



Efficiently Reconfiguring a Connected Swarm of Labeled Robots

Sándor P. Fekete ✉ 

Department of Computer Science, TU Braunschweig, Germany

Peter Kramer ✉ 

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, Germany

Arne Schmidt ✉ 

Department of Computer Science, TU Braunschweig, Germany

Abstract

When considering motion planning for a swarm of n labeled robots, we need to rearrange a given start configuration into a desired target configuration via a sequence of parallel, continuous, collision-free robot motions. The objective is to reach the new configuration in a minimum amount of time; an important constraint is to keep the swarm connected at all times. Problems of this type have been considered before, with recent notable results achieving *constant stretch* for not necessarily connected reconfiguration: If mapping the start configuration to the target configuration requires a maximum Manhattan distance of d , the total duration of an overall schedule can be bounded to $\mathcal{O}(d)$, which is optimal up to constant factors. However, constant stretch could only be achieved if *disconnected* reconfiguration is allowed, or for scaled configurations (which arise by increasing all dimensions of a given object by the same multiplicative factor) of *unlabeled* robots.

We resolve these major open problems by (1) establishing a lower bound of $\Omega(\sqrt{n})$ for connected, labeled reconfiguration and, most importantly, by (2) proving that for scaled arrangements, constant stretch for connected reconfiguration can be achieved. In addition, we show that (3) it is NP-hard to decide 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.

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

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

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.17

Related Version *Full Version:* <https://arxiv.org/abs/2209.11028> [20]

1 Introduction

Motion planning for sets of objects is a theoretical and practical problem of great importance. A typical task arises from relocating a large collection of agents from a given start into a desired goal configuration, while avoiding collisions between objects or with obstacles. Previous work has largely focused on achieving reconfiguration via sequential schedules, where one robot moves at a time; however, reconfiguring *efficiently* requires reaching the target configuration in a timely or energy-efficient manner, with a natural objective of minimizing the time until completion, called *makespan*. Achieving minimum makespan for reconfiguring a swarm of labeled robots was the subject of the 2021 Computational Geometry Challenge [19]; see [8, 25, 37] for successful contributions.



© Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt; licensed under Creative Commons License CC-BY 4.0

33rd International Symposium on Algorithms and Computation (ISAAC 2022).

Editors: Sang Won Bae and Heejin Park; Article No. 17; pp. 17:1–17:15

Leibniz International Proceedings in Informatics



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

Exploiting parallelism in a robot swarm to achieve an efficient schedule was studied in recent seminal work by Demaine et al. [11]: Under certain conditions, a labeled set of robots can be reconfigured with bounded *stretch*, i.e., there is a collision-free motion plan such that the makespan of the schedule remains within a constant of the lower bound that arises from the maximum distance between origin and destination of individual robots; see also the video by Becker et al. [4] that illustrates these results.

A second important aspect for many applications is *connectivity* of the swarm throughout the reconfiguration, because disconnected pieces may not be able to regain connectivity, and also because of small-scale swarm robots (such as catoms in claytronics [22]), which need connectivity for local motion, electric power and communication; see the video by Bourgeois et al. [5]. Connectivity is not necessarily preserved in the schedules by Demaine et al. [11]. In more recent work, Fekete et al. [18] presented an approach that does achieve constant stretch for *unlabeled* swarms of robots for the class of *scaled* arrangements; such arrangements arise by increasing all dimensions of a given object by the same multiplicative factor and have been considered in previous seminal work on self-assembly, often with unbounded or logarithmic scale factors (along the lines of what has been considered in self-assembly [31]). The method by Fekete et al. [18] relies strongly on the exchangeability of indistinguishable robots, which allows a high flexibility in allocating robots to target configurations, which is not present in labeled reconfiguration.

These results have left two major open problems.

1. Can efficient reconfiguration be achieved in a *connected* manner for a swarm of *labeled* robots in a not necessarily scaled arrangement?
2. Is it possible to achieve constant stretch for connected reconfiguration of scaled arrangements of *labeled* objects?

1.1 Our contributions

We resolve both of these open problems.

1. We show that connected reconfiguration of a swarm of n labeled robots may require a stretch factor of at least $\Omega(\sqrt{n})$.
2. We present a framework for achieving *constant* stretch for connected reconfiguration of scaled arrangements of labeled objects.
3. In addition, we show that it is NP-hard even to decide whether a makespan of 2 for labeled connected reconfiguration can be achieved.

1.2 Related work

Algorithmic efforts for multi-robot coordination date back to the seminal work by Schwartz and Sharir [30] from the 1980s. Efficiently coordinating the motion of many agents arises in a large spectrum of applications, such as air traffic control [9], vehicular traffic networks [17, 29], ground swarm robotics [27, 28], or aerial swarm robotics [7, 36]. 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 [32]. In the *unlabeled* case, objects are indistinguishable and target positions can be covered by any object; see Kloder and Hutchinson [23], Turpin et al. [35], Adler et al. [1], and Solovey et al. [34]. On the negative side, Solovey and Halperin [33] prove that the unlabeled multiple-object motion planning problem is PSPACE-hard. Calinescu, Dumitrescu, and Pach [6] consider the

sequential reconfiguration of objects lying on vertices of a graph. They give NP-hardness and inapproximability results for several variants, a 3-approximation algorithm for the unlabeled variant, as well as upper and lower bounds on the number of sequential moves needed.

We already described the work by Demaine et al. [11] for achieving constant stretch for coordinated motion planning, as well as the recent practical CG Challenge 2021 [8, 19, 25, 37]. None of these approaches satisfy the crucial connectivity constraint, which has previously been investigated in terms of decidability and feasibility by Dumitrescu and Pach [14] and Dumitrescu, Suzuki, and Yamashita [16]. Furthermore, these authors have also proposed efficient patterns for fast swarm locomotion in the plane using sequential moves that allow preservation of connectivity [15]. A closely related body of research concerns itself with sequential pivoting moves that require additional space around moving robots, limiting feasibility and reachability of target states, see publications by Akitaya et al. [2, 3].

Very recently, Fekete et al. [18] presented a number of new results for connected, but unlabeled reconfiguration. In addition to complexity results for small makespan, they showed 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. The involved concept of scale has received considerable attention in self-assembly; achieving constant scale has required special cases or operations. Soloveichik and Winfree [31] 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. [13] 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. [10] achieved logarithmic scale, and constant scale for more constrained classes of polyomino shapes; this was later improved by Demaine et al. [12] to constant scale for a logarithmic number of stages. More recently, Luchsinger et al. [26] employed repulsive forces between tiles to achieve constant scale in two-handed self-assembly.

For further related work see Demaine et al. [11].

1.3 Preliminaries

We consider *robots* at nodes of the (integer) infinite grid $G = (V, E)$, where two nodes are connected if and only if they are in unit distance. A *configuration* is a mapping $C : V \rightarrow \{1, \dots, n, \perp\}$, i.e., each node is mapped injectively to one of the n labeled robots, or to \perp if the node is empty. For a robot ℓ , $C^{-1}(\ell) = (x_\ell, y_\ell)$ refers to its x - and y -coordinate. The configuration C is *connected* if the subgraph $H \subset G$ induced by occupied nodes in C is connected. The *silhouette* of a configuration C is the respective unlabeled configuration, i.e., C without labeling. Unless stated otherwise, we consider labeled connected configurations.

Two configurations *overlap*, if they have at least one occupied position in common. A configuration C is *c-scaled*, if H 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., $\{v_1, v_2\} \in E(H)$.

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 *collision-free* if $v_1 \neq v_2$ and $w_1 \neq w_2$. Note that a *swap*, i.e., two moves $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_1$, causes

17:4 Reconfiguring a Connected Swarm of Labeled Robots

a collision and is not allowed in our model. A *transformation* between two configurations C_1 and C_2 is a set of collision-free moves $\{v \rightarrow w \mid C_1(v) = C_2(w) \neq \perp \wedge |v - w| \leq 1\}$. 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 abbreviated as $C_1 \Rightarrow C_{M+1}$) of transformations, with a *makespan* of M . A *stable* schedule $C_1 \Rightarrow_{\chi} C_{M+1}$ uses only connected configurations. In the context of this paper, we use these notations equivalently.

Let C_s and C_t be two connected configurations with equally many robots called *start* and *target configuration*, respectively. The *diameter* d of the pair (C_s, C_t) is the maximal Manhattan distance between a robot's start and target position. The *stretch (factor)* of a schedule is the ratio between its makespan M and the diameter d of (C_s, C_t) .

2 Fixed makespan

Given two labeled configurations, it is easy to see that it can be determined in linear time whether there is a schedule with a makespan of 1 that transforms one into the other: For every robot, check whether its target position is in distance at most 1; furthermore, check that no two robots want to swap their positions. This involves $O(1)$ checks for every robot, thus $O(n)$ checks in total. We obtain the following.

► **Theorem 1.** *It can be decided in $O(n)$ time whether there is a schedule $C_s \Rightarrow_{\chi} C_t$ with makespan 1 for any pair (C_s, C_t) of labeled configurations, with n robots each.*

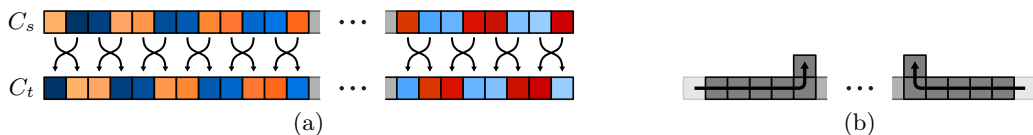
On the other hand, Fekete et al. [18] showed that it is already NP-hard to decide whether a schedule with a makespan of 2 can be achieved, if the robot swarm is unlabeled. Due to the desired makespan, the respective target position of every robot is highly restricted. Thus, it is straightforward to provide a suitable labeling of the configurations such that the very same construction shows NP-hardness for the variant of labeled robot swarms.

For technical details, see the full version [20].

► **Theorem 2.** *It is NP-hard to decide whether there is a schedule $C_s \Rightarrow_{\chi} C_t$ with makespan 2 for any pair (C_s, C_t) of labeled configurations, with n robots each.*

3 Lower bound on stretch factor

We sketch a lower bound of $\Omega(\sqrt{n})$ on the stretch factor. For this, we consider the pair of configurations (C_s, C_t) shown in Figure 1(a), both consisting of n robots. The difference between both configurations is that adjacent robots need to swap their positions. Thus, the diameter of (C_s, C_t) is $d = 1$. Because swaps are not allowed within the underlying model, some robots have to move orthogonally.



■ **Figure 1** Pairs of robots must swap their positions (a), using moves that involve all robots (b).

Any robot occupying a position adjacent to the line after moving orthogonally can be utilized to perform swaps between robots still contained within the line itself. However, for each robot we move out of the line, we have to simultaneously perform a “shrinking”

motion along the line itself to preserve connectivity, see Figure 1(b). Acquiring λ such robots therefore takes $\lambda/2$ steps. To perform n pairwise swaps, we then need roughly n/λ steps. This sum has a minimum at $\lambda = \sqrt{n}$, and thus we obtain the following theorem.

► **Theorem 3.** *There are pairs of labeled configurations C_s and C_t , each with n vertices, so that every schedule $C_s \Rightarrow_\chi C_t$ has a stretch factor of at least $\Omega(\sqrt{n})$.*

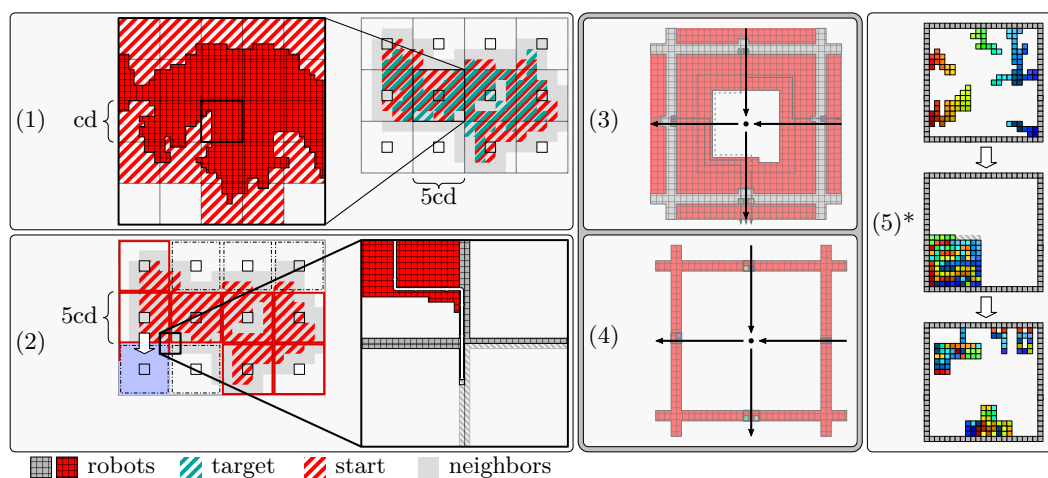
4 Schedules of constant stretch

In this section, we describe an approach to determine stable schedules with constant stretch for labeled configurations of sufficient scale. In particular, we show the following result.

► **Theorem 4.** *There is a constant c^* such that for any pair (C_s, C_t) of labeled configurations with n robots each and scale of at least c^* , there exists a constant stretch schedule $C_s \Rightarrow_\chi C_t$.*

4.1 Overview of the algorithm

Our approach works in different phases; Figure 2 provides an overview. Based on (1) preprocessing steps (Section 4.4) in which we determine the scale c and the diameter d of the configurations, we (2) build a tile-based scaffolding structure (Section 4.5) that guarantees connectivity during the reconfiguration. Then the actual reconfiguration (3+4) consists of shifting robots between adjacent tiles based on flow computations (Sections 4.6–4.9), and (5) reconfiguring tiles in parallel (Sections 4.3 and 4.10). Finally, the deconstruction of the scaffold yields the target configuration. The phases can be summarized as follows.



► **Figure 2** Overview of our algorithm. *Note that ideas of Phase (5) are also used as a subroutine in Phases (2) to (4).

Phase (1) Preprocessing: Compute scale c , diameter d , a tiling \mathcal{T}_1 of the grid into squares of size $cd \times cd$, and a larger tiling \mathcal{T}_5 covering all non-empty tiles of \mathcal{T}_1 .

Phase (2) Scaffold construction: Construct a scaffold along the edges of \mathcal{T}_5 , resulting in a tiled configuration, guaranteeing connectivity during reconfiguration.

Phase (3) Interior flow: Create a flow graph $G_{\mathcal{T}_5}$ to model the movement of interior (non-scaffold) robots between adjacent tiles of \mathcal{T}_5 . Convert the flow into a series of moves, placing all interior robots in their target tiles.

Phase (4) Boundary flow: Analogously to Phase (3), create a flow for the robots that were used to construct a scaffold in Phase (2). Afterwards, move all of those robots into the boundary of their target tiles.

Phase (5) Local tile reconfiguration: Locally reconfigure all tiles of the resulting tiled configuration.

Phase (6) Scaffold deconstruction: Reverse of Phase (2).

In the remainder of this section we provide details of the different phases. We start by providing some preliminaries needed for the detailed descriptions.

4.2 Preliminaries for the algorithm

In the following we give definitions that are fundamental for the understanding of the algorithm. We give additional definitions in each section as needed.

As an intermediate result for the labeled case, Demaine et al. [11] proposed an algorithm that computes schedules with stretch factors that are linear in the dimensions of a fully occupied rectangular area that is to be reconfigured.

► **Lemma 5.** *Let C_s and C_t be two labeled configurations of an $n_1 \times n_2$ rectangle with $n_1 > n_2 \geq 2$. There is a schedule $C_s \Rightarrow_\chi C_t$ with makespan $O(n_1 + n_2)$.*

In a follow-up paper, Fekete et al. [18] considered arbitrary unlabeled configurations, computing stable schedules of constant stretch. Note that stretch in the unlabeled case is defined via a bottleneck matching of robots between the start and target configuration. They make use of so-called *tilings*, *neighborhoods* of tiles, *layers* in tiles, a *scaffold* based on these layers, and *tiled configurations*, which we define as follows.

An m -tiling is a subdivision of a configuration's underlying grid into squares (called *tiles*) of side length $m \in \mathbb{N}$, each of them anchored at coordinates that are multiples of m in both dimensions. With scale c , diameter d , and $m = cd$, we refer to this tiling as \mathcal{T}_1 . For a tiling \mathcal{T} and a subset of tiles $\mathcal{T}' \subseteq \mathcal{T}$, the k -neighborhood $N_k[\mathcal{T}']$ is the set of all tiles from \mathcal{T} with Chebyshev distance at most k to any tile $T \in \mathcal{T}'$. The *boundary* of a tile consists of all nodes in the underlying grid graph that are immediately adjacent to its edge. The first *layer* of the tile is then the set of inward neighboring nodes of the boundary. This relationship applies to successive higher-order layers as well. The *scaffold* of a tiling is the union of all boundaries of its tiles. A *tiled configuration* is a configuration that is a subset of the given tiling and a superset of its scaffold. The *interior* of a tiled configuration is the set of all robots not part of the scaffold.

As an intermediate result they showed the following.

► **Lemma 6.** *Let C_s and C_t be two tiled unlabeled configurations such that C_s and C_t contain the same number of robots in the interior of each tile T . There is a schedule $C_s \Rightarrow_\chi C_t$ with makespan $O(d)$.*

4.3 Subroutine: Single tile reconfiguration

A key insight for our approach is that we can efficiently exploit a globally connected structure locally. Before explaining how we achieve this global structure, we show how to exploit it locally. This provides a fundamental subroutine that is used to locally transform tiled configurations within a makespan of $O(d)$. In particular, we obtain the following.

► **Theorem 7.** *For any two tiled connected configurations C_s and C_t for which each tile consists of the same robots, there is a stable schedule of makespan $O(d)$ that transforms one into the other.*

In the remainder of this subsection we provide several lemmas that yield a proof of Theorem 7, see full version [20] for details. To this end, we first show that the interior of a tile can be reconfigured arbitrarily, followed by a description of how this can be adapted to include the reconfiguration of the boundary, as well as arbitrary exchanges of robots between a tile’s interior and its boundary.

We start by showing how the interior of a tile can be transformed arbitrarily. As moves are reversible, proving that a canonical configuration is reachable is sufficient. Using a technique outlined by Fekete et al. [18], we can gather all robots in a corner of a tile. A straightforward compacting process then yields a configuration that is a subset of a square. Overall this takes $O(d)$ transformations. We thus obtain the following lemma.

► **Lemma 8.** *For any two connected configurations consisting of an immobile $m \times m$ boundary for $m \in O(d)$ and $k \leq (m - 2)^2$ interior robots, there is a stable schedule of makespan $O(d)$ that moves all interior robots into a subset of a $\lceil \sqrt{k} \rceil \times \lceil \sqrt{k} \rceil$ square.*

By applying Lemma 5 to this square we can rearrange the robots in $O(d)$ steps. We obtain the following lemma; see Figure 2, Phase (5) for the resulting procedure.

► **Lemma 9.** *Given $k \in [1, (m - 2)^2]$ robots arranged in a subset of a $\lceil \sqrt{k} \rceil \times \lceil \sqrt{k} \rceil$ square, the robots may be arbitrarily rearranged by a stable schedule of makespan $O(d)$.*

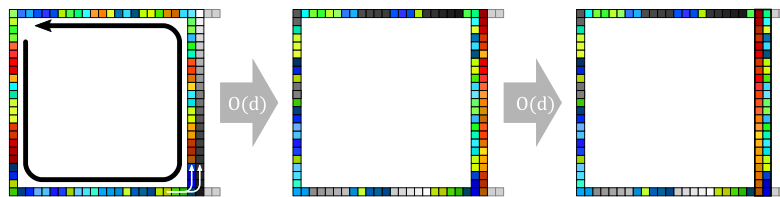
Note that if the $\lceil \sqrt{k} \rceil \times \lceil \sqrt{k} \rceil$ square is not completely occupied, we apply Lemma 5 at most three times on different parts of the square to obtain any desired configuration. See the full version for details [20].

We now show how to exchange robots between the tiles’ interiors and their boundaries.

► **Lemma 10.** *For any tiled configuration of $m \times m$ -tiles for $m \in O(d)$, it is possible to exchange any number of robots from each tile’s interior with its boundary in $O(d)$ steps.*

Proof. Place at most $4(m - 4)$ robots that need to be swapped into the boundary, adjacent to the respective boundary robots that need to swap into the interior. Afterwards apply Lemma 5 to the disjoint areas in parallel. As there are fewer positions on a successive layer, we have to repeat this process at most once. As this operation is performed entirely within a single tile, it may be applied to all tiles simultaneously. ◀

It remains to show that the scaffolding structure can be reconfigured arbitrarily, without modifying its silhouette. To reorder any given tile’s boundary, we make use of the $2 \times d$ scaffold robots that separate it from a neighbor’s interior. By doing a full revolution of the boundary, we can collect any subset of d robots in that portion of the scaffold, which we then reorder using Lemma 5, see Figure 3. Repeating this at most four times yields a sorted boundary, so we obtain the following lemma. Details can be found in the full version [20].



■ **Figure 3** An illustration of the first iteration of the boundary reconfiguration approach.

► **Lemma 11.** *The boundaries of all tiles within a tiled configuration that consists of $m \times m$ -tiles with $m \in O(d)$ may be locally reordered by a schedule of makespan $O(d)$.*

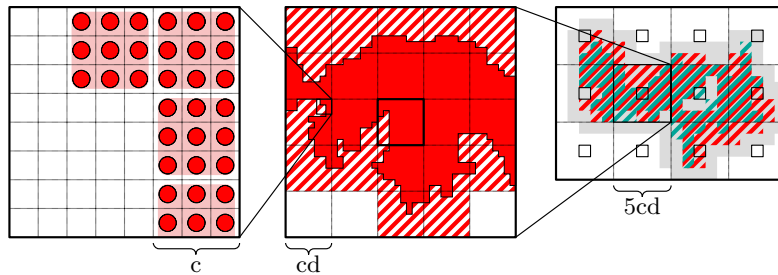
4.4 Phase (1): Preprocessing

Consider start and target configurations C_s and C_t . Let c refer to the minimum of the two configurations' scales and let d be the diameter of the pair (C_s, C_t) . Without loss of generality, assume that the configurations overlap in at least one position. Otherwise, we move the target configuration, such that it overlaps with the start configuration. This can be done in $O(d)$ steps, and results in a new diameter $d' \leq 2d$.

Using c and d , we define a cd -tiling \mathcal{T}_1 of the underlying grid. Let $\mathcal{T}_1(C_s)$ and $\mathcal{T}_1(C_t)$ refer to the set of tiles in \mathcal{T}_1 that contain robots in C_s and C_t , respectively (see red and green area in Figure 4). We observe that $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)]$ is both a subset of $N_2[\mathcal{T}_1(C_s)]$ and of $N_2[\mathcal{T}_1(C_t)]$, because $\mathcal{T}_1(C_s)$ and $\mathcal{T}_1(C_t)$ are fully contained in each other's 1-neighborhoods.

Demaine et al. [11] exploited an $O(d)$ -tiling to guarantee that every robot's target position is either within its starting tile, or an immediate neighbor. In an extension of this approach, Fekete et al. [18] employed a cd -tiling; they constructed a scaffold along the tiling boundaries, achieving connectivity during the reconfiguration process. For configurations of sufficient scale, they drew robots from 2-neighborhoods in order to construct each tile's boundary. In our case, robots have individual target positions, so we must take their heading into account when constructing a scaffold. As a result, we consider a higher-resolution tiling, which allows us to ensure that the diameter of the instance does not increase due to scaffold construction.

Based on the non-empty tiles in \mathcal{T}_1 , we compute the higher-resolution cover of the relevant area by a grid of 5×5 squares of cd -tiles, see Figure 4. Let T_1, \dots, T_n be n tiles of \mathcal{T}_1 , such that the distance between any two of them is a multiple of 5 on both the x - and y -axis, and $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)] \subseteq N_2[T_1] \cup \dots \cup N_2[T_n]$. We define the tiling $\mathcal{T}_5 := \{N_2[T_1], \dots, N_2[T_n]\}$.



■ **Figure 4** We derive a cd -tiling \mathcal{T}_1 (center) from the scale c (left) and the diameter of the instance. We then cover the neighborhood (right, in gray) of non-empty start and target tiles (dashed red and green, respectively) of \mathcal{T}_1 by a larger tiling \mathcal{T}_5 around which we construct a scaffold for stability.

This concludes the theoretical foundation of our approach. We will now proceed with the first transformation phase, which constructs the fundamental scaffold.

4.5 Phase (2): Scaffold construction

Having determined a cover of C_s and C_t in shape of the $5cd$ -tiling \mathcal{T}_5 , a scaffold spanning this cover is to be constructed. For this purpose, a robot is placed at every boundary position in \mathcal{T}_5 , which requires $(5 \cdot 4cd - 4)$ robots per tile. We show that there are sufficiently many robots to build the scaffold. For this, we refer to the *locally available material* for a given tile $T \in \mathcal{T}_5$ as the set of robots contained within T itself and its immediate neighborhood $N_1[T]$ over \mathcal{T}_5 .

► **Lemma 12.** *There is a constant c , such that for all C_s and C_t with scale at least c , there is sufficient locally available material to construct a boundary around all tiles of \mathcal{T}_5 .*

As \mathcal{T}_5 is a cover of $N_1[\mathcal{T}_1(C_s) \cup \mathcal{T}_1(C_t)]$, every tile $T \in \mathcal{T}_5$ contains at least one $T' \in N_2[\mathcal{T}_1(C_s)]$. We can thus guarantee sufficient locally available material for the boundary of T if we can show that it exists in the 4-neighborhood of any such T' .

To achieve this, we distinguish *donor* and *recipient* tiles. A donor tile contains enough robots to construct a boundary around itself and all eight immediate neighbors – any tile that cannot do this will be referred to as a recipient. Suppose there is a recipient T such that $N_1[T]$ does not contain a donor. Due to connectivity, we can show that there must be a path that connects T with some tile non-adjacent tile $T' \notin N_1[T]$. Because the path must cross $N_1[T]$, one of the neighboring tiles must have at least $(c^2d)/4$ robots and is thus a donor tile, contradicting the premise. This directly implies that the 1-neighborhood $N_1[T]$ of any $T \in \mathcal{T}_5$ contains at least one donor.

Because every tile is a donor or has a donor as its neighbor, we can construct the scaffold as follows: First, construct the boundary of each donor. Afterwards, each donor can push out robots to neighboring recipients to construct their boundaries. By a careful analysis we can guarantee that the boundary of a recipient T only consists of robots that have their target position in T or in an adjacent tile. Overall we obtain the following.

► **Lemma 13.** *Constructing the scaffold takes no more than $O(d)$ transformations.*

A detailed description with a proof of this construction are described in the full version [20].

With the help of this global scaffold structure, connectivity is ensured during the actual reconfiguration. It remains to show how we shift robots between tiles, and reconfigure robot arrangements within the constructed scaffold. The latter has been already described in Section 4.3, so we describe how to relocate robots between tiles. This is modeled as a supply-demand flow for interior robots in three subphases: Phase (3.1) – Interior flow computation; Phase (3.2) – Interior flow partition; and Phase (3.3) – Interior flow realization. A similar approach is used to model the flow for boundary robots, see Section 4.9. We give descriptions of the different phases, and start with *Phase (3.1): Interior flow computation*.

4.6 Phase (3.1): Interior flow computation

Given any tile $T \in \mathcal{T}_5$, its interior robots either need to stay within it or move into a neighboring tile. The anticipated motion is represented as a supply-demand flow $G_{\mathcal{T}_5} := (\mathcal{T}_5, E_{\mathcal{T}_5}, f_{\mathcal{T}_5})$ of the dual graph of \mathcal{T}_5 . The flow value of an edge $f_{\mathcal{T}_5}(e)$ corresponds to the cardinality of the set of robots that need to move from one tile into another. A tile $T \in \mathcal{T}_5$ is a *source* (*sink*) if and only if the sum of flow values of incoming edges is smaller (larger) than the sum of flow values of outgoing edges. Otherwise, we call T *flow-conserving*. The difference of weights is called *supply* and *demand* for sources and sinks, respectively. If the flow value of every edge within a given flow graph is bounded from above by some value k , we refer to it as a k -*flow*. For simplicity, we consider the number of robots in the flow model, rather than the specific robots themselves. We observe that the flow value $f_{\mathcal{T}_5}(e)$ of each edge $e \in E_{\mathcal{T}_5}$ is bounded from above by the interior space of the tiles, as each may contain at most $(5cd - 2)^2 < 25c^2d^2$ robots.

We say that a schedule *realizes* a flow graph $G_{\mathcal{T}_5} = (\mathcal{T}_5, E_{\mathcal{T}_5}, f_{\mathcal{T}_5})$ if for each pair $v, w \in \mathcal{T}_5$ of tiles, the number of robots moved by it from their start tile v to their target tile w is $f_{\mathcal{T}_5}((v, w))$, where we let $f_{\mathcal{T}_5}((v, w)) = 0$ if $(v, w) \notin E_{\mathcal{T}_5}$. Additionally, we define an (a, b) -*partition* of a flow graph as a set that contains b many a -flows that sum up to the original

17:10 Reconfiguring a Connected Swarm of Labeled Robots

flow. By construction, the realization of a flow like the one above never requires us to fill a tile over capacity or to remove robots from an empty tile, as this would imply an invalid start or target configuration.

► **Lemma 14.** *It is possible to efficiently compute a stable schedule of makespan $O(d)$ that realizes $G_{\mathcal{T}_5}$.*

In the remainder of this section we provide an overview of the properties of $G_{\mathcal{T}_5}$ along with a detailed description of additional measures that split the graph into an acyclic and another totally cyclic component. For each of these components, we will provide an algorithm which computes a $(d, O(d))$ -partition of it. Finally, we discuss a set of unified movement patterns that realize each such flow partition through a schedule of makespan $O(d)$.

Note that the initial flow graph may contain diagonal, bidirectional, or crossing edges. By exchanging robots between neighboring tiles, we obtain a configuration with a planar, unidirectional flow graph $G_{\mathcal{T}_5}$. See the full version for details [20].

4.7 Phase (3.2): Interior flow partition

$G_{\mathcal{T}_5}$ is now a planar graph, but neither guaranteed to be acyclic nor totally cyclic. Due to the standard result from the theory of network flows (e.g., see [21, 24]) the algorithm partitions $G_{\mathcal{T}_5}$ into two flows G_{\rightarrow} and G_{\circ} , one being acyclic and the other totally cyclic.

► **Lemma 15.** *It is possible to efficiently compute a partition of $G_{\mathcal{T}_5}$ into an acyclic component G_{\rightarrow} and a totally cyclic component G_{\circ} .*

For the case of configurations that do not have to be connected, Demaine et al. [11] considered tiles of side length $24d$. Thus, they obtained a totally cyclic flow graph G_{\circ} with an upper bound of $24d \cdot 24d = 576d^2$ for the flow value of each edge. Furthermore, they showed that it is possible to compute a $(d, O(d))$ -partition of G_{\circ} . In our case, we have to keep configurations connected, resulting in tiles of side length cd . Thus, we extend the peeling algorithm from [11], resulting in a specific flow partition to a more general version.

► **Lemma 16.** *A $(d, O(d))$ -partition of the totally cyclic $k \cdot d^2$ -flow G_{\circ} for $k \in \mathbb{N}$ into totally cyclic flows can be computed efficiently.*

In the context of unlabeled robots, Fekete et al. [18] proposed an algorithm for computing a $(O(d^2), 28)$ -partition of G_{\rightarrow} . Due to the much more complex situation of labeled robots, we employ a number of more refined ideas to provide an algorithm that guarantees the following.

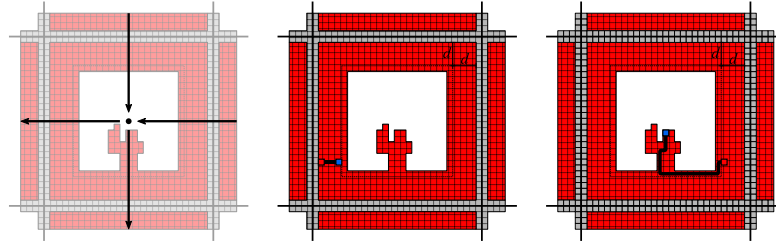
► **Lemma 17.** *A $(d, O(d))$ -partition of the acyclic $k \cdot d^2$ -flow G_{\rightarrow} for $k \in \mathbb{N}$ into acyclic flows can be computed efficiently.*

See the full version [20] for detailed proofs of above lemmas and the respective algorithms.

4.8 Phase (3.3): Interior flow realization

In order to exchange robots between tiles as modeled by the flow $G_{\mathcal{T}_5}$, we have to determine a collision-free protocol that allows robots to pass through the scaffold and into adjacent tiles. To this end, we describe a set of movement patterns for the robots of a single tile; these realize a single d -subflow in a stable manner within $O(d)$ steps. To achieve a compact concatenation of these movement patterns, an invariant type of local tile configuration is of significant importance. Using this invariant, we then provide more compact movement patterns that realize up to d such d -subflows via a schedule of makespan $O(d)$.

These invariant configuration of a tile $T \in \mathcal{T}_5$ are *push-stable* (with respect to a flow $G_{\mathcal{T}_5}$), defined by the following connectivity conditions for every robot r on an i th layer of the interior of T . If $i \leq d$ then r is connected to the closest boundary robot by a straight line of robots. Otherwise, r is connected to a robot r' on layer d by a path of robots in higher-order layers than d , such that the closest side to r' of the boundary of T rests on an edge without outgoing flow.



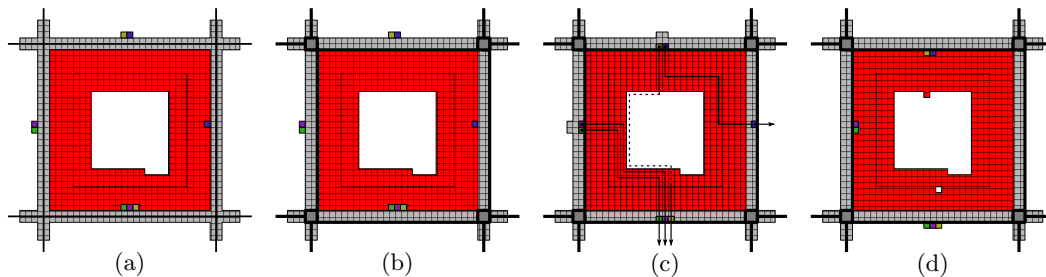
■ **Figure 5** An example of a push-stable configuration of a tile with two incoming and two outgoing edges (left). Highlighted are a robot on layer d (center) and on a higher-order layer (right), with paths that ensure their connectivity.

A *total sink* (*total source*) is a tile T that has four incoming (outgoing) edges of non-zero value over $G_{\mathcal{T}_5}$. Conversely, a *partial sink* (*partial source*) is a tile T that is not flow-conserving over $G_{\mathcal{T}_5}$, but has no more than three incoming (outgoing) edges of non-zero value. Note that by definition, total sources can never be configured in a push-stable manner. As a consequence, we handle both total sinks and total sources separately.

We briefly sketch our approach that realizes a single subflow.

► **Lemma 18.** *Consider a d -subflow $H_{\mathcal{T}_5} \subseteq G_{\mathcal{T}_5}$ and a tile $T \in \mathcal{T}_5$ that is flow-conserving with respect to $H_{\mathcal{T}_5}$. There is a schedule of makespan $O(d)$ that realizes the flow at its location.*

For details, we refer the reader to the full version. However, the high-level idea is to use Theorem 7 to construct a push-stable configuration and connect incoming and outgoing robots in a crossing-free manner by using higher-order layers, see Figure 6(c). We then move every robot along its respective path if it lies on layer at most d , or if the partial path from the incoming position to the robot contains no empty position; see the dashed lines in Figure 6(c). By a careful analysis we obtain the following lemma.



■ **Figure 6** A tile $T \in \mathcal{T}_5$ during the realization of a single d -subflow. The dashed portions of paths indicate that no motion actually occurs in this segment due to the described pushing behavior.

► **Lemma 19.** *The realization of a d -subflow can be performed in a way that results in another push-stable configuration of T .*

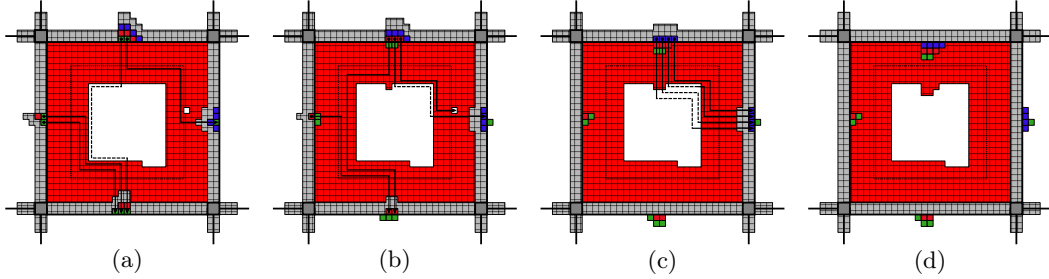
17:12 Reconfiguring a Connected Swarm of Labeled Robots

By slightly modifying this approach, mainly by leaving a specific set of positions empty (partial sources) or letting them become empty (partial sinks) during a sequence of parallel motions, this method becomes applicable for non-flow-conserving tiles.

► **Lemma 20.** *Consider a d -subflow $H_{\mathcal{T}_5} \subseteq G_{\mathcal{T}_5}$ and a tile $T \in \mathcal{T}_5$ that is not a total sink or total source with respect to $G_{\mathcal{T}_5}$. There is a schedule of makespan $O(d)$ that realizes the flow at its location and yields another push-stable configuration.*

The previously described movement patterns may be compactly combined into a single schedule of makespan $O(d)$ that realizes up to d many d -subflows at once. Intuitively, this can be achieved by stacking the outgoing robots against the target tile's boundary during the push-stable construction, see Figure 7. Because there are at most d subflows, these stacks of robots have height at most d . Therefore, we can successively realize each flow after another $O(1)$ steps per subflow.

► **Lemma 21.** *Consider a tile $T \in \mathcal{T}_5$ that is not a total sink or total source with respect to $G_{\mathcal{T}_5}$ and a sequence $S_{\mathcal{T}_5} := (H_{\mathcal{T}_5}^1, \dots, H_{\mathcal{T}_5}^\ell)$ of d -subflows of $G_{\mathcal{T}_5}$ with $\ell \leq d$. There exists a stable schedule of makespan $O(d)$ that realizes all ℓ many d -subflows.*



■ **Figure 7** A tile $T \in \mathcal{T}_5$ during the realization of ℓ d -subflows.

Total sinks and sources (as defined above) form a special case and are handled separately from the described push-stable patterns. Every total sink has at least as many empty positions as the number of incoming robots. By carefully rearranging the interior of the respective tile, we can realize each subflow by filling empty positions with robots. For the set of total sources, we consider the reverse operation.

► **Lemma 22.** *Consider a tile $T \in \mathcal{T}_5$ that is a total sink or total source with respect to $G_{\mathcal{T}_5}$ and a sequence $S_{\mathcal{T}_5} := (H_{\mathcal{T}_5}^1, \dots, H_{\mathcal{T}_5}^\ell)$ of d -subflows of $G_{\mathcal{T}_5}$ with $\ell \leq d$. There exists a stable schedule of makespan $O(d)$ that realizes all ℓ many d -subflows.*

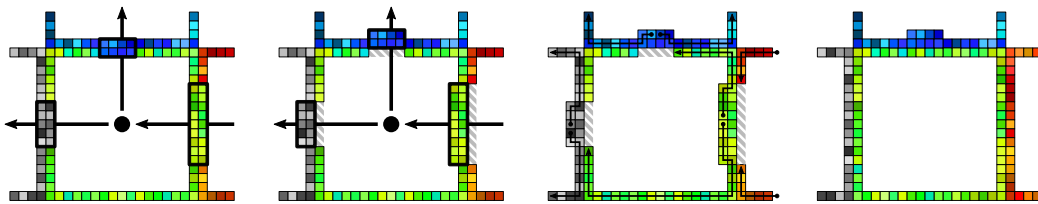
By applying Lemmas 21 and 22, up to d many d -subflows of $G_{\mathcal{T}_5}$ may be realized in $O(d)$ transformations. As we created $O(d)$ such subflows in Section 4.7, all of them may be realized through $O(d)/d = O(1)$ repetitions. These repetitions require a total of $O(d)$ transformations.

4.9 Phase (4): Boundary flow

The next phase of our algorithm deals with the movement of robots that are part of the scaffold. As described in Section 4.5, the robots forming the scaffold in a tiled configuration must remain in the 1-neighborhood of their target tile over \mathcal{T}_5 . The intermediate mapping step in the construction process guarantees that this remains the case even after that phase concludes. However, the scaffold does not necessarily consist of the same robots in the tiled configurations C'_s and C'_t . We observe that for any given tile $T \in \mathcal{T}_5$, up to $20cd - 4$ robots exist in its neighborhood $N_1[T]$ that need to become part of its boundary structure.

This observation induces another supply-demand flow graph similar to the one in Section 4.6, with each edge’s flow value bounded from above by $20cd - 4$. Using the techniques discussed in Sections 4.6 and 4.7, we obtain a $(d, O(1))$ -partition of this flow graph. This requires minor modifications to the involved methods. See full version [20] for details.

Each of the computed subflows can then be realized by a schedule of makespan $O(d)$. As no tile can ever end up with fewer than $20cd - 4$ total robots in the tiled target configuration, we can simply rearrange all tiles according to Theorem 7, placing outgoing robots adjacent to their target tile before pushing them into the boundary of their target tile and displacing part of the scaffold in the process. While this creates “gaps” in the boundary of all involved tiles, such movement never causes a disconnection in the full scaffold, as the receiving tile’s boundary preserves connectivity (see Figure 8). The resulting gap can thus be safely patched up with incoming robots. We repeat this process until each tile contains the correct robots for its scaffold.



■ **Figure 8** We can employ the displayed movement patterns to exchange boundary robots between adjacent tiles.

4.10 Phase (5): Local tile reconfiguration

Once Phase (4) concludes, we have reached a tiled configuration in which every tile contains precisely the robots that it would in a tiled configuration C'_t of the target configuration. This means that we can reconfigure into C'_t by a single application of Theorem 7, which forms the entirety of Phase (5).

As Phase (6) is a reverse of Phase (2), this concludes the description of the algorithm. Because each phase takes $O(d)$ transformation steps, this proves Theorem 4.

5 Conclusion and future work

We resolved two major open problems for connected reconfiguration. Some open problems still remain. Does any connected arrangement of n robots with diameter $D \leq n$ allow a makespan of $O(\sqrt{D})$? Can the involved constants of our methods be significantly reduced? Of interest are also implementations of largely distributed computation and motion control for efficient parallel motion schedules.

References

- 1 Aviv Adler, Mark de Berg, Dan Halperin, and Kiril Solovey. Efficient multi-robot motion planning for unlabeled discs in simple polygons. *Transactions on Automation Science and Engineering*, 12(4):1309–1317, 2015. doi:10.1109/TASE.2015.2470096.
- 2 Hugo A. Akitaya, Esther M. Arkin, Mirela Damian, Erik D. Demaine, Vida Dujmovic, Robin Y. Flatland, Matias Korman, Belén Palop, Irene Parada, André van Renssen, and Vera Sacristán. Universal reconfiguration of facet-connected modular robots by pivots: The $O(1)$ musketeers. *Algorithmica*, 83(5):1316–1351, 2021. doi:10.1007/s00453-020-00784-6.

- 3 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. In *Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, pages 4:1–4:19, 2022. doi:10.4230/LIPIcs.SWAT.2022.4.
- 4 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.
- 5 Julien Bourgeois, Sándor P. Fekete, Ramin Kosfeld, Peter Kramer, Benoît Piranda, Christian Rieck, and Christian Scheffer. Space Ants: Episode II – Coordinating Connected Catoms. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:6, 2022. doi:10.4230/LIPIcs.SoCG.2022.65.
- 6 Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008. doi:10.1137/060652063.
- 7 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.
- 8 Loïc Crombez, Guilherme Dias da Fonseca, Yan Gerard, Aldo Gonzalez-Lorenzo, Pascal Lafourcade, and Luc Libralesso. Shadoks approach to low-makespan coordinated motion planning. In *Symposium on Computational Geometry (SoCG)*, pages 63:1–63:9, 2021. doi:10.4230/LIPIcs.SoCG.2021.63.
- 9 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. Springer, 2014. doi:10.1007/978-4-431-54475-3_12.
- 10 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.
- 11 Erik D. Demaine, Sándor P. Fekete, Phillip Keldenich, Christian Scheffer, and Henk Meijer. 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.
- 12 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.
- 13 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 (STACS)*, pages 201–212, 2011. doi:10.4230/LIPIcs.STACS.2011.201.
- 14 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22(1):37–50, 2006. doi:10.1007/s00373-005-0640-1.
- 15 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Formations for fast locomotion of metamorphic robotic systems. *International Journal of Robotics Research*, 23(6):583–593, 2004. doi:10.1177/0278364904039652.
- 16 Adrian Dumitrescu, Ichiro Suzuki, and Masafumi Yamashita. Motion planning for metamorphic systems: feasibility, decidability, and distributed reconfiguration. *Transactions on Robotics*, 20(3):409–418, 2004. doi:10.1109/TRA.2004.824936.
- 17 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*. Springer, 2011. doi:10.1007/978-3-0348-0130-0_29.
- 18 Sándor P. Fekete, Phillip Keldenich, Ramin Kosfeld, Christian Rieck, and Christian Scheffer. Connected coordinated motion planning with bounded stretch. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 9:1–9:16, 2021. doi:10.4230/LIPIcs.ISAAC.2021.9.

- 19 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](#).
- 20 Sándor P. Fekete, Peter Kramer, Christian Rieck, Christian Scheffer, and Arne Schmidt. Efficiently reconfiguring a connected swarm of labeled robots, 2022. [arXiv:2209.11028](#).
- 21 Lester R. Ford and Delbert R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.
- 22 Seth C. Goldstein and Todd C. Mowry. Claytronics: A scalable basis for future robots, 2004. URL: <http://www.cs.cmu.edu/~claytronics/papers/goldstein-robosphere04.pdf>.
- 23 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.
- 24 Bernhard H. Korte and Jens Vygen. *Combinatorial optimization*. Springer, 2011.
- 25 Paul Liu, Jack Spalding-Jamieson, Brandon Zhang, and Da Wei Zheng. Coordinated motion planning through randomized k-opt. In *Symposium on Computational Geometry (SoCG)*, pages 64:1–64:8, 2021. doi:10.4230/LIPIcs.SoCG.2021.64.
- 26 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.
- 27 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.
- 28 Erol Şahin and Alan F. T. Winfield. Special issue on swarm robotics. *Swarm Intelligence*, 2(2-4):69–72, 2008. doi:10.1007/s11721-008-0020-6.
- 29 Michael Schreckenberg and Reinhard Selten. *Human Behaviour and Traffic Networks*. Springer, 2013. doi:10.1007/978-3-662-07809-9.
- 30 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.
- 31 David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007. doi:10.1137/S0097539704446712.
- 32 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.
- 33 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.
- 34 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.
- 35 Matthew Turpin, Nathan Michael, and Vijay Kumar. Trajectory planning and assignment in multirobot systems. In *Algorithmic Foundations of Robotics X – Workshop on the Algorithmic Foundations of Robotics (WAFR)*, pages 175–190. Springer, 2013. doi:10.1007/978-3-642-36279-8_11.
- 36 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.
- 37 Hyeyun Yang and Antoine Vigneron. A simulated annealing approach to coordinated motion planning. In *Symposium on Computational Geometry (SoCG)*, pages 65:1–65:9, 2021. doi:10.4230/LIPIcs.SoCG.2021.65.