive there; see Theorem 4.1. This setting generalizes both labeled robots (as above) and unlabeled/identical robots where we only care about which locations are occupied by robots.
5. We extend our results to establish constant stretch in the disk model, provided the distance between any two robots is at least two disk diameters; see Theorem 5.10. This result implies that efficient multi-robot coordination is always possible, even with nonconvex robots, under relatively mild separability conditions (and no obstacles).
6. For the disk model with N unit disks and no separation condition, we establish a lower bound of $\Omega(N^{1/4})$ and an upper bound of $\mathcal{O}(\sqrt{N})$ on the worst-case achievable stretch; see Theorems 5.8 and 5.11.

We also highlight the geometric difficulty of computing optimal trajectories even in seemingly simple cases; see section 5.4.

The bulk of this paper provides the mathematical validation of these claims. The interested reader may benefit from our video [6] that motivates, visualizes and demonstrates the high-level concepts of this paper (reachable via https://www.youtube.com/watch?v=_2CsL_vaQTo) before diving into the technical proof details.

1.2. Related work. Several variants of multiple-object motion planning problems have received a large amount of attention from researchers in various areas of computer science and engineering; see [22] for a survey, and [29] for a recent book. Their practical relevance is reflected by the fact that there are industrial solutions used in automated warehouses for certain restricted forms of these problems [83]. There are different independent criteria by which these problems can be characterized. A very important distinction is between *discrete* and *continuous* scenarios. In the *discrete* case, the input is a graph in which no two objects may use a vertex or edge at the same time; depending on the scenario, we may be allowed to rotate fully populated cycles. In the *continuous* or *geometric* setting, the objects are shapes in some geometric space which must be moved to a given target position in such a way that their interiors do not intersect at any time. Depending on the scenario, the shapes may or may not touch. Moreover, the objects may be confined to a certain region and there may be stationary obstacles. Under these restrictions, it is unclear whether the target configuration is reachable at all. Aronov et al. [4] demonstrate that, for up to three robots of constant complexity, a path can be constructed efficiently if one exists. Ramanathan and Alagar [52] as well as Schwartz and Sharir [62] consider the case of several disk-shaped objects moving among polygonal obstacles. They both find algorithms deciding whether a given target configuration is reachable. Their algorithms run in time polynomial in the complexity of the obstacles, but exponential in the number of disks. Hopcroft, Schwartz, and Sharir [39] and Hopcroft and Wilfong [40] demonstrate that the reachability of a given target configuration is PSPACE-complete to decide; this already holds when restricted to rectangular objects moving in a rectangular region. Their proof was later generalized by Hearn and Demaine [36, 37], who proved that rectangles of size 1×2 and 2×1 are sufficient and introduced a more general framework to prove PSPACE-hardness of certain block sliding games. Moreover, this problem is similar to the well-known *Rush Hour Problem*, which was shown to be PSPACE-complete by Flake and Baum [28]. For moving disks, Spirakis and Yap [72] have proved strong NP-hardness of the same problem; however, their proof makes use of disks of varying size. Bereg, Dumitrescu, and Pach [7] as well as Abellanas et al. [1] consider minimizing the *number* of moves of a set of disks into a target arrangement without obstacles. They provide simple algorithms and establish upper and lower bounds on the number of moves, where a move consists of sliding one disk along some curve without intersecting other disks. These bounds were later improved on by Dumitrescu and Jiang [23], who also proved that the problem remains NP-hard for congruent disks even when the motion is restricted to sliding.

Kirkpatrick and Liu [44] consider the case of moving two disks of arbitrary radius from a start into a target configuration in an otherwise obstacle-free plane, minimizing the sum of distances traveled by the disks. They provide optimal solutions for two disks moving from an arbitrary initial configuration into an arbitrary goal configuration. Their arguments do not seem to generalize to the makespan. Yu [84] provides an expected constant-factor approximation for the optimal makespan in the grid case.

Díaz-Báñez et al. [21] consider the task of extracting a single object from a group of convex objects, moving a minimal number of objects out of the way. They present an algorithm that finds the optimal direction for extracting the object in polynomial time.

On the practical side, there are several approaches to solving multiple-object motion planning problems, both optimally and heuristically. For discrete instances with a moderate number of objects, optimal solutions can be found using standard search strategies like A^* [35] in the high-dimensional search space of possible configurations. Numerous techniques can be used to improve the efficiency of these strategies [27, 33, 73]. Moreover, there is some work employing SAT solvers [41, 43] to solve multiple-object motion planning problems to optimality. More recently, Yu and LaValle [86] presented an exact algorithm using Integer Programming (IP) for minimizing the makespan that works for hundreds of robots, even for challenging configurations with densities of up to 100%.

For larger instances, one has to resort to heuristic solutions. In *priority planning* [10, 20, 25, 32, 54, 77, 79], the paths are planned one-by-one by assigning priorities to the objects and planning the movement in decreasing order of priority, treating all objects with higher priority as moving obstacles. Kant and Zucker [42] decompose the problem into planning the paths for all objects and avoiding collisions by adapting the velocity of the objects appropriately, an approach on which several papers are based [14, 48, 49, 51, 66]. Another approach is to compute paths for the objects individually and resolve collisions locally [30].

Between these simple *decoupled* heuristics which only consider one object at a time and high-dimensional *coupled* search algorithms lie *dynamically coupled* algorithms [3, 5, 63, 64, 65, 67, 78] which aim for better solutions at the price of higher computational costs. These algorithms typically consider individual objects and only increase the dimension of the search space once a nontrivial interaction between objects is discovered. Recently, Wagner and Choset [81] provided an exact algorithm, i.e., a method that computes a provably optimal solution, with a worst-case runtime that is exponential in the number of robots; in addition, it is bounded suboptimal, i.e., it can check achievability for any given stretch factor. Furthermore, there are also decentralized AI-based multiple-object motion planning approaches [11, 59].

With the advent of robot swarms, practical solutions to these problems became more important and the robotics community started to develop practical sampling-based algorithms [38, 57, 58, 69, 70, 74, 80] which, while working well in practice, are not guaranteed to find an (optimal) solution. In another recent work, Yu and Rus [87] present a practical algorithm based on a fine-grained discretization combined with an IP for the resulting discrete problem to provide near-optimal solutions even for densely populated environments. Other related work includes that of Rubenstein, Cornejo, and Nagpal [55], who demonstrated how to reconfigure a large swarm of simple, disk-shaped *Kilobots*; however, their method is sequential, relocating one robot at a time, so a full reconfiguration of 1000 robots takes about a day, highlighting the relevance of truly parallel motion planning. Further extensions to higher-dimensional problems (with a wide range of additional motion constraints) are swarms of drones (e.g., the work by Kumar; see [76]) and even air traffic control (see Delahaye et al. [15] for a recent survey).

In both discrete and geometric variants of the problem, the objects can be *labeled*, *colored*, or *unlabeled*. In the *labeled* case, the objects are all distinguishable, and each object has its own, uniquely defined target position. This is the most extensively studied scenario among the three. In the *colored* case, the objects are partitioned into k groups, and each target position can only be covered by an object with the right color. This case was recently considered by Solovey and Halperin [67], who present and evaluate a practical sampling-based algorithm. In the *unlabeled* case, the objects are indistinguishable, and each target position can be covered by any object. This scenario was first considered by Kloder and Hutchinson [45], who present a practical sampling-based algorithm. In this situation, Turpin, Michael, and Kumar [75] prove that it is possible to find a solution in polynomial time, if one exists. This solution is optimal with respect to the longest distance traveled by any one robot. However, their results only hold for disk-shaped robots under additional restrictive assumptions on the free space. For unit disks and simple polygons, Adler et al. [2] provide a polynomial-time algorithm under the additional assumption that the start and target positions have some minimal distance from each other. Under similar separability assumptions, Solovey et al. [71] provide a polynomial-time algorithm that produces a set of paths with total length at most $\mathrm{OPT} + 4m$, where m is the number of robots. However, they do not consider the makespan, but only the total path length. On the negative side, Solovey and Halperin [68] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard, even when restricted to unit square objects in a polygonal environment.

Regarding discrete multiple-object motion planning, Călinescu, Dumitrescu, and Pach [9] consider the nonparallel motion planning problem on graphs, where each object can be moved along an unoccupied path in one move. They prove that both in the unlabeled and in the labeled case, minimizing the number of moves required is APX-hard. They provide 3-approximation algorithms for the unlabeled case on general graphs. Moreover, they prove that the problem remains NP-complete on the infinite rectangular grid. Their results are different from our results because the objective they consider is not closely related to the makespan. For other work, see [8, 18, 19, 34] for particular examples.

On grid graphs, the problem can be cast as a very restrictive variant of mesh-connected routing, where each processor can only hold one packet at any time. However, approaches developed for this problem (see Kunde [47] and Cheung and Lau [12]) typically assume that at least a constant number of packets can be held at any processor. On the other hand, on grid graphs, the problem resembles the generalization of the 15-puzzle, for which Wilson [82] and Kornhauser, Miller, and Spirakis [46] have given an efficient algorithm that decides reachability of a target configuration and have provided both lower and upper bounds on the number of moves required. However, Ratner and Warmuth [53] proved that finding a shortest solution for this puzzle remains NP-hard. Demaine, Demaine, and Verrill [16] also consider various grids. For the triangular grid, they give efficiently verifiable conditions for checking whether a solution exists.

During the review period of the abstract version [17] of our work, Yu [85] independently proposed a similar approach that also achieves a constant-factor approximation for the case of a rectangular grid.

2. Preliminaries.

In the following, we provide formal definitions for the grid case (treated in sections 3 and 4) and for the continuous case (the focus of section 5).

2.1. The grid case.

In the grid setting of section 3 we consider a four-connected $n_1 \times n_2$ grid $G = (V, E)$, where $V = \{1, \dots, n_1\} \times \{1, \dots, n_2\}$. A *configuration* of P is a mapping $\mathcal{C} : V \to \{1, \dots, N, \perp\}$, which is injective on the labels $\{1, \dots, N\}$ of the $N \leq |P|$ robots to be moved, whereas $\perp$ denotes an empty square and is thus used $|P| - N$ times. The inverse image of a robot's label ℓ is $\mathcal{C}^{-1}(\ell) = (x_\ell, y_\ell)$. We call x_ℓ and y_ℓ the x- and y-coordinates of the robot. In the following, we consider a *start configuration* $\mathcal{C}_s$ and a *target configuration* $\mathcal{C}_t$; for $i \in \{1, \dots, N\}$, we call $\mathcal{C}_s^{-1}(i) = (x_s, y_s)$ and $\mathcal{C}_t^{-1}(i) = (x_t, y_t)$ the *start* and *target positions* of the robot i. The L_1 (or *Manhattan*) distance between a robot's start and target positions is $|x_s - x_t| + |y_s - y_t|$; this is a lower bound on the number of steps to get a robot to its destination. The L_∞-distance is $\max\{|x_s - x_t|, |y_s - y_t|\}$, and we denote by d the maximum such distance over all robots. Because of $|x_s - x_t| + |y_s - y_t| \leq 2 \max\{|x_s - x_t|, |y_s - y_t|\}$, it suffices to focus on d when establishing upper bounds on the length of a reconfiguration schedule.

A configuration $\mathcal{C}_1 : V \to \{1, \dots, N, \perp\}$ can be *transformed within one single transformation step* into another configuration $\mathcal{C}_2 : V \to \{1, \dots, N, \perp\}$, denoted $\mathcal{C}_1 \to \mathcal{C}_2$, if $\mathcal{C}_1^{-1}(\ell) = \mathcal{C}_2^{-1}(\ell)$ or $(\mathcal{C}_1^{-1}(\ell), \mathcal{C}_2^{-1}(\ell)) \in E$ holds for all $\ell \in \{1, \dots, N\}$, i.e., if each robot does not move or moves to one of the at most four adjacent squares. Furthermore, two robots cannot exchange their squares in one transformation step, i.e., for all occupied squares $v \neq w \in V$, we require that $\mathcal{C}_2(v) = \mathcal{C}_1(w)$ implies $\mathcal{C}_2(w) \neq \mathcal{C}_1(v)$. A configuration $\mathcal{C}_s$ *can be transformed* into a configuration $\mathcal{C}_t$ if there is a sequence $\mathcal{C}_s \to \mathcal{C}_1 \to \cdots \to \mathcal{C}_k \to \mathcal{C}_t$ of transformation steps. For $M \in \mathbb{N}$, a *schedule* is a sequence $\mathcal{C}_1 \to \cdots \to \mathcal{C}_M$ of transformations. The number of steps in a schedule is called its *makespan*. Given a start configuration $\mathcal{C}_s$ and a target configuration $\mathcal{C}_t$, the *optimal makespan* is the minimum number of steps in a schedule starting with $\mathcal{C}_s$ and ending with $\mathcal{C}_t$. Let $n > 1$. Note that for the 2×2, $1 \times n$, and $n \times 1$ rectangles, there are pairs of start and target configurations where no such sequence exists. For all other rectangles, such configurations do not exist; we provide an $\mathcal{O}(1)$-approximation of the makespan in section 3.

2.2. The continuous case.

For the continuous setting of section 5, we consider N robots $R := \{1, \dots, N\} \subseteq \mathbb{N}$. The Euclidean distance between two points $p, q \in \mathbb{R}^2$ is $|pq| := ||p - q||_2$. We model the robots by disks of radius one. The *position* of a robot is the midpoint of its disk; thus, disjoint robots have a distance of at least 2. Every robot r_i has a *start* and a *target position* $s_i, t_i \in \mathbb{R}^2$ with $|s_i s_j|, |t_i t_j| \geq 2$ for all $i \neq j$. In the following, $d := \max_{r \in R} |s_r t_r|$ is the maximum distance a robot has to cover. A *trajectory* of a robot r_i is a curve $m_i : [0, T_i] \to \mathbb{R}^2$, where $T_i \in \mathbb{R}^+$ denotes the *travel time* of r_i. This curve m_r does not have to be totally differentiable, but must be totally left- and right-differentiable. Intuitively, at any point in time, a robot has unique *past* and *future directions* that are not necessarily identical. This allows the robot to make sharp turns, but does not allow jumps. We bound the speed of the robot by 1, i.e., for each point in time, both left and right derivative of m_r have Euclidean length at most 1. Let $m_i : [0, T_i] \to \mathbb{R}^2$ and $m_j : [0, T_j] \to \mathbb{R}^2$ be two trajectories; w.l.o.g., all travel times are equal to the maximum travel time $T_{\max}$ by extending m_i with $m_i(t) = m_i(T_i)$ for all $T_i < t \leq T_{\max}$. The trajectories m_i and m_j are *compatible* if the corresponding robots do not intersect at any time, i.e., if $|m_i(t)m_j(t)| \geq 2$ holds for all $t \in [0, T_i]$. A *trajectory set* of R is a set of compatible trajectories $\{m_1, \dots, m_N\}$, one for each robot. The *(continuous) makespan* of a trajectory set $\{m_1, \dots, m_N\}$ is defined as $\max_{r_i \in R} T_i$. A trajectory set $\{m_1, \dots, m_N\}$ *realizes* a pair of start and target configurations $\mathcal{S} := (\{s_1, \dots, s_N\}, \{t_1, \dots, t_N\})$ if $m_i(0) = s_i$ and $m_i(T_i) = t_i$ hold for all $i \in R$. We are searching for a trajectory set $\{m_1, \dots, m_N\}$ realizing $\mathcal{S}$ with minimal makespan.

3. Labeled grid permutation.

Let $n_1 \geq n_2 \geq 2$, $n_1 \geq 3$ and let P be an $n_1 \times n_2$ rectangle. In this section, we show that computing the optimal makespan of arbitrarily chosen start and target configurations $\mathcal{C}_s$ and $\mathcal{C}_t$ of $n_1 n_2 \geq k > 1$ robots in P is strongly NP-complete. This is followed by an $\mathcal{O}(1)$-approximation for the makespan.

THEOREM 3.1. *The minimum makespan parallel motion planning problem on a grid is strongly NP-hard.*

Proof. The proof is based on a reduction from the NP-hard problem MONOTONE 3-SAT, which asks to decide whether a Boolean 3-conjunctive normal form (3-CNF) formula φ is satisfiable, where in each clause the literals are either all positive or all negative [31]. All coordinates and the makespan M are constructed to be polynomial in the input size, implying strong NP-hardness. A *fully polynomial-time approximation scheme (FPTAS)* is an $(1 + \varepsilon)$-approximation algorithm with a runtime polynomial in the number of robots and in $\frac{1}{\varepsilon}$. As M is constructed to be polynomial in the number of variables and the number of clauses of φ, the existence of an FPTAS would imply the existence of an efficient algorithm computing schedules with optimal makespan. Thus, there is no FPTAS unless P = NP.

For the remainder of the proof, let φ have n variables $\{x_0, \dots, x_{n-1}\}$ and m clauses $\{C_1, \dots, C_m\}$. From φ, we construct an instance of the minimum makespan parallel motion planning problem that has optimum makespan M if φ is satisfiable and $M + 1$ otherwise. During the description of the construction, we keep M variable, fixing its value once the construction is complete. The structure of the resulting instance is sketched in Figure 2.

Each variable x_j is represented by a *variable robot*. Additionally, for each variable there are two *auxiliary robots* that force the variable robot to take one of two different paths to its goal in any solution with makespan M; see Figure 2. The *left auxiliary robots* start at positions $(1, 6j + 1)$ and move down towards their target positions $(1, 6j + 1 - M)$ in each time step. The *right auxiliary robots* start at positions $(M - 3, -M + 6j + 1)$ and have to move up towards their target positions $(M - 3, 6j + 1)$. The variable robot for variable x_j starts at position $(0, 6j)$ and has to travel $M - 2$ units to the right towards its goal position $(M - 2, 6j)$. In the first time step, each variable robot can either wait or move upwards. Afterwards, it must move to the right in every

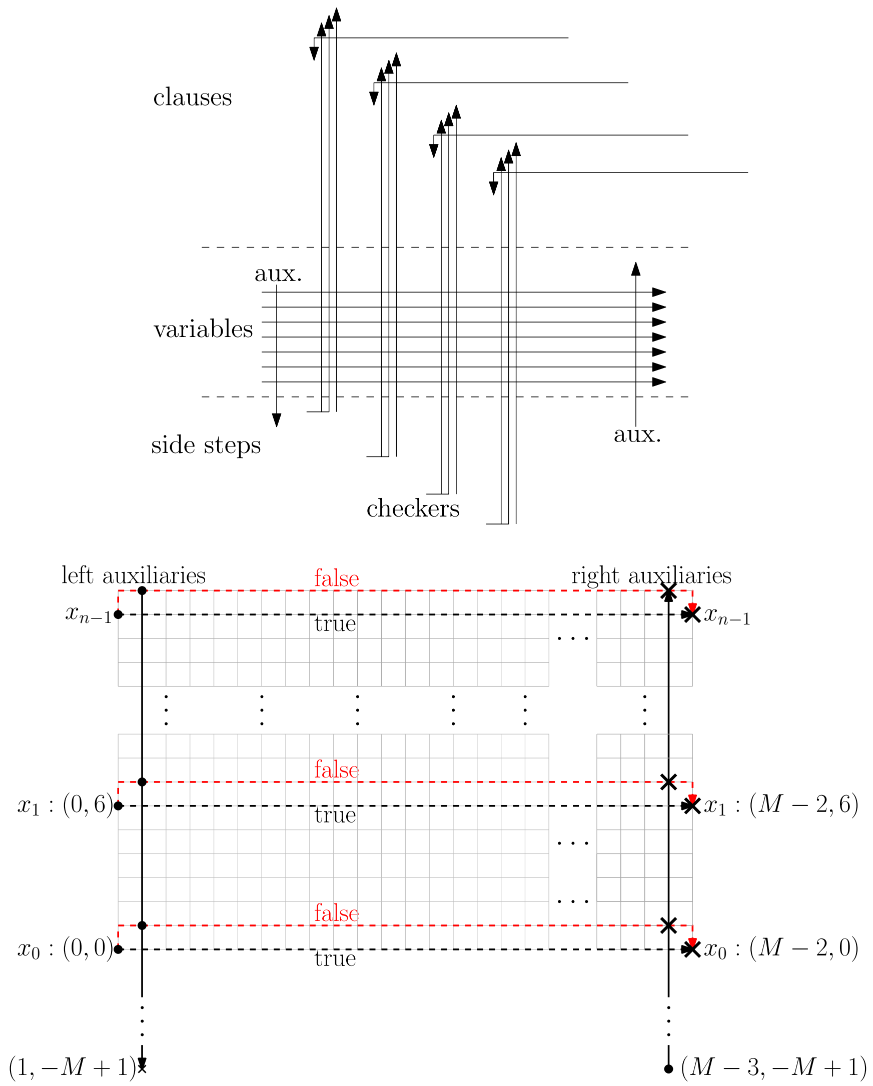


FIG. 2. Top: *The structure of the resulting parallel motion planning problem instance.* Bottom: *Start configuration (disks) and target configuration (crosses) of variable robots and their auxiliaries. The* left auxiliary robot *for x_j starts at position $(1, 6j+1)$ and has to move down towards its target position $(1, 6j+1-M)$ in each time step. The* right auxiliary robot *for x_j starts at $(M-3, -M+6j+1)$ and has to move up towards its target position $(M-3, 6j+1)$.*

time step until passing the right group of auxiliary robots at $x = M-3$. It cannot wait or move down before this point, as this would lead to a collision with the corresponding right auxiliary robot. Therefore in any schedule with makespan M, after the kth time step, each variable robot has x-coordinate $k-1$ for any $1 \le k \le M-3$.

For each clause $C_i = \{x_{j_1}, x_{j_2}, x_{j_3}\}$ with $j_1 < j_2 < j_3$, we have three *checker robots* c_i^1, c_i^2, c_i^3 checking whether their corresponding literal satisfies the clause. The checkers for clause C_i start at positions $\alpha_i^1 := \big(6(ni+j_1), -6ni-f_i\big), \dots, \alpha_i^3 := \big(6(ni+j_3), -6ni-f_i\big)$, where $f_i = 1$ if and only if C_i is negative and $f_i = 0$ otherwise.

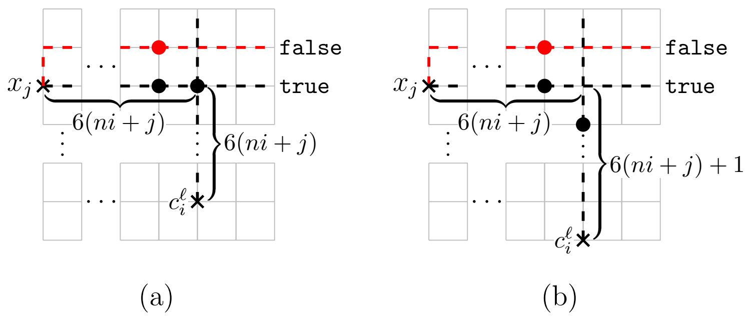


FIG. 3. (a) *A checker c_i^ℓ for variable x_j in a positive clause C_i.* (b) *A checker c_i^ℓ for variable x_j in a negative clause C_i. Checkers must wait if and only if the variable assignment does not match.*

As depicted in Figure 3, a checker has to wait one time step for the corresponding variable if and only if the checked literal is not true.

Checker c_i^3 has to move $M-1$ units up to its target position $t_i^3 := \alpha_i^3 + (0, M-1)$. Let $d_1 := 6(j_3 - j_1)$ be the horizontal distance between the initial positions of c_i^1 and c_i^3, and let $d_2 := 6(j_3 - j_2)$ analogously. Both d_1 and d_2 are always even and at least 6; therefore $s_1 := \frac{d_1}{2} + 2 < d_1$ and $s_2 := \frac{d_2}{2} + 1 < d_2$ are integer. We force c_i^1 to take s_1 steps to the right towards its target position $t_i^1 := \alpha_i^1 + (s_1, M-1-s_1)$. Analogously, c_i^2 has target position $t_i^2 := \alpha_i^2 + (s_2, M-1-s_2)$. Each checker travels a total distance of $M-1$; thus it is allowed to wait for one time step, but has to move on an xy-monotone path towards its target position.

Because moves to the right do not change the position of a checker relative to the variables, we may assume that the checkers move to the right from their initial position before moving up. In fact, we enforce this behavior using auxiliary robots as depicted in Figure 4. Moreover, each clause C_i also has a *clause robot* ensuring that there is at least one satisfied literal. The clause robots start to the right of the checkers and above the variables and have to move $M-2$ units to the left and two units downwards, and therefore have to move towards their target in every round without waiting for the checkers. The clause robot of each clause is placed such that checkers for other clauses cannot interfere with its path; see Figure 2. To be more precise, as shown in Figure 4, the clause robot stops at position $t_i^1 - (3,3)$ and starts at position $t_i^1 + (-1, M-5)$. The vertical offset between the checkers introduced by the side steps that c_i^1 and c_i^2 perform is chosen such that the clause robot can pass through the checkers without waiting if and only if one of the checkers did not wait. This is the case if and only if at least one literal of the clause is satisfied.

It remains to determine the critical makespan M. This critical makespan M must be large enough to allow the checkers of the last clause C_m to pass through the variable robots and their clause robot. Moreover, it must also allow the variable robots to cross paths with all checkers. The checkers of the last variable travel left of the line $x = 6n(m+1) - 6$. Therefore, a makespan $M \geq 6n(m+1)$ suffices for the variable robots. Regarding the clauses, if the last clause is negative, the starting points of its checkers are located on the line $y = -6nm - 1$. The topmost variable robot travels below the line $y = 6(n-1) + 1$. To keep our argument simple, we want to make sure that the clauses stay strictly above all variables. Due to the position of the clauses, this means that we have to ensure that the checker for the first literal of the last clause has target position above the line $y = 6(n-1) + 5$. Therefore, not accounting for the side steps of the checkers, we have to set $M \geq (6nm+1) + (6(n-1)+5) = 6n(m+1)$.

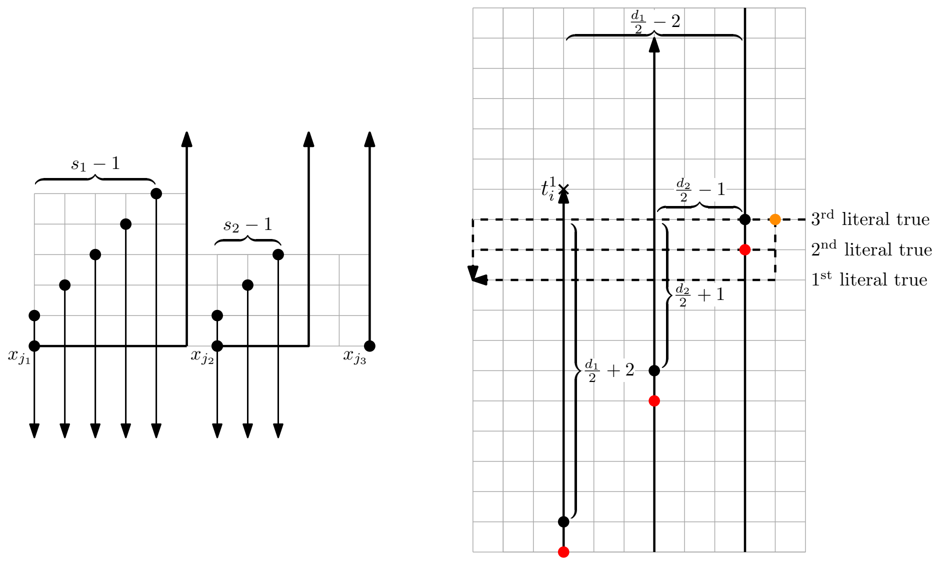


FIG. 4. Left: *A group of auxiliary robots is used to force the first two checkers of each clause to perform their side steps before moving up. Each auxiliary robot has to move downwards by M units.* Right: *A clause robot (orange) meeting the corresponding checkers (black for satisfied checkers, red for nonsatisfied checkers). (Color available online only.)*

Clearly, the number of side steps performed by each checker is less than $6n$. Therefore, in total, a critical makespan of $M := 6n(m + 2)$ is sufficient.

In our construction, a makespan of M is feasible if and only if for every clause robot there is one checker that does not wait, which implies that each clause has a satisfied literal under the assignment induced by the variable robots. Therefore, a makespan of M is feasible if and only if φ is satisfiable.

In summary, we use $3n + 3m + 6n(m + 2)$ robots: n variable robots, $2n$ auxiliary robots for the variable robots, $3m$ checker robots for m clauses, and at most $6n(m+2)$ auxiliary robots for the checker robots.

Finally, observe that even though our reduction uses individually labeled robots, three colors are already sufficient. One can use color 1 for variables, color 2 for checkers, and color 3 for clauses and all auxiliaries. ◻

Our constant-factor approximation is based on an algorithm that computes a schedule with a makespan upper-bounded by $\mathcal{O}(n_1 + n_2)$ described by Lemma 3.3. Based on Lemma 3.3, we give a constant-factor approximation of the makespan; see Theorem 3.4. Finally, we embed the algorithm of Theorem 3.4 into a more general approach to ensure simultaneously a runtime polynomial in the number N of input robots and a constant approximation factor; see Theorem 3.5.

Next, we give the details of the algorithm that computes a sequence of $\mathcal{O}(n_1 + n_2)$ steps transforming an arbitrary start configuration $\mathcal{C}_s$ into an arbitrary target configuration $\mathcal{C}_t$ of an $n_1 \times n_2$ rectangle; see Lemma 3.3. This algorithm is based on a sorting algorithm, called ROTATESORT, that uses swap operations, in which two robots exchange their positions within one single step, as elementary operations. As our model does not allow swap operations, we first have to show how to simulate swap operations at the expense of increasing the makespan by a factor upper-bounded by some constant.

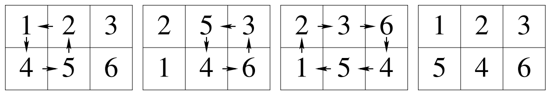


FIG. 5. *Using three moves for swapping two positions in a 2×3 arrangement.*

In order to simulate swap operations, we first observe that Yu and LaValle [86] proved that for a 3×3 square, each start configuration can be transformed into an arbitrary target configuration. This result is easily established for 2×3 rectangles; see Figure 5 for how to realize a transposition.

LEMMA 3.2. *For a pair of start and target configurations $\mathcal{C}_s$ and $\mathcal{C}_t$ of a 2×3 rectangle, we can compute a sequence of at most seven steps transforming $\mathcal{C}_s$ into $\mathcal{C}_t$.*

Lemma 3.2 is the building block for permuting $n_1 \times n_2$ rectangles within makespan $\mathcal{O}(n_1 + n_2)$.

LEMMA 3.3. *For a pair of start and target configurations $\mathcal{C}_s$ and $\mathcal{C}_t$ of an $n_1 \times n_2$ rectangle, we can compute in time polynomial in n_1 and n_2 a sequence of $\mathcal{O}(n_1 + n_2)$ steps transforming $\mathcal{C}_s$ into $\mathcal{C}_t$.*

Proof. The straightforward proof relies on covering the rectangle by a set of disjoint 2×3 and 3×2 rectangles, on which swap operations are performed in parallel, with each swap operation exchanging the position of two adjacent robots. We say that two swap operations are *disjoint* if all four positions of the two swaps are distinct. Although direct swap operations of adjacent robots are not possible, Lemma 3.2 allows us to perform an arbitrary number of pairwise disjoint swap operations within each 2×3 rectangle with $\mathcal{O}(1)$ transformation steps. As illustrated in Figure 6, we cover P by 12 different layers of rectangles, such that each pair of adjacent unit squares from P lies in one of the 2×3 rectangles or in one of the 3×2 rectangles.

In particular, we distinguish between 2×3 and 3×2 rectangles inside the $n_1 \times n_2$ rectangle. Furthermore, we distinguish between different positions of 2×3 rectangles according to row numbers modulo 2 and column numbers modulo 3; see Figures 6(a)–(f). Analogously, we distinguish between different positions of 3×2 rectangles according to row numbers modulo 3 and column numbers modulo 2; see Figures 6(g)–(l). This results in 12 different classes of rectangles.

Given a set S of pairwise disjoint swap operations, we subdivide S into these 12 layers, such that the two robots of each swap operation lie in the same small rectangle of the corresponding layer. Lemma 3.2 implies that all swap operations of one layer can be done in parallel with $\mathcal{O}(1)$ transformation steps. Therefore, all swap operations in S can be done in $\mathcal{O}(1)$ transformation steps.

This allows us to apply a sorting algorithm for $n_1 \times n_2$ meshes, called ROTATESORT [50], whose only elementary steps are swap operations of adjacent cells. We employ ROTATESORT by labeling the robots in the target configuration based on the snake-like ordering guaranteed by ROTATESORT. Applying ROTATESORT to the start configuration with the robots labeled in this way, we obtain the required target configuration. Marberg and Gafni [50] show that ROTATESORT needs $\mathcal{O}(n_1 + n_2)$ phases, where each phase consists of pairwise disjoint swap operations. This leads to $\mathcal{O}(n_1 + n_2)$ transformation steps in our model. □

Based on the algorithm of Lemma 3.3, we can give a constant-factor approximation algorithm.

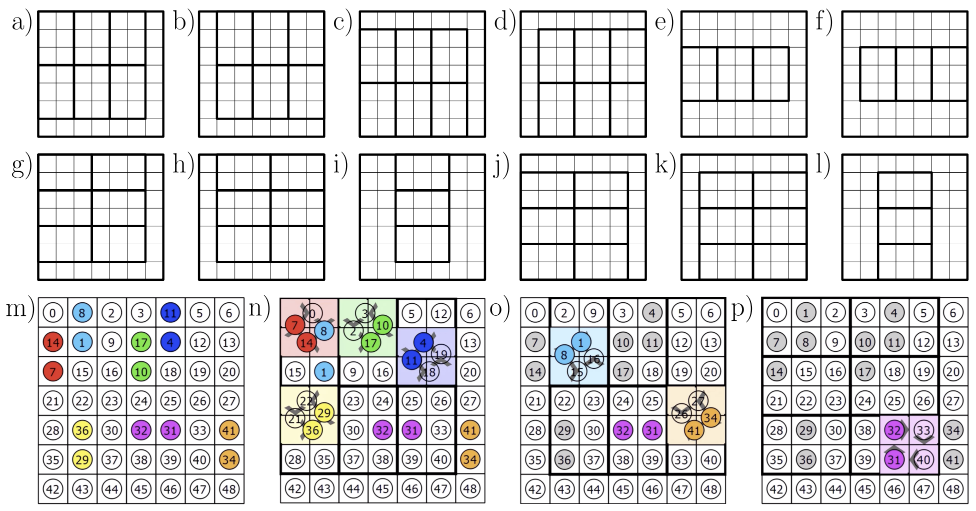


FIG. 6. (a)–(l) *Covering of P by pairwise disjoint 2×3 and 3×2 rectangles in 12 layers.* (m)–(p) *Simulating seven distinct swap operations by iteratively using the classes of rectangles of* (a), (b), *and* (g) *as animated in our video* [6].

THEOREM 3.4. *There is an algorithm with runtime $\mathcal{O}(dn_1n_2)$ that, given an arbitrary pair of start and target configurations of an $n_1 \times n_2$ rectangle with maximum distance d between any start and target position, computes a schedule of makespan $\mathcal{O}(d)$, i.e., an approximation algorithm with constant stretch.*

For the algorithm of Theorem 3.4, Lemma 3.3 is repeatedly applied to rectangles of side length $\mathcal{O}(d)$, resulting in $\mathcal{O}(d)$ transformation steps in total. Because d is a lower bound on the makespan, this yields an $\mathcal{O}(1)$-approximation of the makespan.

At a high level, the algorithm of Theorem 3.4 first computes the maximal $L_{\inf}$ distance d between a robot's start and target positions. Then we partition P into a set T of pairwise disjoint rectangular *tiles*, where each tile $t \in T$ is an $n'_1 \times n'_2$ rectangle for $n'_1, n'_2 \leq 24d - 1 < 24d$. Intuitively speaking, to ensure that each tile has a side length of least $12d$, we partition P into $\lfloor \frac{n_1}{12d} \rfloor$ columns, followed by one row of width between $12d + 1$ and $24d - 1$. Furthermore, we partition each column into $\lfloor \frac{n_1}{12d} \rfloor$ tiles of height $12d$ followed by one tile of height between $12d + 1$ and $24d - 1$; therefore, a tile size of $24d$ is sufficient. Thus, instead of 24, we could have used any other sufficiently large constant. We will analyze that for each side s of a tile t only the three tiles t_1, t_2, t_3 lying on the opposite side of the line induced by s and sharing at least one point with t may contain robots that want to travel from t to t_1, t_2, or t_3 or vice versa; see Figure 7 (left).

Our approach realizes these exchanges via the side s shared between t and t_2, where t_2 is the tile that lies between t_1 and t_3. For each pair (t, t_i) we will exchange d robots between t and t_i or vice versa within one transformation step, implying that $3d$ robots need to pass s within one transformation step. Applying a sequence of $3d$ transformation steps ensures that $\Theta(d^2)$ robots achieve their target step within $\mathcal{O}(d)$ transformation steps. In order to allow this, we need tiles of side length of at least $4 \cdot 3d = 12d$. In particular, pushing $3d$ groups of $3d$ robots simultaneously from t_2 into t within $3d$ transformation steps needs an axis-aligned ring $R \subset t_2$ of width $3d$ and which is adjacent to the boundary of t_2; see Figure 7 (right). Furthermore, realizing each transformation step inside the tile needs another square $t' \subset t_2$ of side length

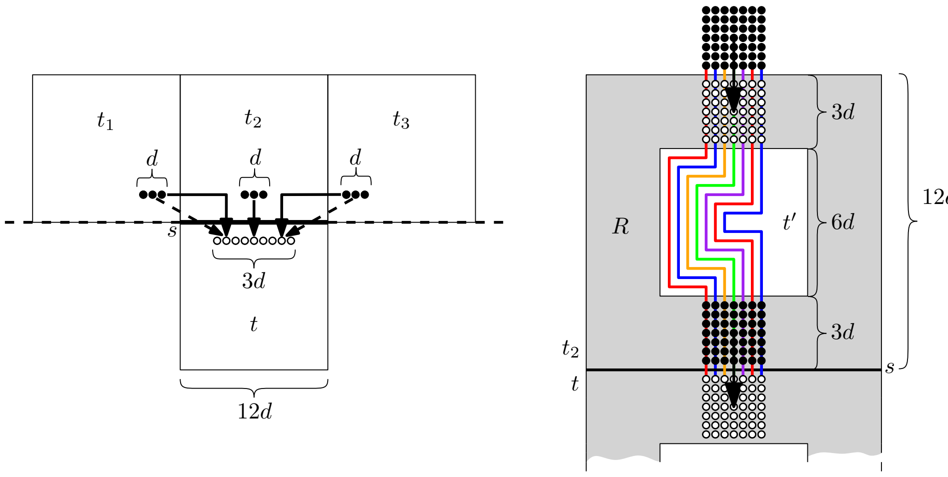


FIG. 7. Left: *A schematic illustration of why at most* $3d$ *robots need to pass in parallel within one transformation step a side* s *of a tile* t. Right: *Pushing* $3d$ *rows of* $3d$ *robots over a side* s *of a tile* t *within* $3d$ *transformation steps needs an axis-aligned ring* $R \subset t_2$ *of width* $12d$ *that is adjacent to the boundary of* t*. Furthermore, realizing each transformation step inside the tile needs another square of side length* $6d$ *that is concentric with* t_2*. (Color available online only.)*

$6d$ that is concentric with t_2; see the colored lines in Figure 7 (right). We then use an algorithm based on flows to compute a sequence of $\mathcal{O}(d)$ transformation steps, ensuring that all robots are in their target tile. Figure 8 shows an example of a pair of start and target configurations and the resulting flow.

Once all robots are in the correct tile, we use Lemma 3.3 simultaneously on all tiles to move each robot to the correct position within its target tile. The details of the algorithm of Theorem 3.4 are given later in this section.

The above-mentioned tiling construction ensures that each square of P belongs to one unambiguously defined tile and that each robot has a *start* and a *target tile*.

Based on the approach of Theorem 3.4 we give a $\mathcal{O}(1)$-approximation algorithm for the makespan with a runtime polynomial in the number N of robots to be moved.

THEOREM 3.5. *There is an algorithm with runtime* $\mathcal{O}(N^3)$ *that, given an arbitrary pair of start and target configurations of a rectangle* P *with* N *robots to be moved and maximum distance* d *between any start and target positions, computes a schedule of makespan* $\mathcal{O}(d)$*, i.e., an approximation algorithm with constant stretch.*

Intuitively speaking, the approach of Theorem 3.5 distinguishes two cases.

(1) Both $\lfloor \frac{n_2}{4} \rfloor$ and the maximum distance d between the robots' start and target positions are lower-bounded by the number N of input robots.

(2) $N > \lfloor \frac{n_2}{4} \rfloor$ or $N > d$.

In case (1), the grid is populated sparsely enough that the robots' trajectories in northern, eastern, southern, and western directions can be done sequentially by four individual transformation sequences.

In case (2), the grid is populated densely enough that the approach of Theorem 3.4 can be applied while guaranteeing a polynomial runtime regarding the number N of robots.

Proof. Our algorithm considers the two cases (1) $N \leq \lceil \frac{n_1}{4} \rceil, d$ and (2) $N > \lceil \frac{n_1}{4} \rceil$ or $N > d$ separately as follows.

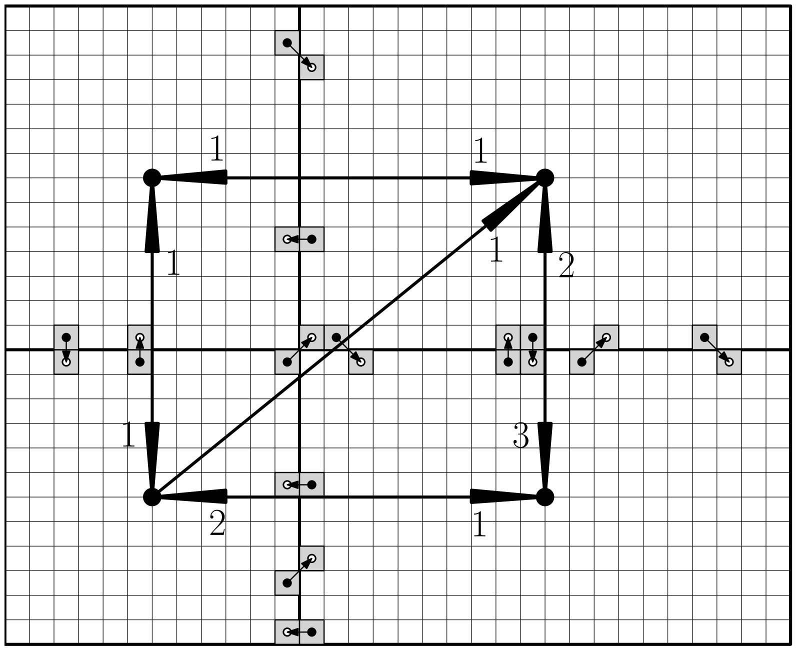


FIG. 8. *A tiling of a 32×26 rectangle by four tiles with $d = 1$. Robots not in their target tile are illustrated by small dots. Their target positions are depicted as white disks. The dual graph of the tiling is illustrated by large dots and directed edges between them. The edges of the dual graph are annotated with the value of the flow on the corresponding edge. In general, it is not guaranteed that robots that have to change the tile lie adjacent to the border between their start and target tiles. However, this is the case for $d = 1$, as illustrated in this figure.*

In case (1), we apply the following approach whose steps, described next, are all realizable because $N \leq \lceil \frac{n_1}{4} \rceil, d$. We assume w.l.o.g. that n_1 and n_2 are even. Otherwise, starting from the start configuration, we move all robots from the last line into the second-to-last line, and all robots from the last column into the second-to-last column within $\mathcal{O}(d)$ transformation steps. The reversed argument implies that there is a sequence of $\mathcal{O}(d)$ transformation steps leading from an even-sized configuration to the target configuration. Thus, from now on, we restrict our considerations to even-sized rectangles.

For each pair of start and target configurations $\mathcal{C}_s$ and $\mathcal{C}_t$ of P, there are two configurations $\mathcal{C}_o$ and $\mathcal{C}_e$ (that are uniquely defined later), such that the two following conditions are fulfilled: (1) the coordinates of the robots in $\mathcal{C}_o$ are odd, and the coordinates of the robots in $\mathcal{C}_e$ are even, and (2) $\mathcal{C}_s$ and $\mathcal{C}_e$ can be transformed into $\mathcal{C}_o$ and $\mathcal{C}_t$ within $\mathcal{O}(d)$ transformation steps. Thus, we still have to give an approach for how $\mathcal{C}_o$ can be transformed into $\mathcal{C}_e$ within $\mathcal{O}(d)$ transformation steps.

First, we ensure in parallel for all robots that they achieve the position that is induced by the x-coordinate of their position in $\mathcal{C}_e$ and the y-coordinate of their position in $\mathcal{C}_o$. We call the corresponding configuration *intermediate configuration* $\mathcal{C}_i$ with *intermediate positions and coordinates*. In order to obtain the intermediate configuration, starting from $\mathcal{C}_o$, we first push in parallel all robots that have to move to the right one position upwards, then move them simultaneously to the right until they arrive at their intermediate x-coordinate. Moreover, we push a robot immediately one position downwards when it reaches its intermediate x-coordinate; see Figure 9.

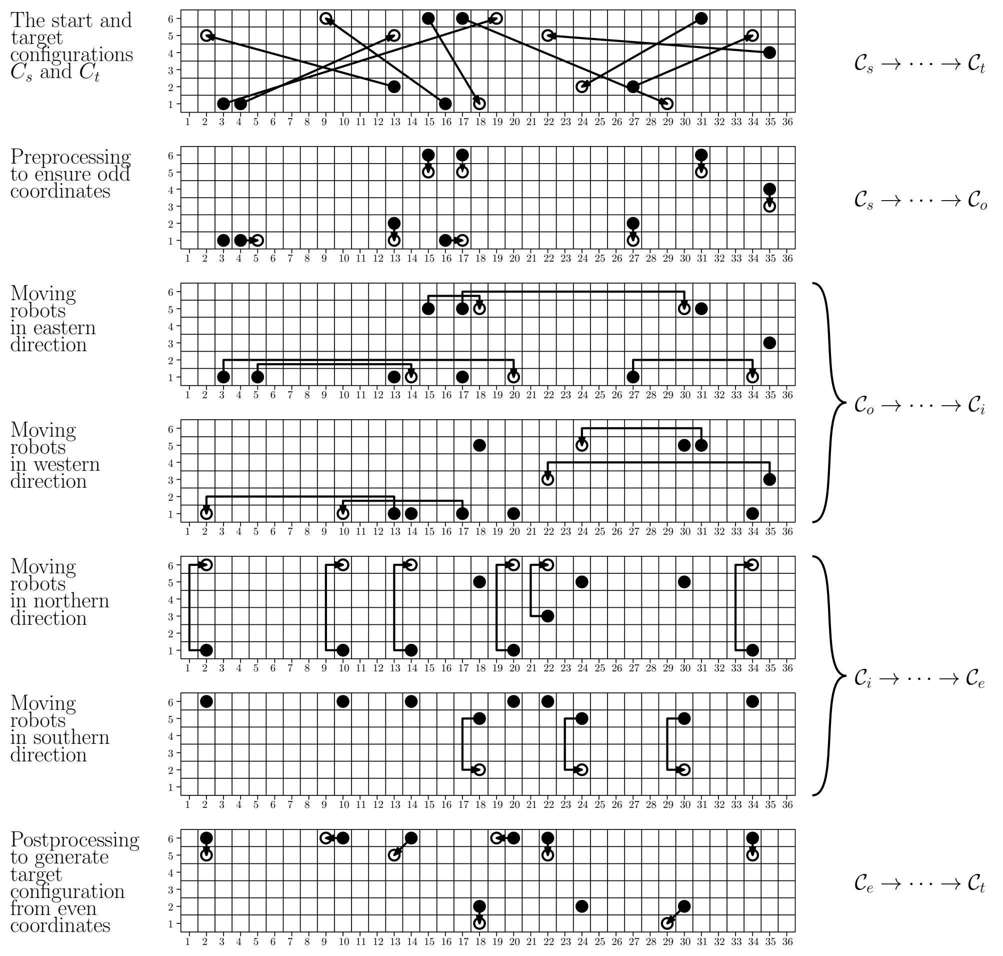

The start and target configurations C_s and C_t
$\mathcal{C}_s \to \cdots \to \mathcal{C}_t$
Preprocessing to ensure odd coordinates
$\mathcal{C}_s \to \cdots \to \mathcal{C}_o$
Moving robots in eastern direction
Moving robots in western direction
$\mathcal{C}_o \to \cdots \to \mathcal{C}_i$
Moving robots in northern direction
Moving robots in southern direction
$\mathcal{C}_i \to \cdots \to \mathcal{C}_e$
Postprocessing to generate target configuration from even coordinates
$\mathcal{C}_e \to \cdots \to \mathcal{C}_t$

FIG. 9. *A stepwise illustration of the approach for case* (1) *of Theorem* 3.5.

After that, we apply the analogous approach for robots that have to move to the left, resulting in the intermediate configuration.

Second, starting from the intermediate configuration, we apply the above-described two-step approach for horizontal movements in an analogous version in order to ensure that the y-coordinate of each robot r is equal to the y-coordinate of r in the configuration $\mathcal{C}_e$, while guaranteeing that the x-coordinate of r stays the same. This results in the configuration $\mathcal{C}_e$.

The transformation steps leading from $\mathcal{C}_s$ to $\mathcal{C}_o$ and leading from $\mathcal{C}_e$ to $\mathcal{C}_t$ can be computed in $\mathcal{O}(N \cdot N)$ time by making use of the fact that the robots' positions are explicitly given via their coordinates. In particular, in each row the robots' positions are processed in increasing order of their x-coordinates, as follows. Initially, all positions are *not reserved*. Consider a robot i with coordinates (x_i, y_i). The *reserved position* of robot i is $(\overline{x}_i, y_i)$, where $\overline{x}_i \geq x_i$ is the smallest odd value such that $(\overline{x}_i, y_i)$ is not reserved. Moving all robots in all rows in parallel to their reserved positions leads to a schedule that ensures that all robots have odd x-coordinates within a makespan no larger than $\mathcal{O}(N)$, which is by assumption no larger than $\mathcal{O}(d)$. Analogously, we construct a schedule that ensures that all robots have odd y-coordinates within a

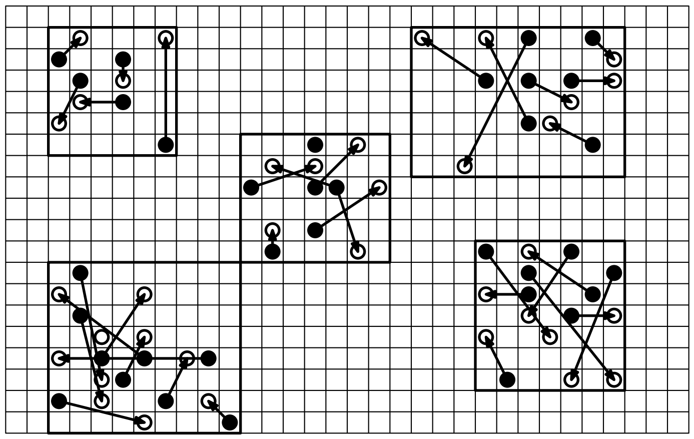


FIG. 10. *An illustration of how the approach for case* (2) *of Theorem* 3.5 *clusters the pairs of robots' start and target positions.*

makespan no larger than $\mathcal{O}(d)$. The same reasoning implies that the sequences of transformation steps leading from $\mathcal{C}_o$ to the intermediate configuration and leading from the intermediate configuration to $\mathcal{C}_e$ can be computed in $\mathcal{O}(N \cdot N)$ time.

In case (2), we apply the approach of Theorem 3.4 as a subroutine in the following approach (illustrated in Figure 10). For each robot we consider the smallest rectangle that contains the robot's start and target positions. If the rectangle has a height or width of 1, we extend the height or width to 2. Now we iteratively replace two rectangles R_1 and R_2 intersecting each other by the smallest rectangle that contains R_1 and R_2.

This results in a set of rectangles that are pairwise intersection free, allowing us to apply the approach of Theorem 3.4 to each resulting rectangle in parallel, while ensuring that each robot is involved in at most one application of the approach of Theorem 3.4.

As the side lengths of the initial rectangles are upper-bounded by d, we conclude that the sum of the lengths of the finally computed rectangles is upper-bounded by $N \cdot d$, which in turn is upper-bounded by N^2 in that case. This implies a runtime of $\mathcal{O}(d \cdot N^2) \leq \mathcal{O}(N^3)$. ∎

In the rest of section 3, we give the proof of Theorem 3.4, i.e., we give an algorithm that computes a schedule with makespan linear in the maximum distance between robots' start and target positions. The remainder of the proof of Theorem 3.4 is structured as follows. In section 3.1 we give an outline of our *flow algorithm* that ensures that each robot reaches its target tile in $\mathcal{O}(d)$ transformation steps. Section 3.2 gives the full details of this algorithm.

3.1. Outline of the approximation algorithm of Theorem 3.4.

We model the trajectories of robots between tiles as a flow f_T, using the weighted directed graph $G_T = (T, E_T, f_T)$, which is dual to the tiling T defined in the previous section. In G_T, we have an edge $(v, w) \in E_T$ if there is at least one robot that has to move from v into w. Furthermore, we define the weight $f_T((v, w))$ of an edge as the integer number of robots that move from v to w. As P is fully occupied, f_T is a *circulation*, i.e., a flow with no sources or sinks, in which flow conservation has to hold at all vertices. Because the side lengths of the tiles are greater than d, G_T is a grid graph with additional diagonal edges and thus has degree at most 8.

The maximum edge value of f_T is $\Theta(d^2)$, but only $\mathcal{O}(d)$ robots can possibly leave a tile within a single transformation step. Therefore, we decompose the flow f_T of robots into a *partition* consisting of $\mathcal{O}(d)$ *subflows*, where each individual robot's motion is modeled by exactly one subflow and each edge in the subflow has value at most d. Thus we are able to *realize* each subflow in a single transformation step by placing the corresponding robots adjacent to the boundaries of their corresponding tiles before we realize the subflow. To facilitate the decomposition into subflows, we first preprocess G_T. In total, the algorithm consists of the following subroutines, elaborated in detail in section 3.2.

Step 1: Compute d, the tiling T, and the corresponding flow G_T.
Step 2: Preprocess G_T in order to remove intersecting and bidirectional edges.
Step 3: Compute a partition into $\mathcal{O}(d)$ d-subflows.
Step 4: Realize the $\mathcal{O}(d)$ subflows using $\mathcal{O}(d)$ transformation steps.
Step 5: Simultaneously, apply Lemma 3.3 to all tiles, moving each robot to its target position.

3.2. Details of the approximation algorithm of Theorem 3.4.

In this section we give more detailed descriptions only for Steps 1–4 because Step 5 is a trivial application of Lemma 3.3 to all tiles in parallel.

3.2.1. Step 1: Compute d, the tiling T, and the corresponding flow G_T.

The maximal distance between robots' start and target positions can be computed in a straightforward manner.

For the tiling, we assume that the rectangle P is axis-aligned and that its bottom-left corner is $(0, 0)$. We consider $k_v := \lfloor \frac{n_1}{12d} \rfloor$ vertical lines $\ell^v_1, \ldots, \ell^v_{k_v}$ with x-coordinate modulo $12d$ equal to 0. As already noted before, the constant 12 can be replaced by any sufficiently large constant value. Analogously, we consider $k_h := \lfloor \frac{n_2}{12d} \rfloor$ horizontal lines $\ell^h_1, \ldots, \ell^h_{k_h}$ with y-coordinate modulo $12d$ to 0. Finally, we consider the tiling of P that is induced by the arrangement induced by $\ell^v_1, \ldots, \ell^v_{k_v-1}, \ell^h_1, \ldots, \ell^h_{h_v-1}$ and the boundary of P; see Figure 8. This implies that the side length of a tile is upper-bounded by $24d - 1$.

Finally, computing the flow G_T is straightforward by considering the tiling T and the robots' start and target positions.

3.2.2. Step 2: Ensuring planarity and unidirectionality.

After initialization, we preprocess G_T, removing edge intersections and bidirectional edges by transforming the start configuration C_s into an intermediate start configuration C'_s, obtaining a planar flow without bidirectional edges. This transformation consists of two steps: (1) ensuring planarity and (2) ensuring unidirectionality.

Step (1). We observe that edge crossings only occur between two diagonal edges with adjacent source tiles, as illustrated in Figures 11(a),(b). To remove a crossing, it suffices to eliminate one of the diagonal edges by exchanging robots between the source tiles. To eliminate all crossings, each robot is moved at most once, because after moving, the robot no longer participates in a diagonal edge. Thus, all necessary exchanges can be done in $\mathcal{O}(d)$ steps by Lemma 3.3, covering the tiling T by constantly many layers, similarly to the proof of Lemma 3.3.

Step (2). We delete a bidirectional edge $(v, w), (w, v)$ by moving $\min\{f_T((v, w)), f_T((w, v))\}$ robots with target tile w from v to w and vice versa, which achieves that $\min\{f_T((v, w)), f_T((w, v))\}$ robots reach their target tile w and $\min\{f_T((v, w)), f_T((w, v))\}$ robots reach their target tile v, thus eliminating the edge

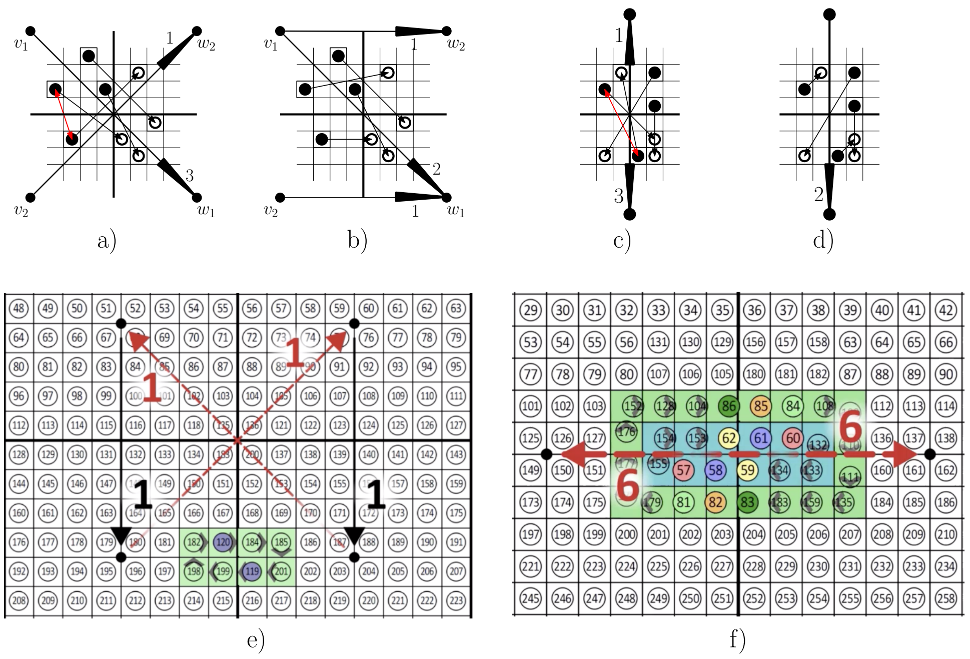


Fig. 11. *Illustration of the preprocessing (step* (1)*: before and after removing crossing edges* (a),(b)*, and step* (2)*: before and after removing bidirectional edges* (c),(d)*). The red arrows indicate how robots change their positions during the preprocessing steps.* (e),(f) *Removing crossings and bidirectional edges is performed by placing the corresponding robots in queues adjacent to the boundary between the two corresponding tiles, as animated in our video* [6]*. (Color available online only.)*

with lower flow value. This process is depicted in Figures 11(c),(d). As in step (1), this can be done in $\mathcal{O}(d)$ parallel steps by Lemma 3.3. As we do not add any edges, we maintain planarity during step (2). Observe that during the preprocessing, we do not destroy the grid structure of G_T.

Steps (1) and (2) maintain the flow property of f_T without any other manipulations to the flow f_T, because both preprocessing steps can be represented by local circulations.

3.2.3. Step 3: Computing a flow partition. After preprocessing, we partition the flow G_T into d-subflows.

Definition 3.6. *A* subflow *of G_T is a circulation $G'_T = (T, E', f'_T)$, such that $E' \subseteq E_T$, and $0 \leq f'_T(e) \leq f_T(e)$ for all $e \in E'$. If $f'_T(e) \leq z$ for all $e \in E'$ and some $z \in \mathbb{N}$, we call G'_T a z-flow.*

The flow partition relies on an upper bound on the maximal edge weight in G_T. By construction, tiles have side length at most $24d$; therefore, each tile consists of at most $576d^2$ unit squares. This yields the upper bound in the following observation; a tighter constant factor can be achieved using a more sophisticated argument.

Observation 3.7. *We have $f_T(e) \leq 576d^2$ for all $e \in E_T$.*

Definition 3.8. *A (z, ℓ)-partition of the flow G_T is a set of ℓ z-subflows $\{G_1 = (V_1, E_1, f_1), \ldots, G_\ell = (V_\ell, E_\ell, f_\ell)\}$ of G_T, such that $G_1, \ldots, G_\ell$ sum up to G_T.*

Lemma 3.9. *We can compute a $(d, \mathcal{O}(d))$-partition of G_T in polynomial time.*

Proof. In a slight abuse of notation, throughout this proof, the elements in sets of cycles are not necessarily unique. A $(d, \mathcal{O}(d))$-partition can be constructed using the following steps.

- We start by computing a $(1, h)$-partition $\mathbb{C}_{\bigcirc}$ of G_T consisting of $h \leq n_1 n_2$ cycles. This is possible because G_T is a circulation. If a cycle C intersects itself, we subdivide C into smaller cycles that are intersection-free. Furthermore, h is clearly upper bounded by the number of robots $n_1 n_2$, because every robot can contribute only 1 to the sum of all edges in G_T. As the cycles do not self-intersect, we can partition the cycles $\mathbb{C}_{\bigcirc}$ by their orientation, obtaining the set $\mathbb{C}_{\circlearrowright}$ of clockwise and the set $\mathbb{C}_{\circlearrowleft}$ of counterclockwise cycles.
- We use $\mathbb{C}_{\circlearrowright}$ and $\mathbb{C}_{\circlearrowleft}$ to compute a $(1, h')$-partition $\mathbb{C}^1_{\circlearrowright} \cup \mathbb{C}^2_{\circlearrowright} \cup \mathbb{C}^1_{\circlearrowleft} \cup \mathbb{C}^2_{\circlearrowleft}$ with $h' \leq n_1 n_2$, such that two cycles from the same subset $\mathbb{C}^1_{\circlearrowright}$, $\mathbb{C}^2_{\circlearrowright}$, $\mathbb{C}^1_{\circlearrowleft}$, or $\mathbb{C}^2_{\circlearrowleft}$ share a common orientation. Furthermore, we guarantee that two cycles from the same subset are either edge-disjoint or one lies nested in the other. A partition such as this can be constructed by applying a recursive peeling algorithm to $\mathbb{C}_{\circlearrowright}$ and $\mathbb{C}_{\circlearrowleft}$ as depicted in Figure 12, yielding a decomposition of the flow induced by $\mathbb{C}_{\circlearrowright}$ into two cycle sets $\mathbb{C}^1_{\circlearrowright}$ and $\mathbb{C}^2_{\circlearrowright}$, where $\mathbb{C}^1_{\circlearrowright}$ consists of clockwise cycles and $\mathbb{C}^2_{\circlearrowright}$ consists of counterclockwise cycles, and a similar partition of $\mathbb{C}_{\circlearrowleft}$. In particular, we apply the following approach iteratively to $\mathbb{C}_{\circlearrowright}$: we consider the union A of the area bounded by the cycles from $\mathbb{C}_{\circlearrowright}$. We remove a flow value of 1 from all edges of the outer boundary component of A. In particular, we add the corresponding 1-subflow G_1 to $\mathbb{C}^1_{\circlearrowright}$ and remove G_1 from $\mathbb{C}_{\circlearrowright}$. Analogously, we remove 1-subflows from $\mathbb{C}_{\circlearrowright}$ that are induced by inner boundary components and add these 1-subflows to $\mathbb{C}^2_{\circlearrowright}$.
- Afterwards, we partition each set $\mathbb{C}^1_{\circlearrowright}$, $\mathbb{C}^2_{\circlearrowright}$, $\mathbb{C}^1_{\circlearrowleft}$, and $\mathbb{C}^2_{\circlearrowleft}$ into $\mathcal{O}(d)$ subsets, each inducing a d-subflow of G_T. This can be done as follows. Let $\mathbb{C} \in \{\mathbb{C}^1_{\circlearrowright}, \mathbb{C}^2_{\circlearrowright}, \mathbb{C}^1_{\circlearrowleft}, \mathbb{C}^2_{\circlearrowleft}\}$. Recall that every pair of cycles from $\mathbb{C}$ consists either of one cycle nested inside the other or of edge-disjoint cycles. The cycles induce a dual forest $D = (\mathbb{C}, E_D)$, where a cycle v has a child w if and only if w lies inside v and there is no other cycle lying in v that w lies in.

  We label the cycles by their depth in D modulo $576d$ and let G_i be the flow induced by all cycles carrying label i, thus obtaining $\mathcal{O}(d)$ subflows G_i.

Finally, we show that each subflow G_i obtained in this way is a d-subflow of G_T. To this end, we observe the following. Let $e \in E_T$ be an arbitrarily chosen edge and let $v, w \in \mathbb{C}$ be two cycles sharing e. This implies that v and w lie nested inside of each other; w.l.o.g., assume that w lies inside v. Thus, in D, v lies on the path from w to its root, and e is contained in all cycles on the path between v and w. On the other hand, due to Observation 3.7, all cycles containing e lie on a path of length at most $576d^2$ in D. Therefore, e has a weight of at most $\frac{576d^2}{576d} = d$ in each G_i, and G_i is a d-subflow. ∎

3.2.4. A subroutine of Step 4: Realizing a single subflow. In this section, we present a procedure for *realizing* a single d-subflow G'_T of G_T.

DEFINITION 3.10. *A schedule* $t := C_1 \to \cdots \to C_{k+1}$ realizes *a subflow* $G'_T = (T, E', f'_T)$ *if, for each pair* v, w *of tiles, the number of robots moved by* t *from their start tile* v *to their target tile* w *is* $f'_T((v, w))$*, where we let* $f'_T((v, w)) = 0$ *if* $(v, w) \notin E'$.

LEMMA 3.11. *Let* $G'_T = (T, E'_T, f'_T)$ *be a planar unidirectional d-subflow. There is a polynomial-time algorithm that computes a schedule* $C_1 \to \cdots \to C_{k+1}$ *realizing* G'_T *for a constant* $k \in \mathcal{O}(1)$.

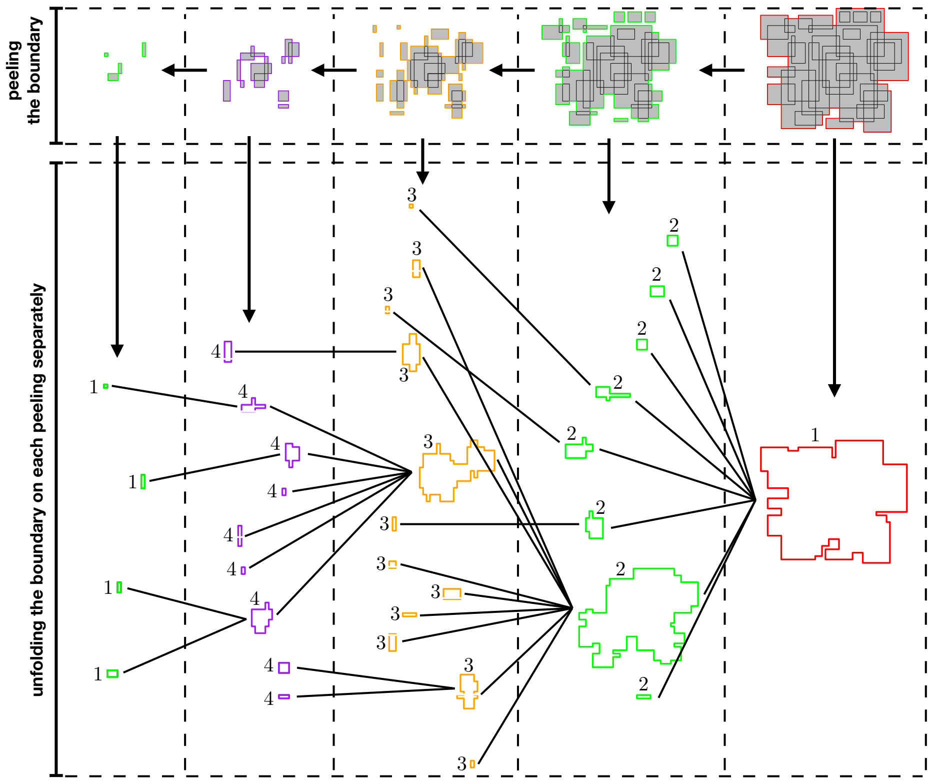


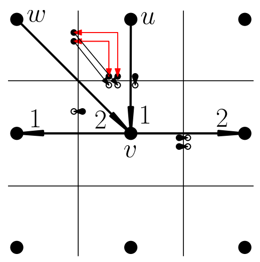
FIG. 12. Top: *Recursive peeling of the area bounded by the cycles from* $\mathbb{C}_{\circlearrowright}$. *Note that an edge* e *vanishes when* $f_T(e)$ *cycles containing that edge are removed by the peeling algorithm described above.* Bottom: *Unfolding cycles that are peeled simultaneously, combined with the nested structure of the peeled boundaries. This leads to a tree, as animated in our video* [6].

Proof. Our algorithm uses $k = \mathcal{O}(d)$ preprocessing steps $C_1 \to \cdots \to C_k$, as depicted in Figures 13(a),(b), and one final realization step $C_k \to C_{k+1}$, shown in Figure 13(c), moving the robots from their start tiles into their target tiles.

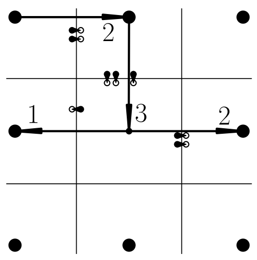
The preprocessing replaces diagonal edges by pairs of orthogonal edges—see the red arrows in Figure 13(a)—and places the moving robots next to the border of their target tiles. Note that the replacements of the diagonal edges cannot be done as part of the preprocessing of Step 2. This is because the replaced diagonal edges may be part of circular flows that cannot be realized locally, as is done for crossing or bidirectional edges in Step 2 of our algorithm.

For the final realization step we compute a pairwise disjoint matching between incoming and outgoing robots, such that each pair is connected by a tunnel inside the corresponding tile in which these tunnels do not intersect each other; see Figure 13(a). The final realization step is given via the robots' motion induced by moving each robot into the interior of the tile and by moving this one-step motion through the corresponding tunnel into the direction of the corresponding outgoing robot.

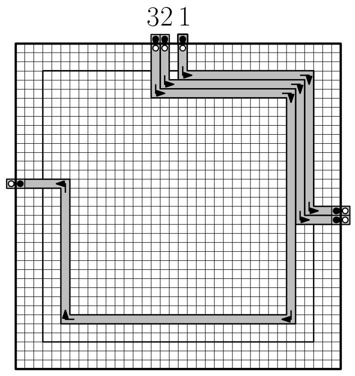
The preprocessing steps $\mathcal{C}_1 \to \cdots \to \mathcal{C}_k$. Let v be an arbitrary tile. We place all robots corresponding to horizontal and vertical edges (v, w) of G'_T in a row adjacent to the side shared by v and w. We can do this for all tiles using $\mathcal{O}(d)$ parallel steps by applying Lemma 3.3.



(a) Preprocessing of diagonal edges.



(b) Configuration and flow after preprocessing.



(c) A crossing-free matching of incoming and outgoing robots and the connecting paths inside the corresponding tile, for $d = 3$.

FIG. 13. *Procedure for computing transformation steps that realize a d-subflow. Figures* (a) *and* (b) *illustrate how we preprocess G'_T such that E'_T consists of horizontal and vertical edges only. Figure* (c) *illustrates the main approach. White disks illustrate start positions, and black disks illustrate target positions.*

Next, we eliminate diagonal edges $(w, v) \in E'_T$ as follows. There are two tiles sharing a side with both w and v; let u be one of them. First we place the $f'_T((w, v))$ robots with start tile w and target tile v in a row next to the side between w and u. Then, we move them to u by exchanging them with $f'_T((w, v))$ robots with both start and target tile u that lie next to the side between u and v, as shown in Figure 13(a). In the resulting flow, the diagonal edge (w, v) with weight $f'_T((w, v))$ is replaced by adding a flow of value $f'_T((w, v))$ on the edges $(w, u), (u, v)$.

We process all tiles as described above in two parallel phases by applying the approach of Lemma 3.3 twice: first on all rows with even index and then on all rows with odd index, thus ensuring that parallel applications of Lemma 3.3 do not interfere with each other.

The realization step $\mathcal{C}_k \to \mathcal{C}_{k+1}$. Let t be an arbitrary tile. For the transformation step $\mathcal{C}_k \to \mathcal{C}_{k+1}$, we need a matching between incoming and outgoing robots of t, such that there is a set of nonintersecting paths in t connecting each incoming robot with its corresponding outgoing robot. As illustrated in Figure 13(c), these paths induce the required transformation $\mathcal{C}_k \to \mathcal{C}_{k+1}$.

We compute this matching by selecting an incoming robot r_{in} and matching it to a robot r_{out}, such that there is a path $p \subseteq \partial t$ between r_{in} and r_{out} that does not touch another incoming or outgoing robot. We remove the matched robots from consideration and repeat the matching procedure until no further unmatched robots exist.

The nonintersecting paths between the positions of the matched robots are constructed as follows. For $i \geq 1$, the ith *hull* of t is the union of all squares on the boundary of the rectangle remaining after the hulls $1, \ldots, i - 1$ are removed. The path between r_{in} and r_{out} consists of three pieces, as shown in Figure 13(c). Let ∂t be the boundary of t. For the ith matched pair of robots, the initial and the last part of the path are straight line segments orthogonal to ∂t, from the position of r_{in} to the $(d + i)$th hull and from the $(d + i)$th hull towards the position of r_{out}. The main part of the path lies on the $(d+i)$th hull, connecting the end of the initial part to the beginning of the last part. ∎

3.2.5. Step 4: Realizing all subflows. In the following we extend the idea of Lemma 3.11 to $\ell \leq d$ subflows instead of one and demonstrate how this can be leveraged to move all robots to their target tile using $\mathcal{O}(d)$ transformation steps.

LEMMA 3.12. *Let $\mathcal{S} := \langle G_1 = (V_1, E_1, f_1), \ldots, G_\ell = (V_\ell, E_\ell, f_\ell)\rangle$ be a sequence of $\ell \leq d$ unidirectional planar d-subflows of G_T. There is a polynomial-time algorithm computing $\mathcal{O}(d) + \ell$ transformation steps $C_1 \to \cdots \to C_{k+\ell}$ realizing $\mathcal{S}$.*

Proof. Let t be an arbitrary tile. Similarly to the approach of Lemma 3.11, we first apply a preprocessing step guaranteeing that the robots to be moved into or out of t are in the right position close to the boundary of t; see Figure 14(a). Thereafter we move the robots into their target tiles, using ℓ applications of the algorithm from Lemma 3.11 without the preprocessing phase; see Figure 14(a). In particular, we realize a sequence of ℓ d-subflows by applying ℓ times the single realization step of the algorithm from Lemma 3.11.

In order to ensure that a sequence of ℓ realization steps from Lemma 3.11 without intermediate preprocessing steps realizes a sequence of ℓ d-subflows, we apply the following $\mathcal{O}(d)$ preprocessing steps for all ℓ realization steps in advance: For each side of the tile t, we place all leaving or entering robots that belong to the same subflow in a common row and stack these rows in the order which is induced by the sequence of the subflows to be realized; see Figure 14(a). Finally, pushing all stacked robots downwards into the direction of the boundary ∂t of the tile ensures that processing one realization step implies that all robots involved in the following realization step lie in a row adjacent to ∂t; see Figure 14.

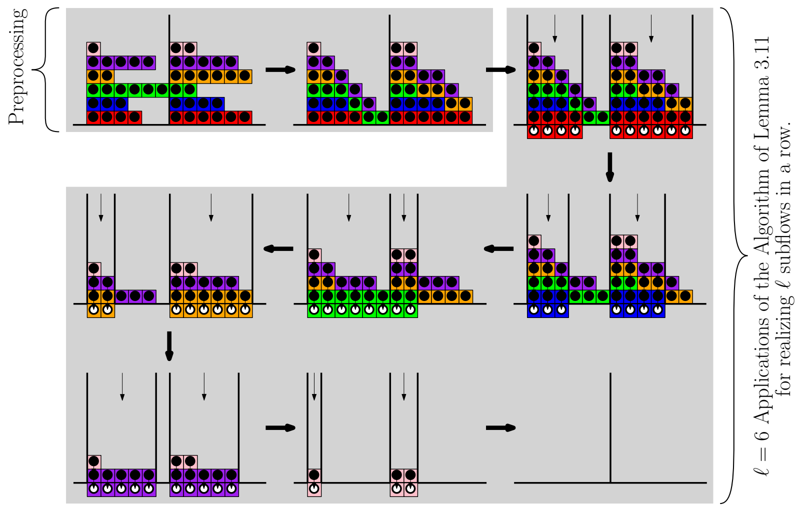
In the following we describe how we place the robots in their start tiles as a preprocessing step. First, we use the same preprocessing step as in Lemma 3.11 to eliminate diagonal edges. For a simplified illustration, we describe the remainder of the preprocessing in two steps that can be realized by just one application of Lemma 3.3. After elimination of diagonal edges, we proceed by stacking the rows of robots moving out of t in the order in which the subflows are to be processed; see Figure 14(a). Then we push the robots towards the boundary ∂t of their start tile until they meet either ∂t or another moving robot. See Figure 14(a), image 2, for an example.

The above described preprocessing ensures that after each application of the algorithm of Lemma 3.11, all robots moving out of t in the next transformation step lie in a row adjacent to ∂t. Therefore this preprocessing can be used to replace the preprocessing done in Lemma 3.11. For an example, see Figure 14(a), images 3–9.

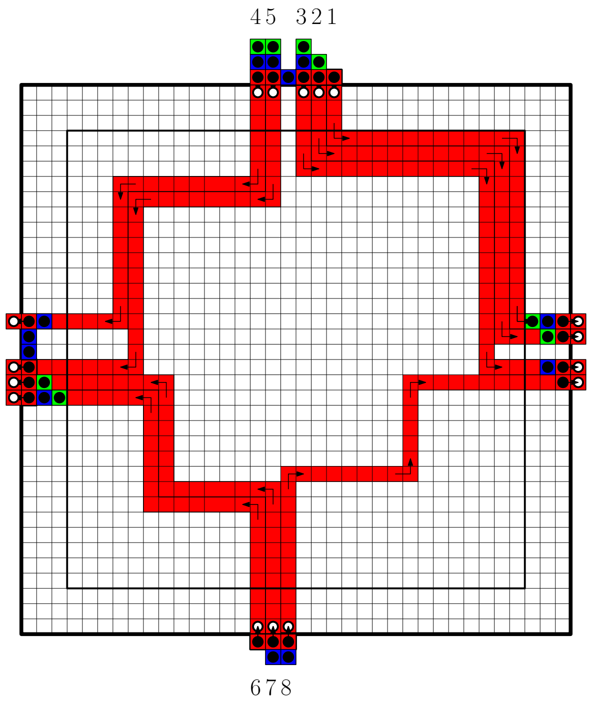
As $\ell \leq d$, the stacked rows have a height of at most d. Thus, they are contained in hulls 1 to d. Therefore, and because the flows are unidirectional and diagonals are eliminated, the structure of the stacks is not damaged by the applications of Lemma 3.11, allowing us to realize $\ell \leq d$ subflows in $\mathcal{O}(d)$ transformation steps instead of one. ∎

LEMMA 3.13. *There is a polynomial-time algorithm computing $\mathcal{O}(d)$ transformation steps moving all robots into their target tiles.*

Proof. By Lemma 3.9, we can compute a (d, cd)-partition of G_T for $c \in \mathcal{O}(1)$. We group the corresponding d-subflows into $\frac{cd}{d} = c$ sequences, each consisting of at most d d-subflows. We realize each sequence by applying Lemma 3.12, using $\mathcal{O}(d)$ transformation steps for each sequence. This leads to $\mathcal{O}(cd) = \mathcal{O}(d)$ steps for realizing all sequences of d-subflows. ∎


Preprocessing
$\ell = 6$ Applications of the Algorithm of Lemma 3.11 for realizing ℓ subflows in a row.

(a) Stacking the rows of robots corresponding to the flow values on the edges of the subflows to be realized.



(b) The realization of a single subflow and the corresponding matching between five incoming and five outgoing robots involved in that subflow.

FIG. 14. *Realizing a sequence of subflows by stacking the rows of robots to be moved onto each other in the order in which the subflows are realized.*

For the proof of Theorem 3.4, we still need to analyze the time complexity of our approach, which we do next.

Proof of Theorem 3.4. The steps of our algorithm have the following time complexity.

Initialization Step 1: Computing d, T, and G_T is possible in $\mathcal{O}(n_1 n_2)$ time.

Steps 2 and 5: The application of Lemma 3.3 requires $\mathcal{O}(d^3)$ time for each tile, so these steps can be done in $\mathcal{O}(dn_1 n_2)$ time.

Step 3: All subroutines of Step 3 can be done in an overall time of $\mathcal{O}(n_1 n_2)$. In particular, the $(1, h)$-partition $\mathbb{C}_{\bigcirc}$ of G_T can be computed in $\sum_{e \in E_T} f_T(e) \in \mathcal{O}(n_1 n_2)$ time by a simple greedy algorithm. The number of edges in all cycles from $\mathbb{C}_{\bigcirc}$ combined is at most $n_1 n_2$, which is the number of robots in P. Thus, resolving self-intersections of cycles in $\mathbb{C}_{\bigcirc}$ can be done in $\mathcal{O}(n_1 n_2)$ time. As $|\mathbb{C}_{\bigcirc}| \in \mathcal{O}(n_1 n_2)$, the partition of $\mathbb{C}_{\bigcirc}$ into $\mathbb{C}^1_{\circlearrowleft}$, $\mathbb{C}^2_{\circlearrowleft}$, $\mathbb{C}^1_{\circlearrowright}$, and $\mathbb{C}^2_{\circlearrowright}$ takes time $\mathcal{O}(n_1 n_2)$. Furthermore, the partitioning of $\mathbb{C}^1_{\circlearrowleft}$, $\mathbb{C}^2_{\circlearrowleft}$, $\mathbb{C}^1_{\circlearrowright}$, and $\mathbb{C}^2_{\circlearrowright}$ into $\mathcal{O}(d)$ d-subflows can be done in time $\mathcal{O}(n_1 n_2)$.

Step 4: The parallel applications of Lemma 3.3 to disjoint rectangles can be computed in $\mathcal{O}(dn_1 n_2)$. Furthermore, the construction of all connecting paths between incoming and outgoing robots for all tiles needs $\mathcal{O}(dn_1 n_2)$ time per application of the algorithm of Lemma 3.12. By applying Lemma 3.12 constantly many times, Step 4 needs $\mathcal{O}(dn_1 n_2)$ time. ∎

4. Variants on labeling.

A different version is the unlabeled variant, in which all robots are the same. A generalization of both this and the labeled version arises when robots belong to one of k color classes, with robots from the same color class being identical.

We formalize this variant by using a coloring $c : \{1, \dots, n_1 n_2\} \to \{1, \dots, k\}$ for grouping the robots. By populating unoccupied cells with robots carrying color $k+1$, we may assume that each unit square in the environment P is occupied. The robots *draw an image* $I = \left(I^1, \dots, I^k\right)$, where I^i is the set of cells occupied by a robot with color i. We say that two images I_s and I_t are *compatible* if in I_s and I_t the number of cells colored with color i is equal for each color $i = 1, \dots, k$. By moving the robots, we want to transform a start image I_s into a compatible target image I_t, minimizing the makespan.

THEOREM 4.1. *There is an algorithm with runtime $\mathcal{O}(k(N)^{1.5} \log(N) + N^3)$ for computing, given start and target images I_s, I_t with maximum distance d between start and target positions, an $\mathcal{O}(1)$-approximation of the optimal makespan M, and a corresponding schedule.*

Proof. We transform the input into an instance of the labeled variant, such that an $\mathcal{O}(1)$-approximation for the labeled instance provides an $\mathcal{O}(1)$-approximation for the colored instance. For each color i, we consider the two point sets $A^i, B^i \subset \mathbb{R}^2$, where A^i contains the center points a^i_v of all unit squares $v \in I^i_s$ and B^i contains the center points b^i_v of all $v \in I^i_t$.

A *bottleneck matching* between A^i and B^i is a perfect matching between A^i and B^i that minimizes the maximal distance. The cost of an optimal bottleneck matching between A^i and B^i is in $\mathcal{O}(M)$, because a transformation sequence induces a bottleneck matching on all color classes. Efrat, Itai, and Katz [24] show that the geometric bottleneck matching problem can be solved in $\mathcal{O}(|A + B|^{1.5} \log |A + B|)$ time.

A set of k bottleneck matchings between the sets A^i and B^i induces labeled start and target configurations C_s, C_t. Applying the algorithm from section 3 to these yields a sequence of transformation steps of length $\mathcal{O}(M)$. ∎

5. Continuous motion. In this section, we consider the continuous geometric case in which the robots are identical geometric objects that have to move into a target configuration in the plane without overlapping at any point in time. We want to minimize the makespan under these conditions, where the velocity of each robot is bounded by 1.

5.1. A lower bound for unbounded environments. In this section we give a worst-case lower bound of $\Omega(N^{1/4}d)$ for the continuous makespan where N is the number of robots. To be more precise, we construct a pair of start and target configurations of N robots as illustrated in Figure 15(a). In this instance, we have $d = 2$. In Theorem 5.8, we show that the optimal continuous makespan of this instance is in $\Omega(N^{1/4})$, yielding the worst-case lower bound stated above.

More formally, let $\{m_1, \dots, m_N\}$ be an arbitrary trajectory set with makespan M, realizing the start and target configurations as illustrated in Figure 15(a). By applying a simple continuity argument, we show that there must be a point in time $t \in [0, M]$ such that the area of the convex hull $Conv(m_1(t), \dots, m_N(t))$ of $m_1(t), \dots, m_N(t)$ is lower bounded by $cN + \Omega(N^{3/4})$, where cN is the area of $Conv(m_1(0), \dots, m_N(0))$. Assume $M \in o\left(N^{1/4}\right)$ and consider the area of the convex hull $Conv(m_1(t'), \dots, m_N(t'))$ of $m_1(t'), \dots, m_N(t')$ at some point $t' \in [0, M]$. This area is at most $cN + \mathcal{O}(\sqrt{N}) \cdot o\left(N^{1/4}\right)$, because asymptotically, the area gained during the movement is bounded by the product of makespan and circumference. This contradicts the lower bound stated above.

A key ingredient for the construction of the time point $t \in [0, M]$ is the fact that the distance between the centers of two robots changes continuously. In fact, we know that the Euclidean distance between two centers is 2-*Lipschitz* because the velocity of the robots is bounded by 1.

DEFINITION 5.1. *A function $f : \mathbb{R} \to \mathbb{R}$ is λ-Lipschitz (continuous) if $|f(x) - f(y)| \leq \lambda |x - y|$ holds for all $x, y \in \mathbb{R}$.*

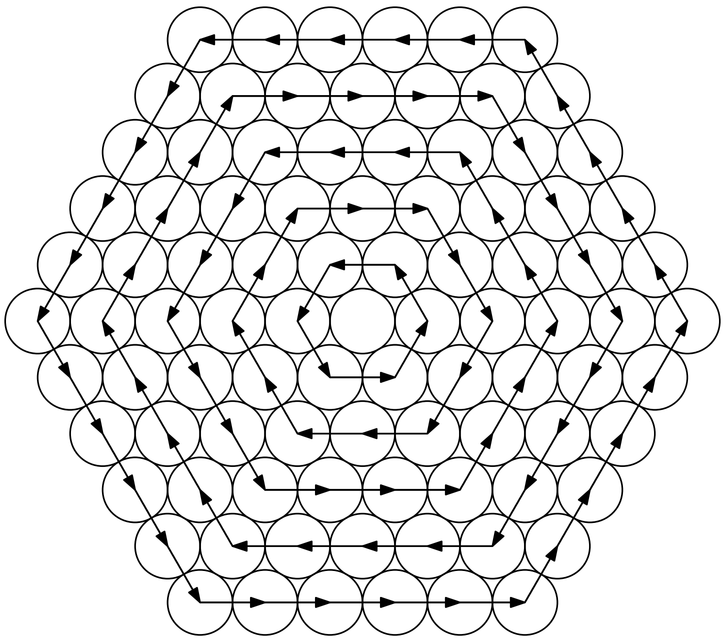
OBSERVATION 5.2. *For all $i, j \in R$, the distance between the centers $m_i(\cdot)$ and $m_j(\cdot)$ of robots i and j is 2-Lipschitz.*

Let V be the Voronoi diagram of the sites $m_1(M), \dots, m_N(M)$ restricted to $Conv(m_1(M), \dots, m_N(M))$ in the target configuration, as illustrated in Figure 15(b). For $m \in \{m_1, \dots, m_N\}$ and $t \in [0, M]$, let $V(m(t))$ denote the Voronoi region of $m(t)$ with respect to sites $\{m_1(t), \dots, m_N(t)\}$. Let p be the trajectory of an arbitrary robot not on the convex hull in the target configuration. Furthermore, let $p_1, \dots, p_6 \in \{m_1, \dots, m_N\}$ be the trajectories of the six robots $1, \dots, 6$ adjacent to p in the target configuration.

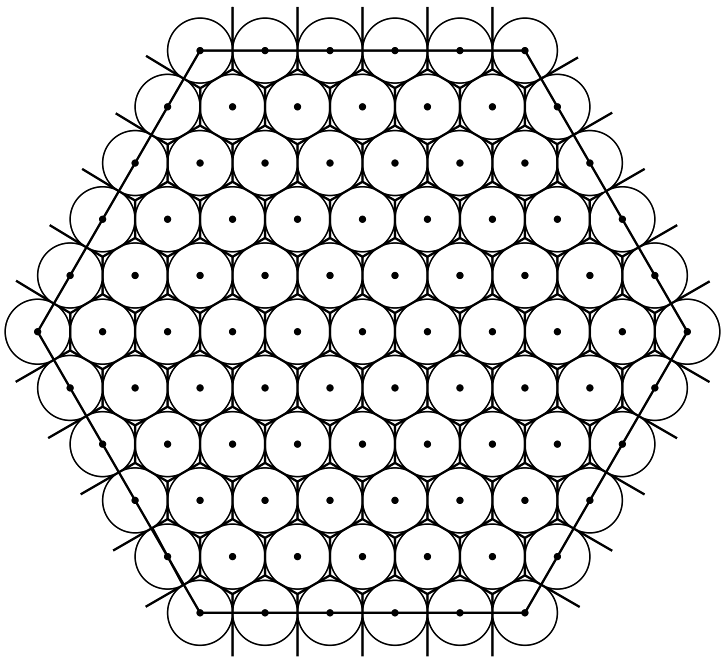
In the following, we show that there is a time interval $I = [t', t' + \frac{1}{20}]$ such that the area of $V(p(t''))$ is lower bounded by 3.479 for all $t'' \in I$; see Lemma 5.5. This is larger than the area of $V(p(0))$ and $V(p(M))$ by a constant factor. Based on that, we construct the time point $t \in [0, M]$ such that the area of $Conv(m_1(t), \dots, m_N(t))$ is lower bounded by $cN + \Omega(N^{3/4})$; see Lemma 5.6. To this end, we need to relate the area of a Voronoi region to the length of the corresponding Delaunay edges.

LEMMA 5.3. *Let $t' \in [0, M]$ and $p(t') \in \{m_1(t'), \dots, m_N(t')\}$. If the maximal distance between $p(t')$ and its Voronoi neighbors is $\lambda \in \left[2, 4\cos(50^\circ)\right)$, the area of $V(p(t'))$ is at least*

$$\frac{\lambda}{4}\left(3\sin\left(\arccos\left(\frac{\lambda}{4}\right)\right) - \frac{\lambda}{4}\tan\left(90^\circ - \arccos\left(\frac{\lambda}{4}\right)\right)\right) + \frac{4}{\sqrt{3}},$$



(a) Start and target positions of the robots.

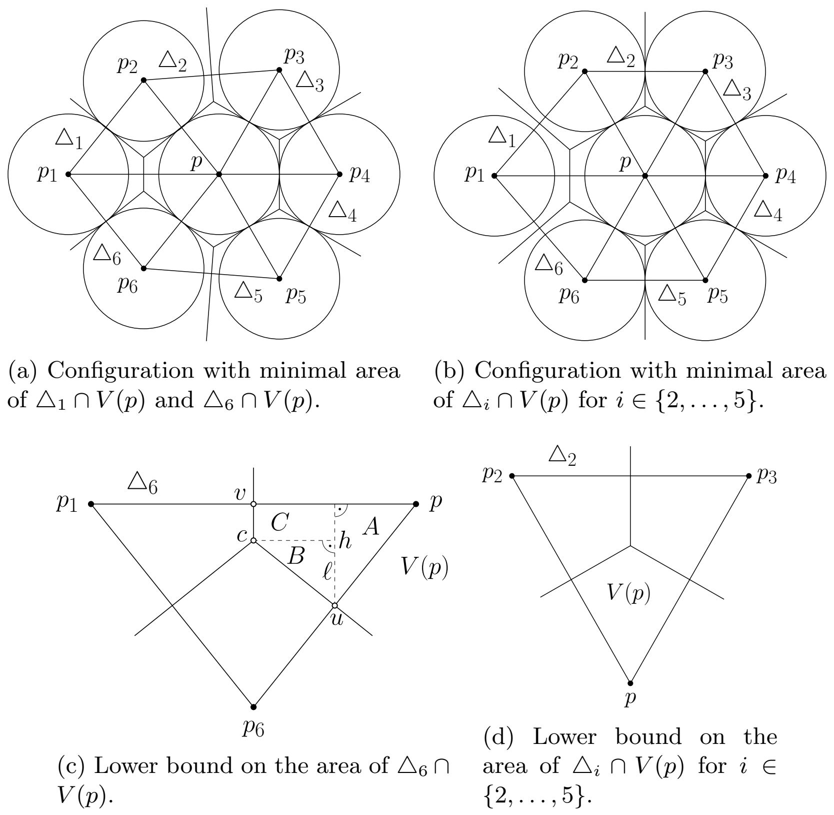


(b) Voronoi diagram in the start and target configurations.

FIG. 15. *The start and target configurations of our lower-bound construction where an arrow points from a start position to the corresponding target position.*

which is at least 3.479 *for* $\lambda \in [2.1, 2.2]$. *Furthermore, the area of* $V(p(M))$ *in the target configuration is* $\frac{6}{\sqrt{3}} = 2\sqrt{3} \leq 3.465$.

Proof. Let $p_1(t') \in \{m_1(t'), \ldots, m_N(t')\}$ be the center of a robot with $|p(t')p_1(t')| = \lambda$. Because all Voronoi neighbors of $p(t')$ have distance less than $4\cos(50^\circ)$, the angle between two Voronoi neighbors of $p(t')$ in $p(t')$ is greater than 50°. Thus, $p := p(t')$ has at most six Voronoi neighbors, and the area of $V(p(t'))$ is minimized if $p(t')$ has five additional Voronoi neighbors $p_2(t'), \ldots, p_6(t')$. We can assume $|p(t')p_2(t')| = \cdots = |p(t')p_6(t')| = 2$ because this does not increase the area of $V(p(t'))$. W.l.o.g., let $p_1 := p_1(t'), \ldots, p_6 := p_6(t')$ be in counterclockwise order around p. This situation is depicted in Figure 16.



(a) Configuration with minimal area of $\triangle_1 \cap V(p)$ and $\triangle_6 \cap V(p)$.

(b) Configuration with minimal area of $\triangle_i \cap V(p)$ for $i \in \{2, \ldots, 5\}$.

(c) Lower bound on the area of $\triangle_6 \cap V(p)$.

(d) Lower bound on the area of $\triangle_i \cap V(p)$ for $i \in \{2, \ldots, 5\}$.

FIG. 16. *Lower bounding the area of* $V(p)$ *by lower bounding the sum of the areas of the intersections of* $V(p)$ *with the Delaunay triangles* $\triangle_i$ *for* $i \in \{1, \ldots, 6\}$ *for a maximal distance of* 2.5 *between* p_1 *and* p.

We find a lower bound on $V(p)$ by lower bounding the intersections of $V(p)$ with the Delaunay triangles that are adjacent to p, i.e., with the triangles built by the edges $p_1p_2, \ldots, p_5p_6$ and p_6p_1 with p; see Figure 16. The area of the two triangles $\triangle_1$ and $\triangle_6$ built by p_1p_2 and p_6p_1 with p are minimized by assuming the configuration of Figures 16(a),(b), i.e., for $|p_1p_2| = |p_1p_6| = 2$.

In the configuration of Figures 16(a),(b), we lower bound the area of $\triangle_6 \cap V(p)$ as follows: we subdivide the area of $\triangle_6 \cap V(p)$ into three subsets A, B, and C, and lower bound the area of $\triangle_6 \cap V(p)$ by the sum of lower bounds for $|A|$, $|B|$, and $|C|$.

Let u, v, and c be the midpoints of pp_6, pp_1, and $\triangle_6$; see Figure 16(b). Furthermore, let h be the vertical side length of A, and let ℓ be the vertical side length of B. The interior angle of $\triangle_6$ at p is $\arccos(\frac{\lambda}{4})$. Thus, we obtain $h = \sin\left(\arccos\left(\frac{\lambda}{4}\right)\right)$, which implies $|A| = \frac{1}{2} \cdot \frac{\lambda}{4} \sin\left(\arccos\left(\frac{\lambda}{4}\right)\right)$. The interior angle of B at c is $90° - \arccos\left(\frac{\lambda}{4}\right)$. Hence, we get $\ell = \frac{\lambda}{4} \tan\left(90° - \arccos\left(\frac{\lambda}{4}\right)\right)$ because the length of B's horizontal side is $\frac{\lambda}{4}$. Therefore, $|B| = \frac{1}{2} \cdot \frac{\lambda}{4} \cdot \frac{\lambda}{4} \tan\left(90° - \arccos\left(\frac{\lambda}{4}\right)\right)$. Finally, we have $|C| = \frac{\lambda}{4}(h-\ell) = \frac{\lambda}{4}\left(\sin\left(\arccos\left(\frac{\lambda}{4}\right)\right) - \frac{\lambda}{4}\tan\left(90° - \arccos\left(\frac{\lambda}{4}\right)\right)\right)$. As $\triangle_6 \cap V(p)$ and $\triangle_1 \cap V(p)$ are symmetric, this gives us a lower bound of $2(|A| + |B| + |C|)$ on $|(\triangle_1 \cup \triangle_6) \cap V(p)|$.

Furthermore, the area of $(\triangle_2 \cup \cdots \cup \triangle_5) \cap V(p)$ is minimized by the configuration, implying the highest possible packing density, illustrated in Figures 16(c),(d) for $|p_2 p| = \cdots = |p_6 p| = |p_2 p_3| = \cdots = |p_5 p_6| = 2$. Therefore, this area is at least $4 \cdot \frac{1}{3} \cdot |\triangle_i| = 4 \cdot \frac{1}{3} \cdot \frac{1}{2} \cdot 2 \cdot \sqrt{3} = \frac{4}{\sqrt{3}}$. All in all, we obtain $|V(p)| \geq 2(|A|+|B|+|C|) + \frac{4}{\sqrt{3}}$. For $\lambda \in [2.1, 2.2]$, this is at least $1.17046 + \frac{4}{\sqrt{3}} \geq 3.479$. In the target configuration, we have $|p(M)p_i(M)| = 2$ for $i \in \{1, \ldots, 6\}$. Therefore the area of $V(p(M))$ is $\frac{6}{\sqrt{3}} = 2\sqrt{3} \leq 3.465$. ∎

Next, we prove that there is a time t' with an interval $I := [t', t' + \frac{1}{20}]$ during which the area of $Conv(p, p_1, \ldots, p_6)$ is greater by a constant factor than the area of $Conv(p, p_1, \ldots, p_6)$ in the target configuration. To this end, we use the following observation that is an immediate consequence of the intermediate value theorem.

OBSERVATION 5.4. *There is a time $t' \in [0, M]$ for which the maximal distance between $p(t')$ and $p_1(t'), \ldots, p_6(t')$ is* 2.2.

LEMMA 5.5. *There is a time $t' \in [0, M]$ such that for all $t'' \in [t', t' + \frac{1}{20}]$, the area of $V(p(t''))$ is at least $3.479 \geq 1.004 \cdot |V(p(M))|$.*

Proof. Let $\lambda(t)$ be the maximal distance between $p(t)$ and $p_1(t), \ldots, p_6(t)$. By Observation 5.4, there is a maximal time t' with $\lambda(t') = 2.2$. Therefore, and because $2.2 < 4\cos(50°) < 2\sqrt{2}$, the points $p_1(t'), \ldots, p_6(t')$ are the Voronoi neighbors of $p(t')$. By Observation 5.2, $\lambda(t)$ is 2-Lipschitz. This, together with the maximality of t', implies $2.1 \leq \lambda(t'') \leq 2.2$ for $t'' \in [t', t' + \frac{1}{20}]$. Thus, Lemma 5.3 applies and yields $|V(p(t''))| \geq 3.479 \geq 1.004 \cdot |V(p(M))|$ for all $t'' \in [t', t' + \frac{1}{20}]$. ∎

LEMMA 5.6. *There is a $t \in [0, M]$ for which the area of $Conv(m_1(t), \ldots, m_N(t))$ is lower-bounded by $3.479\left(\left\lfloor \frac{N}{20M} \right\rfloor - \sqrt{2}\pi\left(2\sqrt{N} + M\right)\right) + 2\sqrt{3}\left(N - \left(\left\lfloor \frac{N}{20M} \right\rfloor\right)\right)$.*

Proof. By Lemma 5.5, for each robot i that does not lie on the boundary of the start configuration, there is a point in time $t'' \in [0, M]$ such that the area of $V(p(t''))$ is at least 3.479 for all $t'' \in [t', t' + \frac{1}{20}]$. The continuous pigeonhole principle yields a time point $t \in [0, M]$ such that the area of $k := \left\lfloor \frac{N}{20M} \right\rfloor \in \Theta(\frac{N}{M})$ Voronoi regions $V(q_1(t)), \ldots, V(q_k(t))$ is at least 3.479. For all the remaining Voronoi regions, the area is at least $2\sqrt{3}$ corresponding to the largest possible packing density as achieved in the start and target configurations.

We give an upper bound $N \leq \sqrt{2}\pi(2\sqrt{N} + M)$ on the number of robots whose Voronoi regions are not contained in $Conv(m_1(t), \ldots, m_N(t))$. W.l.o.g., we assume that all these regions are Voronoi regions whose area we lower bounded by 3.479. Moreover, we can assume all of these regions have zero area, i.e., ignoring them when lower bounding the area of $Conv(m_1(t), \ldots, m_N(t))$. Thus, we obtain that the area of $Conv(m_1(t), \ldots, m_N(t))$ is at least

$$3.479(k - N) + 2\sqrt{3}(N - k) = 3.479\left(\left\lfloor \tfrac{N}{20M} \right\rfloor - \sqrt{2}\pi\left(2\sqrt{N} + M\right)\right) + 2\sqrt{3}\left(N - \left(\left\lfloor \tfrac{N}{20M} \right\rfloor\right)\right).$$

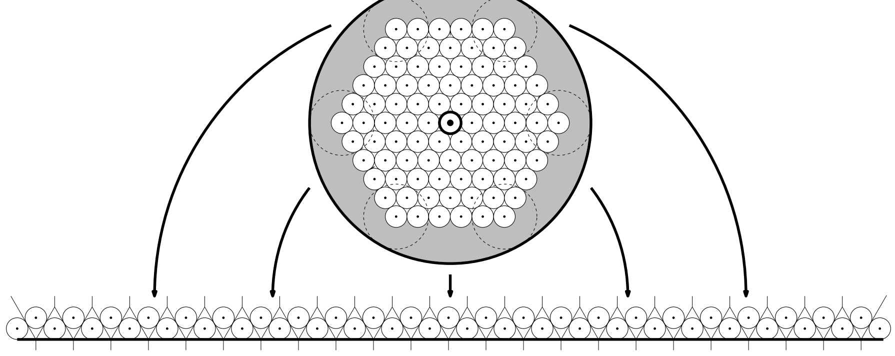


FIG. 17. *An upper-bound construction for the number of robots whose Voronoi regions may intersect the boundary of the smallest enclosing ball for $\{m_1(t), \dots, m_N(t)\}$. The radius of the smallest enclosing ball is upper-bounded by the distance from the center to the boundary in the start configuration plus the considered makespan illustrated by the dashed circles.*

It still remains to prove the upper bound $N \leq \sqrt{2}\pi(2\sqrt{N} + M)$ on the number of robots whose Voronoi regions are not contained in $Conv(m_1(t), \dots, m_N(t))$. First, we observe that the length of the boundary of $Conv(m_1(t), \dots, m_N(t))$ is at most $B := 2\pi(2\sqrt{N} + M)$, because $2\sqrt{N} + M$ is an upper bound on the radius of the smallest ball containing $m_1(t), \dots, m_N(t)$. In order to estimate N, we consider the maximal number of points from $[0, B] \times \mathbb{R}_{\geq 0}$ whose Voronoi regions intersect the x-axis. This number is achieved for the configuration as illustrated in Figure 17, implying $N \leq \frac{B}{\sqrt{2}} = \sqrt{2}\pi(2\sqrt{N} + M)$, thus concluding the proof. ∎

LEMMA 5.7. *For each time $t \in [0, M]$, the area of $Conv(m_1(t), \dots, m_N(t))$ is lower-bounded by $2\sqrt{3}N + 2\pi(\sqrt{N} + M)M$.*

Proof. In the start configuration, the intersection of each Voronoi cell with $Conv(m_1(0), \dots, m_N(0))$ has an area of $2\sqrt{3}$. Thus, the convex hull of the start configuration has an area of at most $2\sqrt{3}N$. We give an upper bound on the area A gained during the motion, i.e., the area of $Conv(m_1(t), \dots, m_N(t)) \setminus Conv(m_1(0), \dots, m_N(0))$, corresponding to the gray region in Figure 17. The length of the boundary $\partial Conv(m_1(t), \dots, m_N(t))$ is at most $2\pi(\sqrt{N} + M)$, implying $A \leq 2\pi(\sqrt{N} + M)M$, thus concluding the proof. ∎

THEOREM 5.8. *There is an instance with optimal makespan $M \in \Omega(N^{1/4})$; see Figure* 15.

Proof. Combining the bounds from Lemma 5.6 and Lemma 5.7 yields

$$\begin{aligned} &3.479\left(\left\lfloor \frac{N}{20M} \right\rfloor - \sqrt{2}\pi\left(2\sqrt{N} + M\right)\right) + 2\sqrt{3}\left(N - \left(\left\lfloor \frac{N}{20M} \right\rfloor\right)\right) \\ &\leq 2\sqrt{3}N + 2\pi(\sqrt{N} + M)M. \\ \Leftrightarrow \quad &0,014 \left\lfloor \frac{N}{20M} \right\rfloor - 3,479\sqrt{2}\pi\left(2\sqrt{N} + M\right) \leq 2\pi(\sqrt{N} + M)M. \end{aligned}$$

If $M \in \Omega(\sqrt{N})$ holds, we are done. Otherwise we obtain $\frac{N}{M} \in \mathcal{O}(M\sqrt{N}) \Leftrightarrow \sqrt{N} \in \mathcal{O}(M^2)$, and thus $M \in \Omega(N^{1/4})$, concluding the proof. ∎

It is a straightforward consequence of continuity that the above arguments still apply if disks do not touch in the start configuration, as long as the initial distance of adjacent disk centers is at most $2+\delta$ for a sufficiently small $\delta > 0$.

COROLLARY 5.9. *There is a constant $\delta > 0$, such that the optimal makespan can be $M \in \Omega(N^{1/4})$, even if no two disk centers are closer than $2+\delta$.*

5.2. An upper bound for unbounded environments. Next we give upper bounds on the stretch and makespan for moving disks in unbounded environments. First, we show that we can achieve constant stretch for well-separated robots.

THEOREM 5.10. *If the distance between the centers of two robots of radius 1 is at least 4 in the start and target configurations, we can achieve a makespan in $\mathcal{O}(d)$, i.e., constant stretch.*

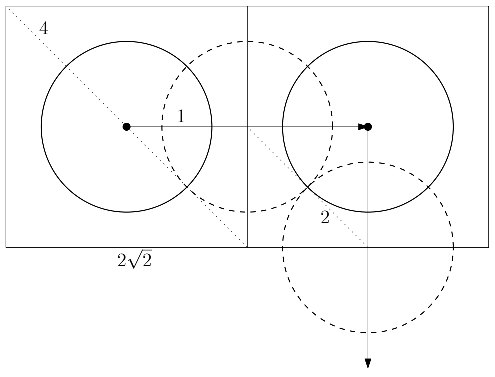


FIG. 18. *A mesh size of $2\sqrt{2}$ avoids robot collisions, and the cell diagonals have length 4. Note that robots may have arbitrary shape, as the separation argument applies to their circumcircles.*

Proof. We consider a grid D with mesh size $2\sqrt{2}$. In this way, as shown in Figure 18, two robots starting simultaneously from different cells and traveling along two incident edges can touch when they reach the midpoints, but do not collide. Moreover, the diagonals have length 4. By choosing a grid that has no robot center on a grid line, every cell of D contains at most one start and one target position of a robot. Additionally, we can move each robot in the start and target configuration to the center of its own cell, allowing us to to use our algorithm from section 3. Overall, we achieve a set of trajectories with makespan in $\mathcal{O}(d)$. ∎

In the remainder of this section we give an $\mathcal{O}(\sqrt{N})$-approximation algorithm for the continuous makespan for this kind of *well-separated* arrangements, by extending the approach for discrete grids.

Again, we make use of an underlying grid with mesh size $2\sqrt{2}$. Our algorithm proceeds in three phases: (1) moving the robots to vertices of the grid, (2) applying our $\mathcal{O}(1)$-approximation for the discrete case, and (3) moving the robots from the vertices of the grid to their target positions. To ensure a $\mathcal{O}(\sqrt{N})$-approximation, we move each robot center to a grid vertex within a distance of $\mathcal{O}(\sqrt{N})$. Phases (1) and (3) are symmetric in the following sense. By applying the steps of phase (1) to the target configuration in reverse, we compute a grid configuration that serves as target configuration for phase (2).

Phase (1) works as follows. (1.1) We begin by sorting the N robots according to the (x, y)-lexicographical order. Then we subdivide them into $\lceil\sqrt{N}\rceil$ vertical slices, each containing at most $\lceil\sqrt{N}\rceil$ robots. To the right of every slice, we add a vertical buffer slice of width $4\sqrt{2}$ by moving all robots not yet considered by $4\sqrt{2}$ units to the right. These trajectories are used in parallel; the distance covered by each robot is in $\mathcal{O}(\sqrt{N})$. The buffer slices guarantee that in all following steps, the robots in each vertical slice are independent of each other.

(1.2) We continue by sorting the robots within the vertical slices according to the (y, x)-lexicographical order. We separate the robots by ensuring vertical distance at least $4\sqrt{2}$ between every pair of robots. This can be done by moving the robots upwards, starting from the second-to-lowest one. These trajectories can be done in parallel, and the distance covered by each robot is in $\mathcal{O}(\sqrt{N})$. (1.3) We finally move each robot to the bottom-left vertex of the grid cell containing its center.

THEOREM 5.11. *There is an algorithm that computes a trajectory set with continuous makespan of $\mathcal{O}(d + \sqrt{N})$. If $d \in \Omega(1)$, this implies a $\mathcal{O}(\sqrt{N})$-approximation algorithm.*

Proof. Phase (1) guarantees that either the horizontal or the vertical distance between each pair of robots is at least $4\sqrt{2}$. Therefore, in each grid cell, there is at most one robot and each robot is moved to its own grid vertex. In phase (2), each robot is moved by $\mathcal{O}(d')$ units, where d' is the maximal distance between a robot's start and target position in the grid. As the distance each robot covers in phases (1) and (3) is in $\mathcal{O}(\sqrt{N})$, the distance traveled in phase (2) is in $\mathcal{O}(d + \sqrt{N})$. Therefore, the trajectory set computed by the algorithm has continuous makespan in $\mathcal{O}(d + \sqrt{N})$. The runtime as described above is pseudopolynomial; it becomes polynomial by using standard compression techniques, e.g., by compressing large empty rectangles. ∎

5.3. Colored and unlabeled disks. We can combine the positive results of the previous section with the technique of Theorem 5.10 to achieve the same result for colored (and in particular, unlabeled) disks.

COROLLARY 5.12. *There is an algorithm with runtime $\mathcal{O}(k(mn)^{1.5}\log(mn) + dmn)$ that computes, given start and target images I_s, I_t, an $\mathcal{O}(1)$-approximation of the optimal makespan M and a corresponding set of trajectories.*

Proof. The proof proceeds analogously to Theorem 4.1: after computing an optimal bottleneck matching, apply Theorem 3.4 in the setting of Theorem 5.10. ∎

5.4. Difficulties with exact solutions. For small instances, we might hope to obtain provably optimal trajectories, as opposed to solutions that are within a provable constant factor of the optimum. Such scenarios might arise both in practice and as gadgets (building blocks) for NP-hardness proofs.

In this section, we briefly consider one simple such scenario: moving one disk to the opposite side of another disk that wants to remain where it is. Refer to Figure 19. This is closely related to recent work by Kirkpatrick and Liu [44], who devote a whole paper to computing optimal trajectories for two disks in arbitrary initial and target configurations, with the objective of minimizing the total distance traveled instead of the makespan. A key insight is that optimal trajectories consist of a limited number of circular arcs. This is not necessarily the case for trajectories that minimize the makespan. Even for the seeming simplicity of our example, we do not have a proof of optimality of the trajectory T_3 shown on the right. This illustrates the difficulty of characterizing and establishing optimal trajectories that minimize the total duration of a parallel schedule, highlighting the special role of geometry for the problem.

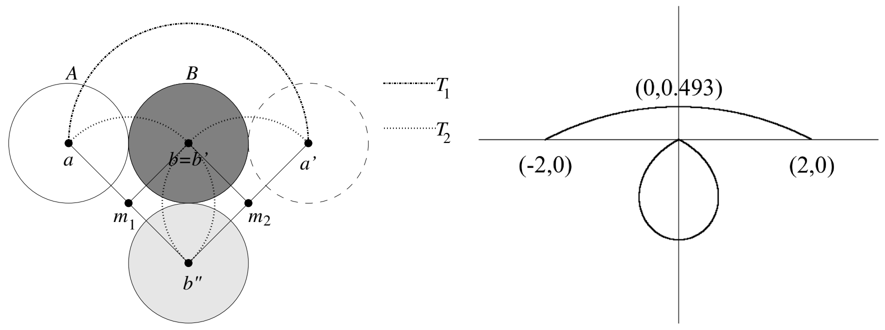


FIG. 19. *Moving the left unit disk A from position a to position a' at distance 4, with disk B starting and ending at $b = b'$. See our video* [6] *for animations of the following two solutions.* Left: *Trajectory T_1 rotates disk A around the stationary disk B resulting in makespan $2\pi = 6.28\ldots$. Trajectories T_2 rotate both disks around the centers m_1 and m_2, resulting in makespan $\sqrt{2}\pi = 4.44\ldots$.* Right: *Choosing a circular arc through $(-2, 0), (2, 0)$ and the (numerically optimized) point $(0, 0.493\ldots)$ for disk A (with B moving accordingly at distance 2) yields the trajectory T_3 with makespan $4.16\ldots$.*

6. Conclusion. We have presented progress on several algorithmic problems in parallel motion planning. We hope that this research will open up a new field of approximation algorithms for optimal motion plans to achieve desired goals. In particular, the following open problems are natural next steps.

The first set of problems consider complexity. We showed that the labeled problem of section 3 is NP-complete in the grid. It is natural to conjecture that the geometric version is also hard. It seems tougher to characterize the family of optimal trajectories (as explored in section 5.4), so even membership in NP seems challenging.

A second set of questions considers the relationship between stretch factor and disk separability in the continuous setting. We believe that the upper bound of $\mathcal{O}(\sqrt{N})$ on the worst-case stretch factor for dense arrangements is tight. What is the critical separability σ^* of disks for which constant stretch can be achieved? As we have shown, $\sigma^* \in [2 + \delta, 2.8284\ldots]$ when using L_∞-distances. How does the stretch factor increase as a function of N below this threshold σ^*? For *sparse* arrangements of disks, simple greedy straight-line trajectories between the origins and destinations of disks encounter only isolated conflicts, resulting in small stretch factors close to 1, namely, $1 + o(1)$. What is the relationship between (local) density and the achievable stretch factor along the whole density spectrum?

Finally, practical motion planning requires a better handle on characterizing and computing optimal solutions for specific instances, along with lower bounds, possibly based on numerical methods and tools. Moreover, there is a wide range of additional objectives and requirements, such as accounting for acceleration or deceleration of disks, turn cost, or multistop tour planning. All of these are left for future work.

Acknowledgment. We thank the anonymous reviewers of a preliminary version of the paper for helping to improve the overall presentation.

REFERENCES

[1] M. Abellanas, S. Bereg, F. Hurtado, A. G. Olaverri, D. Rappaport, and J. Tejel, *Moving coins*, Comput. Geom., 34 (2006), pp. 35–48.
[2] A. Adler, M. de Berg, D. Halperin, and K. Solovey, *Efficient multi-robot motion plan-*

ning for unlabeled discs in simple polygons, IEEE Trans. Automat. Sci. Eng., 12 (2015), pp. 1309–1317.

[3] K. M. Al-Wahedi, *A Hybrid Local-global Motion Planner for Multi-agent Coordination*, master's thesis, Case Western Reserve University, 2000.

[4] B. Aronov, M. de Berg, A. F. van der Stappen, P. Švestka, and J. Vleugels, *Motion planning for multiple robots*, Discrete Comput. Geom., 22 (1999), pp. 505–525.

[5] M. Barer, G. Sharon, R. Stern, and A. Felner, *Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem*, in Proceedings of the Seventh Annual Symposium on Combinatorial Search (SoCS), 2014, pp. 19–27.

[6] A. T. Becker, S. P. Fekete, P. Keldenich, L. Lin, and C. Scheffer, *Coordinated motion planning: The video*, in Proceedings of the 34th International Symposium on Computational Geometry (SoCG), 2018, pp. 74:1–74:6; video available via http://www.computational-geometry.org/SoCG-videos/socg18video/.

[7] S. Bereg, A. Dumitrescu, and J. Pach, *Sliding disks in the plane*, Internat. J. Comput. Geom. Appl., 18 (2008), pp. 373–387.

[8] P. Berman, E. D. Demaine, and M. Zadimoghaddam, *$O(1)$-approximations for maximum movement problems*, in Proceedings of the 14th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX), 2011, pp. 62–74.

[9] G. Călinescu, A. Dumitrescu, and J. Pach, *Reconfigurations in graphs and grids*, SIAM J. Discrete Math., 22 (2008), pp. 124–138, https://doi.org/10.1137/060652063.

[10] C. E. Campbell and J. Y. S. Luh, *A Preliminary Study on Path Planning of Collision Avoidance for Mechanical Manipulators*, Tech. report, School of Electrical Engineering, Purdue University, 1980.

[11] Y. F. Chen, M. Liu, M. Everett, and J. P. How, *Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning*, in Proceedings of the 32nd IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 285–292.

[12] S. Cheung and F. C. M. Lau, *Mesh permutation routing with locality*, Inform. Process. Lett., 43 (1992), pp. 101–105.

[13] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, *A survey on aerial swarm robotics*, IEEE Trans. Robotics, 34 (2018), pp. 837–855.

[14] R. Cui, B. Gao, and J. Guo, *Pareto-optimal coordination of multiple robots with safety guarantees*, Auton. Robots, 32 (2012), pp. 189–205.

[15] D. Delahaye, S. Puechmorel, P. Tsiotras, and E. Féron, *Mathematical models for aircraft trajectory design: A survey*, in Air Traffic Management and Systems, Springer, 2014, pp. 205–247.

[16] E. D. Demaine, M. L. Demaine, and H. Verrill, *Coin-moving puzzles*, in More Games of No Chance, Math. Sci. Res. Inst. Publ. 42, Cambridge University Press, 2002, pp. 405–431.

[17] E. D. Demaine, S. P. Fekete, P. Keldenich, H. Meijer, and C. Scheffer, *Coordinated motion planning: Reconfiguring a swarm of labeled robots with bounded stretch*, in Proceedings of the 34th International Symposium on Computational Geometry (SoCG), 2018, 29.

[18] E. D. Demaine, M. T. Hajiaghayi, H. Mahini, A. S. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam, *Minimizing movement*, ACM Trans. Algorithms, 5 (2009), 30.

[19] E. D. Demaine, M. T. Hajiaghayi, and D. Marx, *Minimizing movement: Fixed-parameter tractability*, ACM Trans. Algorithms, 11 (2014), 14.

[20] V. R. Desaraju and J. P. How, *Decentralized path planning for multi-agent teams with complex constraints*, Auton. Robots, 32 (2012), pp. 385–403.

[21] J. M. Díaz-Báñez, M. A. Heredia, C. Peláez, J. A. Sellarès, J. Urrutia, and I. Ventura, *Convex blocking and partial orders on the plane*, Comput. Geom., 51 (2016), pp. 55–66.

[22] A. Dumitrescu, *Motion planning and reconfiguration for systems of multiple objects*, in Mobile Robots: Perception & Navigation, S. Kolski, ed., InTech, 2007, pp. 1–20.

[23] A. Dumitrescu and M. Jiang, *On reconfiguration of disks in the plane and related problems*, Comput. Geom., 46 (2013), pp. 191–202.

[24] A. Efrat, A. Itai, and M. J. Katz, *Geometry helps in bottleneck matching and related problems*, Algorithmica, 31 (2001), pp. 1–28.

[25] M. Erdmann and T. Lozano-Pérez, *On multiple moving objects*, Algorithmica, 2 (1987), pp. 477–521.

[26] S. P. Fekete, B. Hendriks, C. Tessars, A. Wegener, H. Hellbrück, S. Fischer, and S. Ebers, *Methods for improving the flow of traffic*, in Organic Computing—A Paradigm Shift for Complex Systems, C. Müller-Schloer, H. Schmeck, and T. Ungerer, eds., Autonomic Systems 1, Springer, Basel, 2011, pp. 447–460.

[27] A. Felner, M. Goldenberg, G. Sharon, R. Stern, T. Beja, N. R. Sturtevant, J. Schaef-

fer, and R. Holte, *Partial-expansion A* with selective node generation*, in Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI-12), 2012, pp. 471–477.

[28] G. W. Flake and E. B. Baum, *Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants*," Theoret. Comput. Sci., 270 (2002), pp. 895–911.

[29] P. Flocchini, G. Prencipe, and N. Santoro, *Distributed Computing by Mobile Entities*, Lecture Notes in Comput. Sci. 11340, Springer, 2019.

[30] E. Freund and H. Hoyer, *On the on-line solution of the findpath problem in multi-robot systems*, in Proceedings of the 3rd International Symposium on Robotics Research (ISRR), O. Faugeras and G. Giralt, eds., 1985, pp. 253–262.

[31] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1990.

[32] A. Geramifard, P. Chubak, and V. Bulitko, *Biased cost pathfinding*, in Proceedings of the 2nd AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-06), 2006, pp. 112–114.

[33] M. Goldenberg, A. Felner, R. Stern, G. Sharon, N. R. Sturtevant, R. C. Holte, and J. Schaeffer, *Enhanced partial expansion A**, J. Artificial Intelligence Res., 50 (2014), pp. 141–187.

[34] M. T. Hajiaghayi, R. Khandekar, M. R. Khani, and G. Kortsarz, *Approximation algorithms for movement repairmen*, ACM Trans. Algorithms, 12 (2016), 54.

[35] P. E. Hart, N. J. Nilsson, and B. Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Trans. Syst. Sci. Cybern., 4 (1968), pp. 100–107.

[36] R. A. Hearn and E. D. Demaine, *PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation*, Theoret. Comput. Sci., 343 (2005), pp. 72–96.

[37] R. A. Hearn and E. D. Demaine, *Games, puzzles, and computation*, A K Peters, Wellesley, MA, 2009.

[38] S. Hirsch and D. Halperin, *Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane*, in Algorithmic Foundations of Robotics V, Springer, 2004, pp. 239–255.

[39] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, *On the complexity of motion planning for multiple independent objects; PSPACE-hardness of the "warehouseman's problem*," Internat. J. Robotics Res., 3 (1984), pp. 76–88.

[40] J. E. Hopcroft and G. T. Wilfong, *Reducing multiple object motion planning to graph searching*, SIAM J. Comput., 15 (1986), pp. 768–785, https://doi.org/10.1137/0215055.

[41] R. Huang, Y. Chen, and W. Zhang, *A novel transition based encoding scheme for planning as satisfiability*, in Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10), 2010, pp. 89–94.

[42] K. Kant and S. W. Zucker, *Toward efficient trajectory planning: The path-velocity decomposition*, Internat. J. Robotics Res., 5 (1986), pp. 72–89.

[43] H. Kautz and B. Selman, *Unifying SAT-based and graph-based planning*, in Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), Vol. 99, 1999, pp. 318–325.

[44] D. Kirkpatrick and P. Liu, *Characterizing minimum-length coordinated motions for two discs*, in Proceedings of the 28th Canadian Conference on Computational Geometry (CCCG), 2016, pp. 252–259.

[45] S. Kloder and S. Hutchinson, *Path planning for permutation-invariant multi-robot formations*, IEEE Trans. Robot. Autom., 22 (2006), pp. 650–665.

[46] D. Kornhauser, G. Miller, and P. Spirakis, *Coordinating pebble motion on graphs, the diameter of permutation groups, and applications*, in Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS), 1984, pp. 241–250.

[47] M. Kunde, *Routing and sorting on mesh-connected arrays*, in Proceedings of the 3rd Aegean Workshop on Computation (AWOC), Springer, 1988, pp. 423–433.

[48] S. LaValle and S. A. Hutchinson, *Optimal motion planning for multiple robots having independent goals*, IEEE Trans. Robot. Autom., 14 (1998), pp. 912–925.

[49] S. Leroy, J.-P. Laumond, and T. Siméon, *Multiple path coordination for mobile robots: A geometric algorithm*, in Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), 1999, pp. 1118–1123.

[50] J. M. Marberg and E. Gafni, *Sorting in constant number of row and column phases on a mesh*, Algorithmica, 3 (1988), pp. 561–572.

[51] J. Peng and S. Akella, *Coordinating multiple robots with kinodynamic constraints along specified paths*, Internat. J. Robotics Res., 24 (2005), pp. 295–310.

[52] G. Ramanathan and V. Alagar, *Algorithmic motion planning in robotics: Coordinated mo-*

tion of several disks amidst polygonal obstacles, in Proceedings of the Second IEEE International Conference on Robotics and Automation (ICRA), Vol. 2, 1985, pp. 514–522.

[53] D. Ratner and M. K. Warmuth, *Finding a shortest solution for the $N \times N$ extension of the* 15*-puzzle is intractable*, in Proceedings of the Fifth AAAI Conference on Artificial Intelligence (AAAI-86), 1986, pp. 168–172.

[54] R. Regele and P. Levi, *Cooperative multi-robot path planning by heuristic priority adjustment*, in Proceedings of the 19th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2006, pp. 5954–5959.

[55] M. Rubenstein, A. Cornejo, and R. Nagpal, *Programmable self-assembly in a thousand-robot swarm*, Science, 345 (2014), pp. 795–799.

[56] E. Şahin and A. Winfield, eds., *Special issue on swarm robotics*, Swarm Intell., 2 (2008).

[57] O. Salzman, M. Hemmer, and D. Halperin, *On the power of manifold samples in exploring configuration spaces and the dimensionality of narrow passages*, IEEE Trans. Automation Sci. Eng., 12 (2015), pp. 529–538.

[58] G. Sanchez and J.-C. Latombe, *Using a PRM planner to compare centralized and decoupled planning for multi-robot systems*, in Proceedings of the 19th IEEE International Conference on Robotics and Automation (ICRA), 2002, pp. 2112–2119.

[59] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, *Primal: Pathfinding via reinforcement and imitation multi-agent learning*, IEEE Robot. Automat. Lett., 4 (2019), pp. 2378–2385.

[60] C. Scheideler, *Universal Routing Strategies for Interconnection Networks*, Lecture Notes in Comput. Sci. 1390, Springer, 1998.

[61] M. Schreckenberg and R. Selten, eds., *Human Behaviour and Traffic Networks*, Springer, 2004.

[62] J. T. Schwartz and M. Sharir, *On the piano movers' problem:* III*. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers*, Internat. J. Robotics Res., 2 (1983), pp. 46–75.

[63] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, *Meta-agent conflict-based search for optimal multi-agent path finding*, in Proceedings of the Symposium on Combinatorial Search (SoCS), 2012, pp. 97–104.

[64] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, *Conflict-based search for optimal multi-agent pathfinding*, Artificial Intelligence, 219 (2015), pp. 40–66.

[65] G. Sharon, R. Stern, M. Goldenberg, and A. Felner, *The increasing cost tree search for optimal multi-agent pathfinding*, Artificial Intelligence, 195 (2013), pp. 470–495.

[66] T. Siméon, S. Leroy, and J.-P. Laumond, *Path coordination for multiple mobile robots: A resolution-complete algorithm*, IEEE Trans. Robot. Autom., 18 (2002), pp. 42–49.

[67] K. Solovey and D. Halperin, *k-color multi-robot motion planning*, Internat. J. Robotics Res., 33 (2014), pp. 82–97.

[68] K. Solovey and D. Halperin, *On the hardness of unlabeled multi-robot motion planning*, Internat. J. Robotics Res., 35 (2016), pp. 1750–1759.

[69] K. Solovey and D. Halperin, *Sampling-based bottleneck pathfinding with applications to Fréchet matching*, in Proceedings of the 24th Annual European Symposium on Algorithms (ESA), 2016, pp. 76:1–76:16.

[70] K. Solovey, O. Salzman, and D. Halperin, *Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning*, Internat. J. Robotics Res., 35 (2016), pp. 501–513.

[71] K. Solovey, J. Yu, O. Zamir, and D. Halperin, *Motion planning for unlabeled discs with optimality guarantees*, in Proceedings of the 11th Conference Robotics: Science and Systems (RSS), 2015.

[72] P. Spirakis and C. K. Yap, *Strong NP-hardness of moving many discs*, Inform. Process. Lett., 19 (1984), pp. 55–59.

[73] T. Standley, *Finding optimal solutions to cooperative pathfinding problems*, in Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10), 2010, pp. 173–178.

[74] P. Švestka and M. H. Overmars, *Coordinated path planning for multiple robots*, Robotics Autonomous Systems, 23 (1998), pp. 125–152.

[75] M. Turpin, N. Michael, and V. Kumar, *Trajectory planning and assignment in multirobot systems*, in Algorithmic Foundations of Robotics X, Springer, 2013, pp. 175–190.

[76] M. Turpin, K. Mohta, N. Michael, and V. Kumar, *Goal assignment and trajectory planning for large teams of interchangeable robots*, Autonomous Robots, 37 (2014), pp. 401–415.

[77] J. P. van den Berg and M. H. Overmars, *Prioritized motion planning for multiple robots*, in Proceedings of the 18th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2005, pp. 430–435.

[78] J. P. van den Berg, J. Snoeyink, M. C. Lin, and D. Manocha, *Centralized path planning for multiple robots: Optimal decoupling into sequential plans*, in Proceedings of Robotics: Science and Systems (RSS), Vol. 2, 2009, pp. 2–3.

[79] M. Čáp, P. Novák, M. Seleckỳ, J. Faigl, and J. Vokřìnek, *Asynchronous decentralized prioritized planning for coordination in multi-robot system*, in Proceedings of the 26th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 3822–3829.

[80] G. Wagner and H. Choset, *M*: A complete multirobot path planning algorithm with performance bounds*, in Proceedings of the 24th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011, pp. 3260–3267.

[81] G. Wagner and H. Choset, *Subdimensional expansion for multirobot path planning*, Artificial Intelligence, 219 (2015), pp. 1–24.

[82] R. M. Wilson, *Graph puzzles, homotopy, and the alternating group*, J. Combin. Theory Ser. B, 16 (1974), pp. 86–96.

[83] P. R. Wurman, R. D'Andrea, and M. Mountz, *Coordinating hundreds of cooperative, autonomous vehicles in warehouses*, AI Magazine, 29 (2008), pp. 9–20.

[84] J. Yu, *Constant Factor Optimal Multi-Robot Path Planning in Well-Connected Environments*, preprint, https://arxiv.org/abs/1706.07255v1, 2017.

[85] J. Yu, *Constant Factor Time Optimal Multi-Robot Routing on High-Dimensional Grids in Mostly Sub-Quadratic Time*, preprint, https://arxiv.org/abs/1801.10465, 2018.

[86] J. Yu and S. M. LaValle, *Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics*, IEEE Trans. Robotics, 32 (2016), pp. 1163–1177.

[87] J. Yu and D. Rus, *An effective algorithmic framework for near optimal multi-robot path planning*, in Proceedings of the 27th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 495–511.