



Connected Coordinated Motion Planning with Bounded Stretch

Sándor P. Fekete  

Department of Computer Science, TU Braunschweig, Germany

Phillip Keldenich  

Department of Computer Science, TU Braunschweig, Germany

Ramin Kosfeld  

Department of Computer Science, TU Braunschweig, Germany

Christian Rieck  

Department of Computer Science, TU Braunschweig, Germany

Christian Scheffer  

Faculty of Electrical Engineering and Computer Science,
Bochum University of Applied Sciences, Bochum, Germany

Abstract

We consider the problem of coordinated motion planning for a swarm of simple, identical robots: From a given start grid configuration of robots, we need to reach a desired target configuration via a sequence of parallel, continuous, collision-free robot motions, such that the set of robots induces a connected grid graph at all integer times. The objective is to minimize the *makespan* of the motion schedule, i.e., to reach the new configuration in a minimum amount of time. We show that this problem is NP-hard, even for deciding whether a makespan of 2 can be achieved, while it is possible to check in polynomial time whether a makespan of 1 can be achieved.

On the algorithmic side, we establish simultaneous constant-factor approximation for two fundamental parameters, by achieving *constant stretch* for *constant scale*. Scaled shapes (which arise by increasing all dimensions of a given object by the same multiplicative factor) have been considered in previous seminal work on self-assembly, often with unbounded or logarithmic scale factors; we provide methods for a generalized scale factor, bounded by a constant. Moreover, our algorithm achieves a *constant stretch factor*: If mapping the start configuration to the target configuration requires a maximum Manhattan distance of d , then the total duration of our overall schedule is $\mathcal{O}(d)$, which is optimal up to constant factors.

2012 ACM Subject Classification Theory of computation → Computational geometry; Computing methodologies → Motion path planning

Keywords and phrases Motion planning, parallel motion, bounded stretch, scaled shape, makespan, connectivity, swarm robotics

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.9

Related Version *Full Version*: <https://arxiv.org/abs/2109.12381> [11]

Acknowledgements We thank Linda Kleist and Arne Schmidt for helpful algorithmic discourse and suggestions that improved the presentation of this paper.

1 Introduction

Coordinating the motion of a set of objects is a fundamental problem that occurs in a large spectrum of theoretical and practical contexts. This problem was also the subject of the 2021 CG Challenge [12], highlighting the high relevance for the algorithmic community.



© Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer; licensed under Creative Commons License CC-BY 4.0

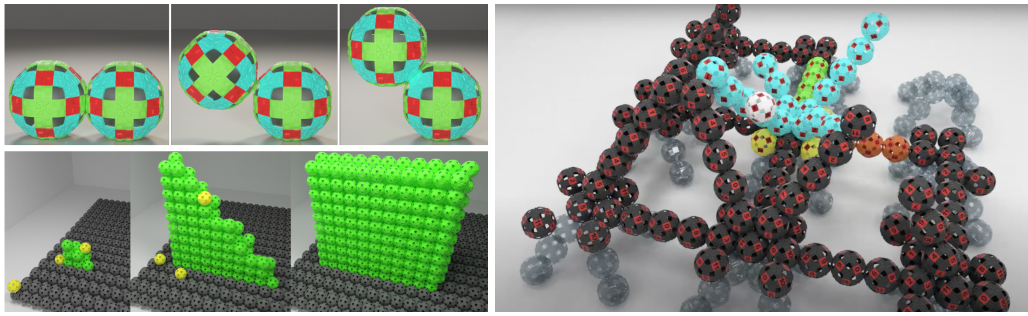
32nd International Symposium on Algorithms and Computation (ISAAC 2021).

Editors: Hee-Kap Ahn and Kunihiko Sadakane; Article No. 9; pp. 9:1–9:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** (Top left) An autonomous, sphere-shaped catom, changing location by rotating around a second catom used as a pivot [17]. (Bottom left) A swarm of catoms building a wall [17]. (Right) A configuration of catoms in the process of building a scaffold structure [26].

In this paper, we consider *connected* swarm reconfiguration: transform a set of mobile agents from a given start into a desired target configuration by a sequence of parallel, continuous, collision-free motions that keeps the overall arrangement connected at all integer times. Problems of this type occur for assemblies in space, where disconnected pieces cannot regain connectivity, or for small-scale swarm robots (such as *catoms* in *claytronics* [13]) which need connectivity for local motion, electric power and communication; see Figure 1.

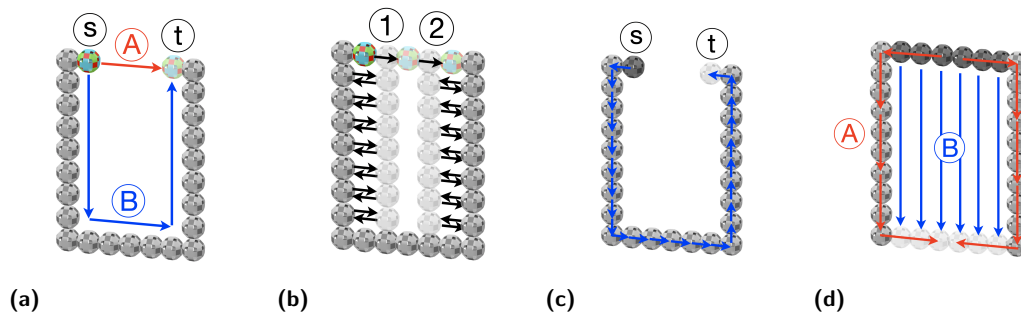
A crucial algorithmic aspect is *efficiency*: How can we coordinate the robot motions, such that a target configuration is reached in timely or energy-efficient manner? Most previous work has largely focused on sequential schedules, where one robot moves at a time, with objectives such as minimizing 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? As illustrated in Figure 2, this is where the connectivity constraints make a tremendous difference.

A critical parameter in self-assembly is the robustness of the involved shapes, corresponding to sufficient local connectivity to prevent fragility. This leads to the concept of *scaled shapes*; intuitively, a scale factor of c corresponds to replacing each pixel of a polyomino shape by a quadratic $c \times c$ array of pixels. This has fundamental connections to Kolmogorov and runtime complexity, as shown by Soloveichik and Winfree [22]: “Furthermore, the independence of scale in self-assembly theory appears to play the same crucial role as the independence of running time in the theory of computability. . . [we] show that the running-time complexity, with respect to Turing machines, is polynomially equivalent to the scale complexity of the same function implemented via self-assembly by a finite set of tile types.” As a consequence, limiting scale has received considerable attention, as sketched in the related work section.

As we demonstrate in this paper, achieving optimal makespan for connected reconfiguration is provably hard, even in relatively basic cases. On the positive side, we present methods that are capable of achieving a constant-factor approximation, assuming not more than a generalization of constant scale of configurations. As can be seen from Figure 2, this is considerably more intricate than in a non-connected setting, even in very basic instances.

1.1 Our Results

We provide a spectrum of new results for questions arising from efficiently reconfiguring a connected, unlabeled swarm of robots from a start configuration C_s into a target configuration C_t , aiming for minimizing the overall makespan and maintaining connectivity in each step.



■ **Figure 2** Reconfiguration with and without connectivity constraints. (a) Relocating the colored particle from s to t , without (red trajectory A) and with connectivity constraint (blue trajectory B). (b) Coordinating many particles to quickly deliver a specific particle to a desired location, while preserving connectivity. (c) Reconfiguring an arrangement of *identical* particles in a single, parallel, connected step. (d) Reconfiguring an arch-shaped arrangement of identical particles into a U-shaped one, without (motion plan A, shown in red) and with connectivity (motion plan B, shown in blue).

- Deciding whether there is a schedule with a makespan of 1 transforming C_s into C_t can be done in polynomial time, see Theorem 1.
- Deciding whether there is a schedule with a makespan of 2 transforming C_s into C_t is NP-hard, see Theorem 2. This implies NP-hardness of approximating the minimum makespan within a constant of $(\frac{3}{2} - \varepsilon)$, for any $\varepsilon > 0$, see Corollary 3.
- As our main algorithmic result, we show that there is a constant c^* such that for any pair of start and target configurations with a (generalized) scale of at least c^* , a schedule with constant stretch can be computed in polynomial time, see Theorem 4 and Corollary 11. This implies that there is a constant-factor approximation for the problem of computing schedules with minimal makespan restricted to pairs of start and target configurations with a scale of at least c^* , see Corollary 12.

1.2 Related Work

In the following, we provide a sketch of the wide spectrum of related work; see the full version of our paper [11] for a more detailed overview, as well as [7].

The basic question of coordinating the motion of many agents in an efficient manner arises in many applications, such as ground swarm robotics [18, 19], aerial swarm robotics [3, 28], air traffic control [5], and vehicular traffic networks [10, 20]. Multi-robot coordination dates back to the seminal work by Schwartz and Sharir [21] from the 1980s. 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. 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; see Solovey and Halperin [23]. In the *unlabeled* case, objects are indistinguishable and target positions can be covered by any object; see Kloder and Hutchinson [15], Turpin et al. [27], Adler et al. [1], and Solovey et al. [25]. On the negative side, Solovey and Halperin [24] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard.

For an instance of parallel reconfiguration, a lower bound for the time required for *all* robots to reach their destinations is the maximum distance between a robot's origin and destination. This motivates the *stretch factor*, i.e., the ratio of the makespan of a parallel motion plan divided by the maximum distance. In recent work, Demaine et al. [2, 7] were

able to develop algorithms that can achieve *constant* stretch factors that are independent of the number of robots; however, these approaches do not satisfy the crucial connectivity constraint, so different algorithmic methods are required.

The concept of scale complexity has received considerable attention in self-assembly; achieving constant scale has required special cases or operations. Soloveichik and Winfree [22] showed that the minimal number of distinct tile types necessary to self-assemble a shape, at some scale, can be bounded both above and below in terms of the shape’s Kolmogorov complexity, leading to unbounded scale in general. Demaine et al. [9] showed that allowing to destroy tiles can be exploited to achieve a scale that is only bounded by a logarithmic factor, beating the linear bound without such operations. In a setting of recursive, multi-level *staged* assembly with a logarithmic number of stages (i.e., “hands” for handling subassemblies), Demaine et al. [6] achieved logarithmic scale, and constant scale for more constrained classes of polyomino shapes; this was later improved by Demaine et al. [8] to constant scale for a logarithmic number of stages. More recently, Luchsinger et al. [16] employed repulsive forces between tiles to achieve constant scale in two-handed self-assembly.

2 Preliminaries

We consider *robots* at integer grid positions. A set of n unlabeled robots forms a *configuration* C , corresponding to a vertex-induced subgraph H of the infinite integer grid, with an edge between two grid vertices $v_1, v_2 \in C$ if and only if v_1 and v_2 are on adjacent grid positions, i.e., a distance of 1 apart. A configuration is *connected*, if H is connected. Two configurations C_1 and C_2 *overlap*, if they have at least one position in common. A configuration C is *c-scaled*, if it is the union of $c \times c$ squares of vertices. The *scale* of a configuration C is the maximal c such that C is c -scaled. This corresponds to objects being composed of pixels at a certain resolution; note that this is a generalization of the uniform pixel scaling studied in previous literature (which considers a c -grid-based partition instead of an arbitrary union), so it supersedes that definition and leads to a more general set of results. Two robots are *adjacent* if their positions v_1, v_2 are adjacent, i.e., $(v_1, v_2) \in E(H)$; and *diagonally adjacent* if their positions are adjacent with a common vertex v such that (v_1, v) and (v, v_2) lie orthogonal.

A robot can move in discrete time steps by changing its location from a grid position v to an adjacent grid position w ; denoted by $v \rightarrow w$. Two moves $v_1 \rightarrow w_1$ and $v_2 \rightarrow w_2$ are called *collision-free* if $v_1 \neq v_2$ and $w_1 \neq w_2$. A *transformation* between two configurations $C_1 = \{v_1, \dots, v_n\}$ and $C_2 = \{w_1, \dots, w_n\}$ is a set of collision-free moves $\{v_i \rightarrow w_i \mid i = 1, \dots, n\}$. Note that a robot is allowed to hold its position. For $M \in \mathbb{N}$, a *schedule* is a sequence $C_1 \rightarrow \dots \rightarrow C_{M+1}$ (also denoted as $C_1 \rightrightarrows C_{M+1}$) of transformations, with a *makespan* of M . A *stable* schedule $C_1 \rightrightarrows_{\chi} C_{M+1}$ uses only connected configurations. Let C_s, C_t be two connected configurations with equally many robots called *start* and *target configuration*, respectively. A *matching* is a one-to-one mapping between vertices from C_s and C_t . The *diameter* of a matching is the maximal Manhattan distance between two matched vertices. A *bottleneck matching* is a matching with a minimal diameter. The *diameter* d of (C_s, C_t) is the diameter of a bottleneck matching. The *stretch (factor)* of a (stable) schedule is the ratio between the makespan M of the schedule and the diameter d of (C_s, C_t) .

3 Makespan 1 and 2

As a first observation we note that it can be decided in polynomial time whether there is a schedule $C_s \rightarrow C_t$ with a makespan of 1 between a start and a target configuration.

► **Theorem 1.** *For a pair of configurations C_s and C_t , each with n vertices, it can be decided in polynomial time whether there is a schedule with a makespan of 1 transforming C_s into C_t .*

Proof. Given two connected configurations C_s and C_t , each with n vertices. We compute the bipartite graph $G_{C_s, C_t} = (V_s \cup V_t, E)$, where V_s and V_t consist of all occupied positions in C_s and C_t . For E , we add an edge if and only if an occupied position in C_t is adjacent (or identical) to an occupied position in C_s . Consider a perfect matching in G_{C_s, C_t} . Because edges only connect positions which are at most one unit step apart, all robots can move along their respective matching edges without crossing the path of another robot. If there is no perfect matching in G_{C_s, C_t} , at least one robot would have to move to a position further away. Thus, a makespan of 1 would not be achievable. So, there is a schedule of makespan 1 if and only if G_{C_s, C_t} admits a perfect matching. Because the graph is sparse, this can be checked in $\mathcal{O}(n^{3/2})$ time, using the method of Hopcroft and Karp [14]. ◀

Note that, because C_s and C_t have to be connected, a schedule with a makespan of 1 is always stable. Even for a makespan of 2, the same problem becomes provably difficult.

► **Theorem 2.** *For a pair of configurations C_s and C_t , each with n vertices, deciding whether there is a stable schedule with a makespan of 2 transforming C_s into C_t is NP-hard.*

The proof is based on a reduction from the NP-hard problem PLANAR MONOTONE 3SAT [4], which asks to decide whether a Boolean 3-CNF formula φ is satisfiable, for which in each clause the literals are either all unnegated or all negated.

The reduction considers an instance φ of PLANAR MONOTONE 3SAT and constructs an instance I_φ with start configuration C_s and target configuration C_t ; see Figure 3, with start configuration (red), target configuration (dark cyan), and positions in both configurations (gray) indicated by colors. We consider a rectilinear planar embedding of the variable-clause incidence graph G_φ of φ , with variable vertices placed horizontally in a row, and clauses with unnegated and negated literals placed above and below, respectively. Variables of φ are represented by horizontal *variable gadgets* (light red). Two additional *auxiliary gadgets* (light blue) are positioned at the top and at the bottom boundary of the instance, connected to the variable gadget via bridges at the right boundary, and a *separation gadget* (yellow) between each adjacent and nested pair of *clause gadgets* (blue). All clause gadgets are connected via bridges to separation gadgets and possibly to the auxiliary gadgets. Further, there are bridges from a clause gadget to the respectively contained variables.

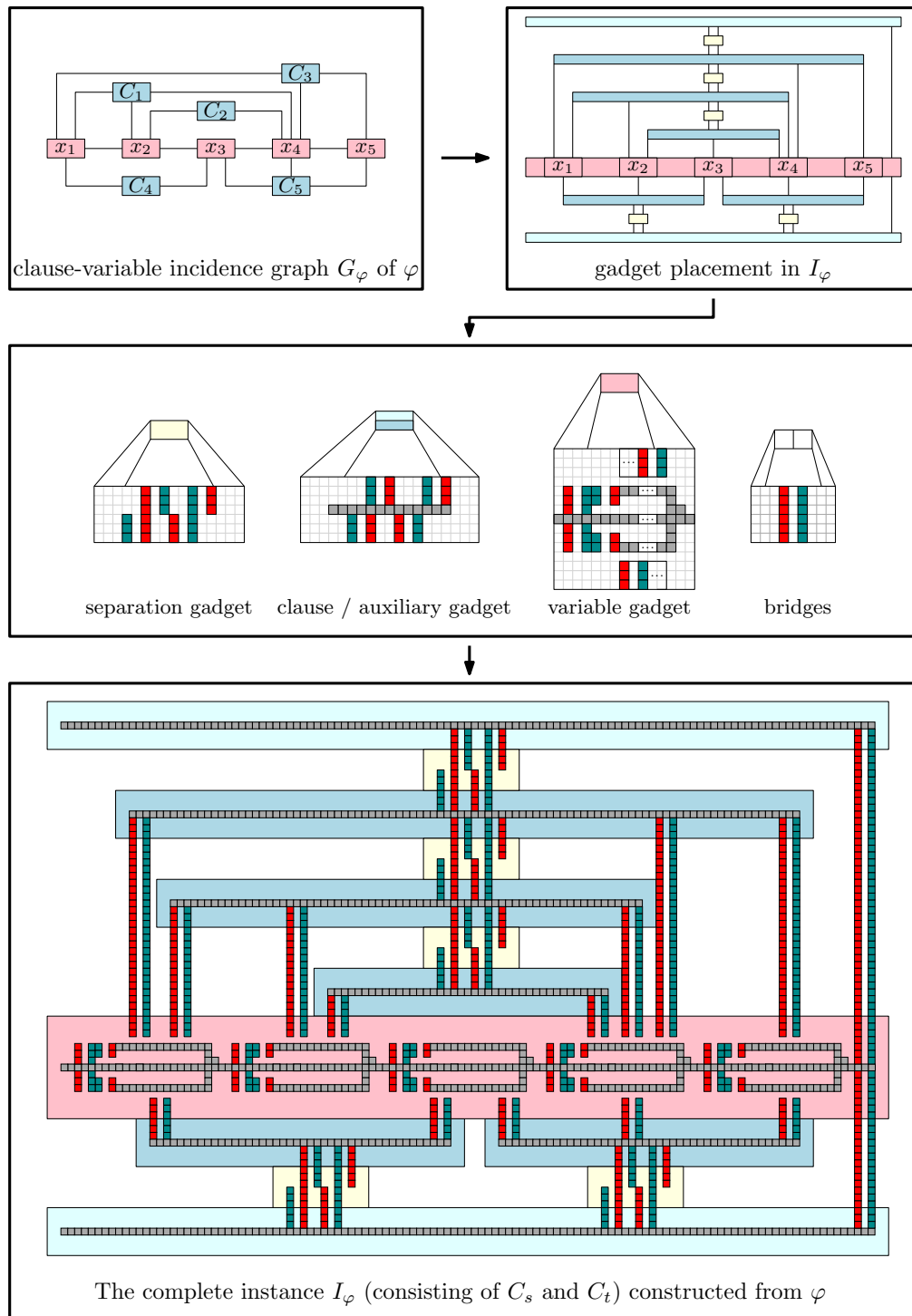
Now a stable schedule for I_φ transforming the start configuration C_s into the target configuration C_t with a makespan of 2 requires some robots of the variable gadget to move in a very particular way to ensure connectivity between the variable gadget and the clause gadgets via corresponding bridges. As shown in technical detail in the full paper, this allows variable robots to connect either to the negated or the unnegated literal of involved clauses, inducing a satisfying variable assignment for φ .

Technical details of the proof of Theorem 2 are given in the full version [11].

As a consequence of Theorem 2, even approximating the makespan is NP-hard.

► **Corollary 3.** *It is NP-hard to compute for a pair of configurations C_s and C_t , each with n vertices, a stable schedule that transforms C_s into C_t within a constant of $(\frac{3}{2} - \varepsilon)$ (for any $\varepsilon > 0$) of the minimum makespan.*

9:6 Connected Coordinated Motion Planning with Bounded Stretch



■ **Figure 3** Symbolic overview of the NP-hardness reduction. The depicted instance is due to the PLANAR MONOTONE 3SAT formula $\varphi = (x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_3 \vee x_4 \vee x_5)$. We use three different colors to indicate occupied positions in the start configuration (red), in the target configuration (dark cyan), and in both configurations (gray).

4 Bounded Stretch for Arbitrary Makespan

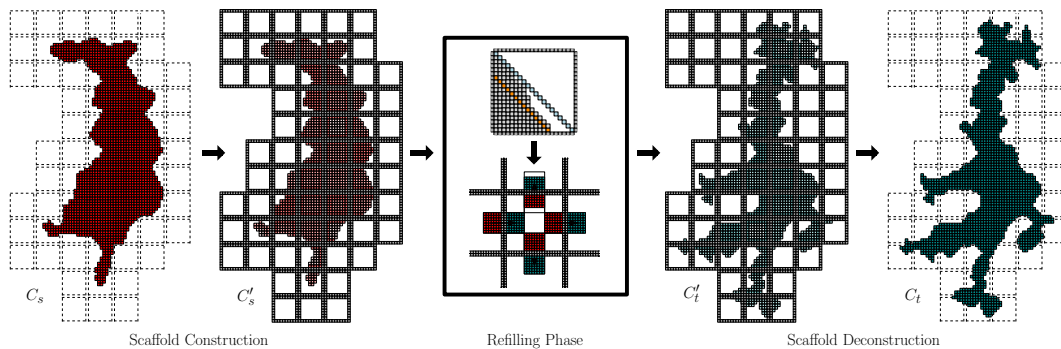
Now we describe our algorithm for computing stable schedules with constant stretch.

► **Theorem 4.** *There is a constant c^* such that for any pair of overlapping start and target configurations with a scale of at least c^* , there is a stable schedule of constant stretch.*

For clearer presentation, we do not focus on the specific value of the constant c^* , but only argue its existence.

4.1 Algorithm Overview and Preliminaries

4.1.1 Informal Outline



■ **Figure 4** Overview of the computed schedule: (Left) Constructing the scaffold of cd -tiles, (middle) the refilling phase, and (right) deconstructing the scaffold.

See Figure 4 for an overview. In two preprocessing phases, we first ensure that the pair (C_s, C_t) overlaps in at least one position. For this, we move C_s towards C_t along a bottleneck matching such that the respective positions that realize the bottleneck distance, coincide. The overlap is necessary to successfully construct the auxiliary structure in the third phase of our approach. Afterwards, we use another bottleneck matching algorithm for mapping the start configuration C_s to the target configuration C_t , minimizing the maximum distance d between a start and a target location. Furthermore, we establish the scale in both configurations, set c to be the minimum of both scale values, and compute a suitable tiling whose tile size is $c \cdot d$, and that contain both C_s and C_t .

In a third phase, we build a scaffolding structure around C_s and C_t , based on the boundaries of cd -tiles of the specific tiling, see Figures 4 (left) and 5. This provides connectivity throughout the actual reconfiguration. Restricting robot motion to their current and adjacent tiles also ensures constant stretch. Note that, as the size of the tiles is related to d , the scaffolding structure is connected.

In a fourth phase, we perform the actual reconfiguration of the arrangement. This consists of refilling the tiles of the scaffold structure, achieving the proper number of robots within each tile, based on elementary flow computations. As a subroutine, we transform the robots inside each tile into a canonical “triangle” configuration, see Figures 4 (middle), 6, and 7.

In a fifth and final phase, we disassemble the scaffolding structure and move the involved robots to their proper destinations, see Figures 4 (right) and 5.

4.1.2 Technical Key Components

On a technical level, the five phases can be summarized as follows; again, refer to Figure 4.

- (1) **Guaranteeing Overlap:** Move the configurations towards each other along a bottleneck matching to ensure that the pair (C_s, C_t) overlaps in at least one position.
- (2) **Preprocessing:** Apply the following three preprocessing steps: (2.1) Set c to be the minimum of c^* and the minimum scale values of C_s and C_t . (2.2) Compute the diameter d of (C_s, C_t) . (2.3) Compute the tiling \mathcal{T} of (C_s, C_t) .

The algorithmic core of our algorithm consists of the following three phases.

- (3) **Scaffold Construction:** Reconfigure the start configuration C_s to a tiled configuration C'_s such that the interior of C'_s is a subset of the start configuration C_s , see Figure 5.
- (4) **Refilling Tiles:** Reconfigure C'_s to a tiled configuration C'_t , such that the interior of C'_t is a subset of the target configuration C_t , see Figures 6 and 7.
- (5) **Scaffold Deconstruction:** Reconfigure C'_t to C_t , see Figure 5.

Note that the scaffold deconstruction is inverse to the scaffold construction.

4.1.3 Preliminaries for the Algorithm

Let $c, d \in \mathbb{N}$ be the scale and the diameter of the pair (C_s, C_t) , respectively. For $x, y \in \mathbb{N}$, a cd -tile T , or *tile* T for short, with *anchor vertex* $(x \cdot cd, y \cdot cd) \in V(G)$ is a set of $(cd)^2$ vertices from the grid G with x -coordinates from the range between $x \cdot cd$ and $x \cdot cd + cd - 1$ and y -coordinates from the range between $y \cdot cd$ and $y \cdot cd + cd - 1$. The *boundary* of T is the set of vertices from T with an x -coordinate equal to $x \cdot cd$ or equal to $x \cdot cd + cd - 1$, or with a y -coordinate equal to $y \cdot cd$ or equal to $y \cdot cd + cd - 1$. The *interior* of T is T without its boundary. The *right, top, left, and bottom sides* of T are the sets of vertices from the boundary of T with maximum x -coordinates, maximum y -coordinates, minimum x -coordinates, and minimum y -coordinates, respectively. The left and right sides of a tile are *vertical sides* and the top and bottom sides are *horizontal sides*. Two tiles T_1, T_2 are *horizontal (vertical) neighbors* if they have two vertical (horizontal) sides $s_1 \subset T_1$ and $s_2 \subset T_2$, such that each vertex from s_1 is adjacent in G to a vertex from s_2 . Two tiles T_1 and T_2 are *diagonal neighbors* if there is another tile T , such that T and T_1 are horizontal neighbors and T and T_2 are vertical neighbors. The *neighborhood* of a tile T is the set of all neighbors of T . A configuration in the interior of a tile T is called *monotone*, if and only if for every robot r in the interior of T all positions to the left and to the bottom are occupied.

A *start tile* is a tile containing a vertex from the start configuration. A *target tile* is a tile containing a vertex from the target configuration. The *cd-tiling* \mathcal{T} of (C_s, C_t) is the union of all start tiles including their neighborhoods and all target tiles. The *scaffold* of \mathcal{T} is the union of all boundaries of tiles from \mathcal{T} . A *cd-tiled configuration* C , or *tiled configuration* C for short, is a configuration that is a subset of \mathcal{T} and a superset of the scaffold of \mathcal{T} . The *interior* of a tiled configuration C is the set of all vertices from C not lying on the scaffold. The *filling level* of a tile $T \in \mathcal{T}$ is the number of robots in the interior of T . The *filling level* of a tiled configuration C is the mapping of each tile onto its filling level in C .

In the following we give the technical description of our algorithm and the corresponding correctness analysis. In particular, we first assume that the start and target configurations overlap in at least one position, resulting in an algorithm guaranteeing constant stretch, and adapt this to the case in that an overlap initially does not exist, afterwards.

4.2 Scaffold Construction

► **Lemma 5.** *For any configuration C_s of scale c there is a stable schedule of makespan $\mathcal{O}(d)$, transforming C_s into a tiled configuration C'_s , with the interior of C'_s being a subset of C_s .*

Outline of the Construction. For the construction we consider $5 \cdot 5$ different classes, based on x - and y -coordinates modulo $5cd$; see Figure 5. We process a single class as follows. For each tile T we consider its *indirect* neighborhood $N[T]$ consisting of all neighbors of T and all neighbors of neighbors of T , i.e., a 5×5 arrangement of tiles centered at T . For constructing the boundary of T , we make use of robots from the interior of a tile in $N[T]$.

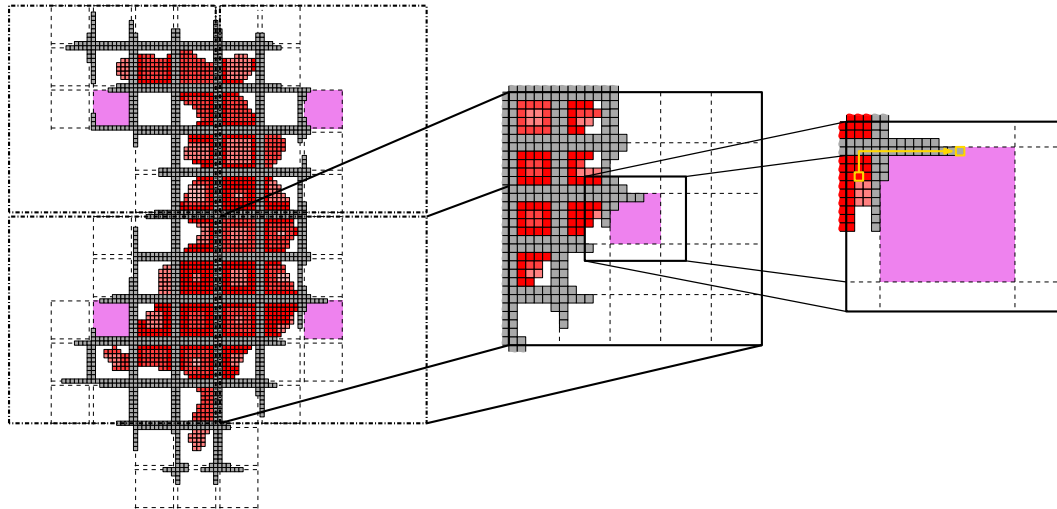
Constructing the Boundary of T works in Two Phases.

(3.1) Constructing the boundaries of all start tiles.

(3.2) Constructing the boundaries of all neighbors of start tiles.

Note that it suffices to construct all boundaries of the start tiles and their neighboring tiles, because each target tile shares a side with a start tile or a side with a tile adjacent to a start tile. Furthermore, the scale condition is only necessary for the construction of the scaffold, i.e., we need to ensure that enough robots are available to build the scaffolding structure. Each additional step of the algorithm works independently from this condition. A very rough estimate on the scale is that $c^* = 400$ is sufficient.

For details of the construction and the proof, we refer to the full version [11].



■ **Figure 5** Constructing the scaffold. Tiles with currently constructed boundaries are marked in purple. The zoom into the start configuration C_s shows the indirect neighborhood $N[T]$ of a tile T (middle) for which its boundary is currently constructed and a further zoom into T with an associated robot motion (right). In each transformation step a robot from the interior of a tile $T' \in N[T]$ is swapped with a free position on the boundary of T based on a path P on a BFS-tree.

4.3 Refilling Tiles

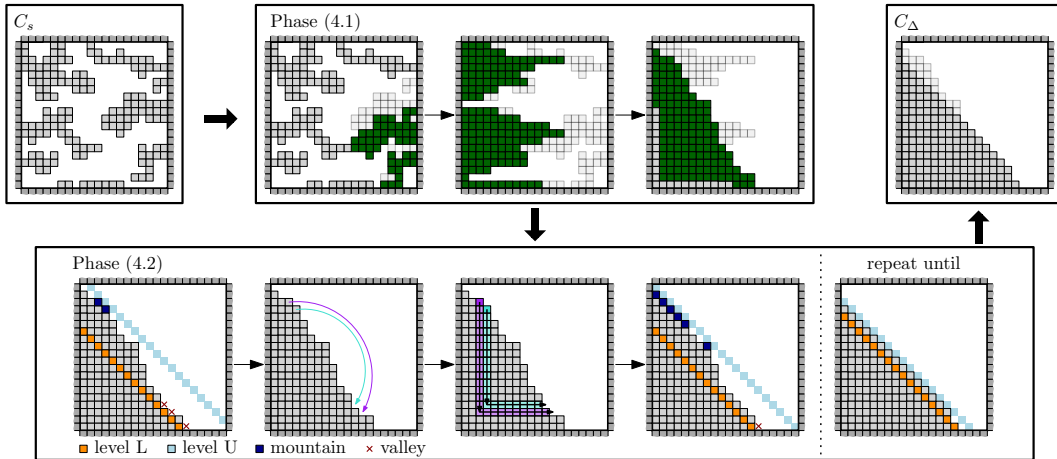
It remains to modify configurations within and between tiles. To this end, we first establish how to efficiently perform reconfigurations between any two tiled configurations *with the same numbers of robots* in the interior of respective tiles; see Section 4.3.1. As a second step, we describe how to relocate robots between tiles such that efficient reconfigurations between any two tiled configurations *with different numbers of robots* in the interior of respective tiles are achieved; see Section 4.3.2.

4.3.1 Reconfiguration Maintaining the Number of Robots inside Tiles

► **Lemma 6.** *Let C'_s, C'_t be two tiled configurations such that for all tiles T , C'_s and C'_t have the same filling levels, i.e., for any tile, the corresponding start and target configurations consist of the same respective numbers of robots. Then there is a stable schedule transforming C'_s into C'_t within a makespan of $\mathcal{O}(d)$.*

In the following we describe reconfigurations that leave all robot movements within the interior of their respective tiles T ; thus, all tiles can be reconfigured in parallel. Therefore, we only have to describe the approach for a start configuration C_s and a target configuration C_t within the interior of a single tile T of a tiled configuration C'_s .

Outline of the Reconfiguration. First compute two stable schedules $C_s \Rightarrow_\chi C_s^m$ and $C_t \Rightarrow_\chi C_t^m$, where C_s^m and C_t^m are monotone configurations. These reconfigurations are achieved by a sequence of down and left movements, maintaining connectivity after each move (see Figure 6 (Phase 4.1)). Proceeding from these monotone configurations, the robots are arranged into a triangular configuration C_Δ that occupies the lower left positions (defined by a diagonal line with a slope of -1) of the interior of T . This is achieved by swapping pairs of occupied and empty positions within a carefully defined area in several one-step moves along L-shaped paths (see Figure 6 (Phase 4.2)). The property of C_Δ is that it is the same for all initial configurations with equally many robots. Thus, to get the stable schedule $C_s \Rightarrow_\chi C_\Delta \Rightarrow_\chi C_t$, we can simply revert $C_t \Rightarrow_\chi C_\Delta$ and combine the result with $C_s \Rightarrow_\chi C_\Delta$.



■ **Figure 6** Turning arrangement C_s (top) into the canonical triangle configuration C_Δ (bottom). Phase (4.1) achieves a monotonic arrangement; light gray indicates previous positions of moved robots (shown in green). Phase (4.2) transforms the monotonic configuration into C_Δ .

Technically, the approach consists of the following four phases, see Figure 6.

- (4.1) **Monotone Start Configuration:** Reconfigure C_s into C_s^m .
- (4.2) **Canonical Triangle:** Reconfigure C_s^m into C_Δ .
- (4.3) **Monotone Target Configuration:** Reconfigure C_Δ into C_t^m .
- (4.4) **Target Configuration:** Reconfigure C_t^m into C_t .

Phase (4.4) corresponds to a reversal of Phase (4.1), and Phase (4.3) to one of Phase (4.2), so we only have to describe the first two phases. We analyze them individually, leading to a proof of Lemma 6. Note that we exclude the corners of a tile, so the robots on the tile’s side now form four *non-adjacent* sides. Furthermore, only robots in a tile’s interior move.

Constructing the Monotone Start Configuration (Phase (4.1)). In the first step, we only consider robots for which the right side of their tile T is the only one to which they are connected through the interior of T . We iteratively move these robots down until further movement is blocked, i.e., any further down move is not collision-free. In the second and third steps, we move all robots left, followed by moving all robots down, each time until further movement is blocked.

For Phase (4.2), we use the following terminology. The *level* of a position in the tile's interior is the sum of its coordinates. A level is *filled*, if all of its positions are occupied by a robot, and *empty*, if none is occupied by a robot. The highest filled level is denoted by L , the lowest empty level by U . Let \mathcal{M} be the set of all positions on level $U - 1$ occupied by a robot, and \mathcal{V} be the set of all positions on level $L + 1$ that are not occupied by a robot (see Figure 6 (Phase 4.2 left)); we call the positions of \mathcal{M} and \mathcal{V} *mountains* and *valleys*, respectively.

Constructing the Canonical Triangle (Phase (4.2)). Choose two equally sized subsets $\mathcal{M}' \subseteq \mathcal{M}$ and $\mathcal{V}' \subseteq \mathcal{V}$ and push each robot from \mathcal{M}' to a different position in \mathcal{V}' along an L-shaped path; this can be done simultaneously in one parallel move for all paths. To determine the paths, simply match mountains and valleys, iteratively, in a way that no pair of paths cross each other. This results in reducing U , and raising L , i.e., the two levels move towards each other. We distinguish two cases.

- $U - L > 2$: If $|\mathcal{M}| \geq |\mathcal{V}|$, choose an arbitrary subset $\mathcal{M}' \subset \mathcal{M}$ with $|\mathcal{M}'| = |\mathcal{V}|$. Otherwise, choose an arbitrary subset $\mathcal{V}' \subset \mathcal{V}$ with $|\mathcal{V}'| = |\mathcal{M}|$.
- $U - L = 2$: Note that mountains and valleys are on the same level, and $|\mathcal{V}| \geq |\mathcal{M}|$ hold. Choose $\mathcal{V}' \subset \mathcal{V}$ to be the subset of size $|\mathcal{M}|$ with *smallest* x -coordinates and set $\mathcal{M}' = \mathcal{M}$.

Due to space constraints, the details can be found in the full version [11].

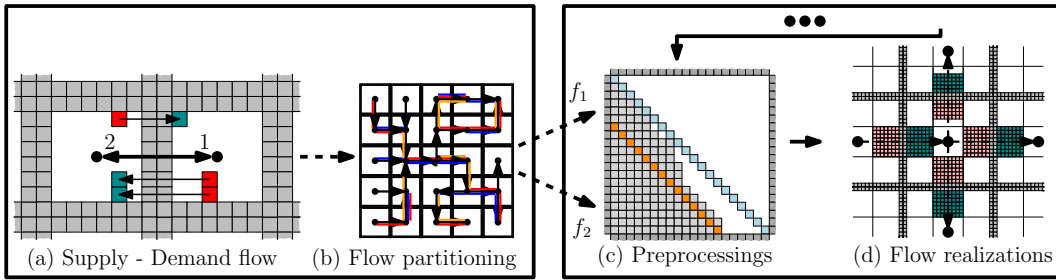
4.3.2 Refilling Tiled Configurations

Now we describe the final step for reconfiguring a tiled start configuration C'_s into a tiled target configuration C'_t ; because no robots are destroyed or created, this hinges on shifting robots between adjacent tiles, such that the required filling levels are achieved.

► **Lemma 7.** *We can efficiently compute a stable schedule transforming C'_s into C'_t within a makespan of $\mathcal{O}(d)$.*

Outline of the Refilling Phase. To compute the schedule of Lemma 7, we transfer robots between tiles, so that each tile T contains the desired number in C'_t . We model this robot transfer by a *supply and demand flow*, see Figure 7, followed by partitioning the flow into $\mathcal{O}(1)$ subflows, such that each subflow can be realized within a makespan of $\mathcal{O}(d)$. For realizing a single subflow, we use the approach of Section 4.3.1 as a preprocessing step, i.e., to rearrange robots participating in a specific subflow and place them at suitable positions.

Modeling Transfer of Robots via a Supply and Demand Flow. We model the transfer of robots between tiles as a flow $F : E(G) \rightarrow \mathbb{N}$, using the directed graph $G = (\mathcal{T}, E)$ which is dual to the tiling \mathcal{T} . Let B be the bottleneck matching between vertices from the original (non-tiled) C_s and vertices from the final (non-tiled) C_t . In G we have an edge $(u, v) \in E$, if there is at least one matching edge $(r_u, r_v) \in B$, such that r_u lies in the interior of the tile u in configuration C'_s , and r_v lies in the interior of the tile v in configuration C'_t . The flow value $F((u, v))$ of (u, v) is equal to the number of such edges $(r_u, r_v) \in B$. A vertex $v \in V(G)$



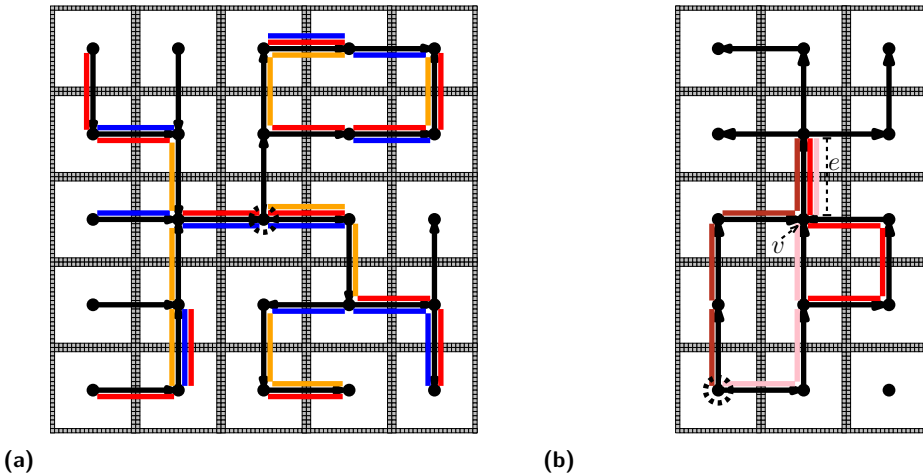
■ **Figure 7** An overview of the schedule refilling tiles: transforming C'_s into C'_t by realizing a partition of a supply and demand flow that is computed in advance.

has a *demand* of $a > 0$ if the sum of the flow values of outgoing edges from v plus a is equal to the sum of the flows of incoming edges to v . Analogously, v has a *supply* of $a > 0$ if the sum of flow values incoming to v plus a equals the sum of flow values outgoing from v .

Flow Partition and Algorithmic Computation. Now we define a *flow partition* of F .

► **Definition 8.** For $k \in \mathbb{N}$, a k -subflow of F is a supply and demand flow $f : E(G) \rightarrow \mathbb{N}$ on G with $f(e) \leq \min\{k, F(e)\}$. A k -partition of the flow F is a set $\{f_1, \dots, f_\ell\}$ of k -subflows of F , such that $\sum_{i=1, \dots, \ell} f_i(e) = F(e)$ holds for all edges $e \in E(G)$.

We describe our approach for computing a ϕ -partition of F , with $\phi := \lfloor \frac{(cd-2)^2}{9} \rfloor$; the value ϕ arises from partitioning the interior (made up of $(cd - 2)^2$ pixels) of each tile into 9 almost equally sized *subtiles* that are used for realizing a single set of paths as described below, see Figure 7(d).



■ **Figure 8** (a) We model the movements of robots between tiles as paths forming a tree. By greedily assigning these paths to sets (here highlighted by different colors), such that inside each set each edge is contained in no more than 3 paths, we obtain that at most $\Theta(d^2)$ sets are needed. (b) An example of a set containing three paths (dark red, pink, and red; assigned in that order to S_j) having a common edge e caused by a vertex v of e with an incoming degree of 3.

We compute a 1-partition of G , with each 1-subflow being either a cycle or a path that connects a supply vertex with a demand vertex. Because the robots are unlabeled, we can simplify G by eliminating all cyclic 1-subflows, as they are not necessary to realize this

specific transfer of robots; note that this also applies to bidirectional edges. Furthermore, we replace diagonal edges $(v, w) \in E(G)$ by a pair of adjacent edges $(v, u), (u, w) \in E(G)$. After all cyclic subflows are removed, G is a planar, directed forest consisting of 1-subflows that are paths, see Figure 8(a). We process each tree $A \subset G$ that is made up of paths P_1, P_2, \dots separately as follows: We choose an arbitrary vertex of A as its *root* and consider the *link distance* of P_i as the minimal length of the path between a vertex from P_i and the root of A . Let $P_1, P_2, \dots \subseteq G$ be sorted by increasing link distances, which is important for our next argument, regarding that we can partition these paths into constant many subflows each of which is realizable in time linear in d . We greedily assign each path $P_i = P_1, P_2, \dots$ to a set S_j , such that the first edge e_1 of P_i is not part of another path inside S_j . If no such a set S_j exists, we create a new set $S \leftarrow \{P_i\}$. For each tree we use the same sets S_1, S_2, \dots of collected paths, because different trees of G are disjoint. Note, that the construction of the sets S_j allow that an edge is part of at most three paths inside S_j . This is due to the fact that the income degree of a head vertex of a directed edge is at most three in the setting of a grid graph, resulting in at most three outgoing edges.

Finally, we greedily partition $\{S_1, S_2, \dots\}$ into subsets G_1, G_2, \dots called *groups*, made up of $\frac{(cd-2)^2}{9 \cdot 3}$ sets. For each group G_i , we define a subflow f_i by setting $f_i(e)$ as the number of paths from G_i containing the edge e . As for each set S_i and each edge e , there are at most three paths inside S_i containing e , each resulting subflow f_i is a $\left(\frac{(cd-2)^2}{9 \cdot 3} \cdot 3\right) = \phi$ -subflow. Finally, we have to upper-bound the number of resulting subflows, i.e., the number of groups.

► **Lemma 9.** *The constructed ϕ -partition $\{f_1, f_2, \dots\}$ consists of at most 28 subflows.*

Realizing a ϕ -Partition. Now we describe how to reconfigure a tiled configuration, such that a ϕ -subflow is removed from G .

► **Definition 10.** *A ϕ -subflow f_i is realized by transforming the current configuration into another configuration, such that for each edge e with $f_i((T, T')) > 0$, the number of robots in the interior of tile T is decreased by $f_i((T, T'))$ and the number of the robots in the interior of tile T' is increased by $f_i((T, T'))$.*

Next we realize a specific ϕ -subflow within a makespan of $\mathcal{O}(d)$. In particular, we partition the interior of each tile T into 9 *subtiles* with equal side lengths (up to rounding), see Figure 7(d). For each subflow f_i , we place $f_i((T, T'))$ robots inside the middle subtile of T that shares an edge with the boundary of T adjacent to T' . In particular, robots placed in the same subtile are arranged in layers of width $\lfloor \frac{cd-2}{9} \rfloor$ as close as possible to the boundary of the tile, see Figure 7(d). The resulting arrangement of robots inside the subtile of T is a *cluster* and T' the *target tile* of the cluster. By a single application of the approach from Section 4.3.1, all clusters of all tiles are arranged simultaneously within a makespan of $\mathcal{O}(d)$. Finally, simultaneously pushing all clusters of all tiles into the direction of their target tiles realizes S_i , see Figure 7(d). Note that not all robots are pushed into the target tile T' but some replace robots on the boundaries between T and T' , see Figure 7(d).

Repeating this approach for each subflow f_i leads to a stable schedule that realizes the entire flow F within a makespan linear in d , i.e., transforms C'_s into C'_t within $\mathcal{O}(d)$ transformation steps, see Figure 7.

The omitted proofs can be found in the full version [11].

This concludes the proof of Theorem 4. Finally, we adapt the result to the general case, for which overlap of the start and target configurations is not guaranteed.

► **Corollary 11.** *There is a constant c^* such that for any pair of start and target configurations with a scale of at least c^* , there is a stable schedule of constant stretch.*

Proof. In case of a pair (C_s, C_t) , consisting of a start and a target configuration, that does not overlap, our algorithm computes in a first step a minimum bottleneck matching mapping C_s to C_t resulting in a bottleneck distance \bar{d} , and translates C_s into a configuration \bar{C}_s overlapping the target configuration within a makespan of \bar{d} . This results in a bottleneck distance d between \bar{C}_s and C_t which is at most $2\bar{d}$. As Theorem 4 guarantees a makespan linear in d , we obtain a makespan linear in $\bar{d} + d$, i.e., linear in \bar{d} for the overall algorithm. ◀

As the diameter of the pair (C_s, C_t) is a lower bound for the makespan of any schedule transforming C_s into C_t , we obtain the following.

► **Corollary 12.** *There is a constant-factor approximation for computing stable schedules with minimal makespan between pairs of start and target configurations with a scale of at least c^* , for some constant c^* .*

5 Conclusion

We have shown that connected coordinated motion planning is challenging even in relatively simple cases, such as unlabeled robots that have to travel a distance of at most 2 units. On the other hand, we have shown that (assuming sufficient scale of the swarm), it is possible to compute efficient reconfiguration schedules with constant stretch.

It is straightforward to extend our approach to other scenarios, e.g., to three-dimensional configurations. Other questions appear less clear. Is it possible to achieve constant stretch for arrangements with very small scale factor? We believe that this may hinge on the ability to perform synchronized shifts on long-distance “chains” of robots without delay, which is not a valid assumption for many real-world scenarios. (A well-known example is a line of cars when a traffic light turns green.) As a consequence, the answer may depend on crucial assumptions on motion control; we avoid this issue in our approach. Can we provide alternative approaches with either weaker scale assumptions or better stretch factors? Can we extend our methods to the labeled case? All these questions are left for future work.

References

- 1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient Multi-Robot Motion Planning for Unlabeled Discs in Simple Polygons. *IEEE Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.
- 2 Aaron T. Becker, Sándor P. Fekete, Phillip Keldenich, Matthias Konitzny, Lillian Lin, and Christian Scheffer. Coordinated Motion Planning: The Video. In *Symposium on Computational Geometry (SoCG)*, pages 74:1–74:6, 2018. Video at <https://www.ibr.cs.tu-bs.de/users/fekete/Videos/CoordinatedMotionPlanning.mp4>. doi:10.4230/LIPIcs.SoCG.2018.74.
- 3 Soon-Jo Chung, Aditya Avinash Paranjape, Philip Dames, Shaojie Shen, and Vijay Kumar. A Survey on Aerial Swarm Robotics. *IEEE Transactions on Robotics*, 34(4):837–855, 2018. doi:10.1109/TR0.2018.2857475.
- 4 Mark de Berg and Amirali Khosravi. Optimal Binary Space Partitions for Segments in the Plane. *International Journal on Computational Geometry and Applications*, 22(3):187–206, 2012. doi:10.1142/S0218195912500045.
- 5 Daniel Delahaye, Stéphane Puechmorel, Panagiotis Tsiotras, and Eric Féron. Mathematical Models for Aircraft Trajectory Design: A Survey. In *Air Traffic Management and Systems*, pages 205–247, 2014. doi:10.1007/978-4-431-54475-3_12.
- 6 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane Souvaine. Staged Self-assembly: Nanomanufacture of Arbitrary Shapes with $O(1)$ Glues. *Natural Computing*, 7(3):347–370, 2008. doi:10.1007/s11047-008-9073-0.

- 7 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Henk Meijer, and Christian Scheffer. Coordinated Motion Planning: Reconfiguring a Swarm of Labeled Robots with Bounded Stretch. *SIAM Journal on Computing*, 48(6):1727–1762, 2019. doi:10.1137/18M1194341.
- 8 Erik D. Demaine, Sándor P. Fekete, Christian Scheffer, and Arne Schmidt. New Geometric Algorithms for Fully Connected Staged Self-Assembly. *Theoretical Computer Science*, 671:4–18, 2017. doi:10.1016/j.tcs.2016.11.020.
- 9 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-Assembly of Arbitrary Shapes Using RNase Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor. In *Symposium on Theoretical Aspects of Computer Science*, volume 9, pages 201–212, 2011. doi:10.4230/LIPIcs.STACS.2011.201.
- 10 Sándor P. Fekete, Björn Hendriks, Christopher Tessars, Axel Wegener, Horst Hellbrück, Stefan Fischer, and Sebastian Ebers. Methods for Improving the Flow of Traffic. In *Organic Computing — A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*. Springer, 2011. doi:10.1007/978-3-0348-0130-0_29.
- 11 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected Coordinated Motion Planning with Bounded Stretch, 2021. arXiv:2109.12381.
- 12 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing Coordinated Motion Plans for Robot Swarms: The CG:SHOP Challenge 2021, 2021. arXiv:2103.15381.
- 13 Seth C. Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots. In *Robosphere 2004*, 2004. URL: <http://www.cs.cmu.edu/~claytronics/papers/goldstein-robosphere04.pdf>.
- 14 John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973. doi:10.1137/0202019.
- 15 Stephen Kloder and Seth Hutchinson. Path planning for permutation-invariant multi-robot formations. *IEEE Transactions on Robotics and Automation*, 22(4):650–665, 2006. doi:10.1109/TR0.2006.878952.
- 16 Austin Luchsinger, Robert T. Schweller, and Tim Wylie. Self-assembly of shapes at constant scale using repulsive forces. *Natural Computing*, 18(1):93–105, 2019. doi:10.1007/s11047-018-9707-9.
- 17 Florian Pescher, Nils Napp, Benoît Piranda, and Julien Bourgeois. GAPCoD: A Generic Assembly Planner by Constrained Disassembly. In *Autonomous Agents and MultiAgent Systems*, pages 1028–1036, 2020. URL: <https://dl.acm.org/doi/abs/10.5555/3398761.3398881>.
- 18 Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014. doi:10.1126/science.1254295.
- 19 Erol Şahin and Alan F. T. Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2):69–72, 2008. doi:10.1007/s11721-008-0020-6.
- 20 Michael Schreckenberg and Reinhard Selten. *Human Behaviour and Traffic Networks*. Springer, 2013. doi:10.1007/978-3-662-07809-9.
- 21 Jacob T. Schwartz and Micha Sharir. On the piano movers’ problem: III. Coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. *International Journal of Robotics Research*, 2(3):46–75, 1983. doi:10.1177/027836498300200304.
- 22 David Soloveichik and Erik Winfree. Complexity of Self-Assembled Shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. doi:10.1137/S0097539704446712.
- 23 Kiril Solovey and Dan Halperin. k -Color Multi-Robot Motion Planning. *International Journal of Robotics Research*, 33(1):82–97, 2014. doi:10.1177/0278364913506268.
- 24 Kiril Solovey and Dan Halperin. On the Hardness of Unlabeled Multi-Robot Motion Planning. *International Journal of Robotics Research*, 35(14):1750–1759, 2016. doi:10.1177/0278364916672311.

9:16 Connected Coordinated Motion Planning with Bounded Stretch

- 25 Kiril Solovey, Jingjin Yu, Or Zamir, and Dan Halperin. Motion Planning for Unlabeled Discs with Optimality Guarantees. In *Robotics: Science and Systems*, 2015. doi:10.15607/RSS.2015.XI.011.
- 26 Pierre Thalamy, Benoît Piranda, and Julien Bourgeois. Distributed Self-Reconfiguration using a Deterministic Autonomous Scaffolding Structure. In *Autonomous Agents and MultiAgent Systems*, pages 140–148, 2019. URL: <https://dl.acm.org/doi/abs/10.5555/3306127.3331685>.
- 27 Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory planning and assignment in multirobot systems. In *Algorithmic Foundations of Robotics X*, pages 175–190. Springer, 2013. doi:10.1007/978-3-642-36279-8_11.
- 28 Matthew Turpin, Kartik Mohta, Nathan Michael, and Vijay Kumar. Goal assignment and trajectory planning for large teams of interchangeable robots. *Autonomous Robots*, 37(4):401–415, 2014. doi:10.1007/s10514-014-9412-1.