# Forming tile shapes with simple robots

# Forming Tile Shapes with Simple Robots *

Robert Gmyr[†] Kristian Hinnenthal[‡] Irina Kostitsyna[§] Fabian Kuhn[¶] Dorian Rudolph[‡]
Christian Scheideler[‡] Thim Strothmann[‡]

**Abstract**

Motivated by the problem of manipulating nanoscale materials, we investigate the problem of reconfiguring a set of tiles into certain shapes by robots with limited computational capabilities. As a first step towards developing a general framework for these problems, we consider the problem of rearranging a connected set of hexagonal tiles by a single deterministic finite automaton. After investigating some limitations of a single-robot system, we show that a feasible approach to build a particular shape is to first rearrange the tiles into an intermediate structure by performing very simple tile movements. We introduce three types of such intermediate structures, each having certain advantages and disadvantages. Each of these structures can be built in asymptotically optimal $O(n^2)$ rounds, where $n$ is the number of tiles. As a proof of concept, we give an algorithm for reconfiguring a set of tiles into a triangle through one of the intermediate structures.

## 1 Introduction

Constructing shapes by nanoscale matter is a very natural and popular problem that has been investigated under various models already. For example, in the DNA computing field a prominent approach is to use DNA tiles [1]. Whereas this approach is based on strictly passive elements, so any changes to the structure have to be enforced externally (e.g., by changing the temperature or exposing the structure to certain kinds of radiation), several models have been proposed that consider active elements instead [2, 3, 4]. In our work, we investigate a *hybrid approach* for the *shape formation problem*, in which we are given a set of *passive* tiles that are uniform and stateless, and (a limited number of) *active* robots. The robots, which only have the computational power of a finite automaton, can transport tiles from one position to another in order to form a desired shape. Compared to the DNA tile-based approach, this approach has the advantage that the tiles no longer have to be tailored to specific shapes and that no external control is needed anymore. Furthermore, in contrast to the approaches based entirely on active elements, we believe that many problems can be solved in our hybrid model using only a small number of active elements. Therefore, considering practical application, our model should presumably be more cost-efficient to realize. In this paper we support this claim by showing that already a single robot is able to solve simple shape formation problems. Our ultimate goal is to investigate how multiple robots can cooperate to speed up the process of shape formation, which is still ongoing work.

## 2 Model and Problem Statement

We assume that a single *active* agent (a *robot*) operates on a set of $n$ *passive* hexagonal *tiles*. Each tile occupies exactly one node of the infinite triangular lattice $G = (V, E)$ (see Figure 1). We assume that the initial position of the robot is occupied by a tile and describe the relative positions of adjacent nodes of the robot's position by six compass directions. Note that even though we describe the algorithms as if the robot knew its global orientation, we do not actually require the robot to have a compass.

*This work was begun at the Dagstuhl Seminar on Algorithmic Foundations of Programmable Matter, July 3–8, 2016. Preliminary results were presented at EuroCG 2017, and the full version of the paper has been submitted to DNA 2018. This work is partly supported by DFG grant SCHE 1592/3-1. Fabian Kuhn is supported by ERC Grant 336495 (ACDC).

[†]University of Houston, USA. rgmyr@uh.edu.

[‡]Paderborn University, Germany. {krijan, dorian, scheidel, thim}@mail.upb.de.

[§]TU Eindhoven, the Netherlands. i.kostitsyna@tue.nl.

[¶]University of Freiburg, Germany. kuhn@cs.uni-freiburg.de.

Whereas tiles cannot perform any computation nor move on their own, the robot may change its position and carry a tile, thereby modify the tile structure. We require that the robot's position is always adjacent to a node occupied by a tile. Additionally, if the robot does not carry a tile, we require the subgraph of $G$ induced by the occupied nodes to be connected; otherwise, the subgraph induced by occupied nodes and the robot's position must be connected. In a scenario where a tile structure floats in a liquid, for example, this restriction prevents the robot or parts of the tile structure from floating apart.

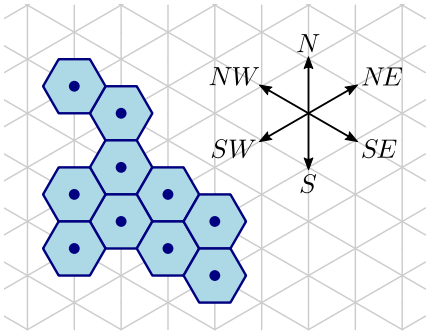


**Figure 1:** A set of tiles placed on nodes of the infinite triangular lattice. The top right part of the figure shows the compass directions we use to describe the movement of a robot.

The robot acts as a *deterministic finite automaton* and operates in rounds of *look-compute-move* cycles. In the *look* phase of a round the robot can observe its current node and the six neighbors of that node. For each of these nodes it can determine whether it is occupied or not. In the *compute* phase the robot potentially changes its state and determines its next move according to the observed information. In the *move* phase the robot can either (1) pick up a tile from its current node, if its current node is occupied, (2) place a tile it is carrying at that node, if it is not occupied, or (3) move to an adjacent node while possibly carrying a tile with it. The robot can carry at most one tile.

We are interested in *Shape Formation* problems, in which the goal is to transform any initial configuration into a configuration in which the tiles form a certain shape on the lattice.

## 3 Forming an Intermediate Structure

In a naive approach to shape formation, the robot could iteratively search for a tile that can be removed without disconnecting the tile structure and then move that tile to some position such that the shape under construction is extended. While there always is a safely removable tile, in the full paper we show that a single robot cannot find it, which makes this approach infeasible.

Instead, we propose to perform simple local tile movements that preserve connectivity, and to use such movements to first rearrange the tile structure into an *intermediate structure*. In the following we show how to construct three different intermediate structures, each having certain advantages and disadvantages. In each of the resulting intermediate structures the robot can easily navigate and remove tiles in order to finally rearrange it into the desired shape. The correctness and runtime proofs can be found in the full paper.

**Forming a Line** We first present an algorithm to rearrange a given tile configuration into a straight line in $O(n^2)$ rounds. We use the labels $N$, $NE$, $SE$, $S$, $SW$, and $NW$ (corresponding to cardinal directions) to refer to the six neighbors of the robot (see Figure 1).

First, the robot moves $S$ as far as possible, i.e., as long as there is a tile in direction $S$. Then, it alternates between a *tile searching phase*, in which it moves $N$, $NW$, and $SW$ (in that precedence) until there is no tile in any of these directions anymore; and a *tile moving phase*, in which it picks up the tile, moves one step $SE$, then $S$ until it reaches an empty node, and finally places the tile there. The line has been built once the robot does not encounter any adjacent western or eastern tiles in the tile searching phase, which can easily be detected. An example of the first several steps of the algorithm can be found in Figure 2a.

**Forming a Block** Although a line can be constructed efficiently, its linear diameter (which is defined as the maximal length of a shortest path between any two occupied nodes of the triangular lattice) might make it an undesirable intermediate structure. Our second algorithm constructs a *block* in time $O(nD)$ and ensures that no tile is ever moved farther than by a distance of $D$, where $D$ is the initial structure's diameter. A block is a structure in which all tiles except for the globally westernmost tiles have a neighbor at $NW$ (see Figure 2b).

Similar to the algorithm to form a line, the robot again alternates between a searching and a moving phase. As before, in the searching phase, the robot finds a tile to pick up by moving $NW$, $SW$, and $N$ (in that

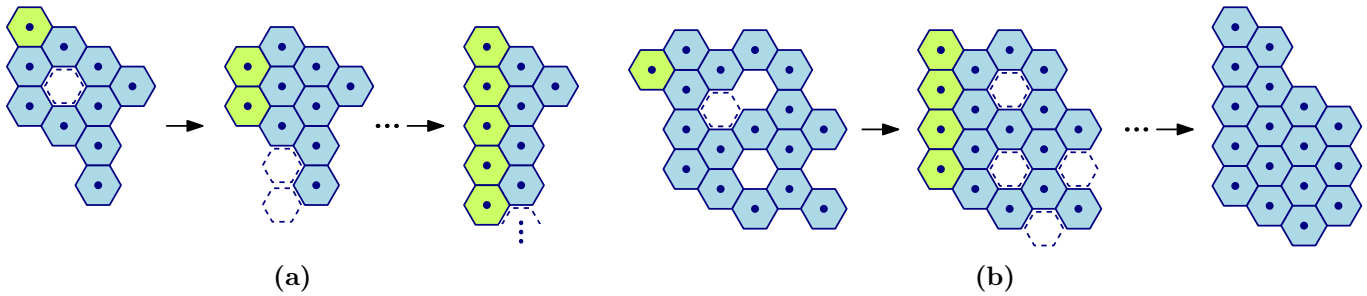


**Figure 2:** (a) First several steps of the line formation. The green tiles are moved to the positions marked by dashed frames. (b) Some steps of the block formation.

precedence) as far as possible. Then, in the moving phase, the robot picks up the tile, moves $SE$ until it reaches an empty node, and places the tile there. Some steps of the algorithm are shown in Figure 2b.

Although this simple algorithm already constructs a block, detecting once it has been built is still a bit more complicated. This is done by performing a series of tests alongside the algorithm's execution. Consider a block as a stack of *rows*, i.e., sequences of consecutive tiles from $NW$ to $SE$. Note that according to the above algorithm the robot will move each tile of the westernmost column of a finished block, starting with the northernmost tile, placing each at the first empty position $SE$ of it. Thereby, the robot detects that a block has been built by verifying the following conditions: (1) after placing a tile, the robot performs at most one $SW$ movement before it takes the next tile, (2) while moving a tile $t$, the robot does not traverse a node (except for $t$'s previous position) that has a neighbor at $NE$, but not at $N$, or a neighbor at $S$, but not at $SW$, (3) the robot never places a tile at a node that has an adjacent tile at $SE$. A test verifying the above conditions is initiated whenever the robot picks a tile that does not have a $NE$ neighbor. If thereafter any of the above conditions gets violated, the test is aborted. If otherwise the robot places the southernmost tile without encountering any violation, the algorithm terminates.

**Forming a Tree** Whereas the previous two algorithms focus on how to *quickly* construct suitable intermediate structures, regarding practical application of the algorithms, it may also be desirable to minimize the required work space. In fact, both the line and the block are in many cases built almost completely outside of the initial configuration's convex hull (where we refer to the convex hull of the corresponding set of hexagonal tiles in the Euclidean plane). We present an algorithm that builds *tree* in time $O(n^2)$ by exclusively moving tiles inside the structure's convex hull. A tree is a connected tile configuration without an *overhang*, i.e., a set of vertically adjacent empty nodes such that (1) the northernmost node has a tile at $N$, (2) the southernmost node has a tile at $S$, and (3) all nodes have adjacent tiles at $NW$ and $SW$. Examples of an overhang and a tree can be found in Figures 3a and 3b, respectively.

Due to space constraints, we only sketch the algorithm from a high level. The details can be found in the full version of the paper. First, the robot traverses the columns of the tile structure (i.e., consecutive connected tiles from $N$ to $S$) in a recursive fashion until it encounters an overhang. Starting at a locally easternmost column, the robot successively moves to the northernmost of the current column's western neighbors, and continues to do so until it reaches a locally westernmost column. In this case, the robot checks whether the current column has an adjacent eastern overhang by traversing the column from $N$ to $S$. If so, it fills the overhang as described in the next paragraph and afterwards restarts the algorithm. Otherwise, the robot moves to the current column's adjacent eastern column (of which there can be at most one). Then, it moves south and continues its recursive search in the next western column. If no such column exists, the robot verifies whether the current column has an adjacent eastern overhang and proceeds as described above. Eventually, the robot will reach a column without an adjacent eastern column, in which case it terminates.

We now describe how the robot fills an overhang. First, to find a tile to place into the overhang, the robot moves in a way that assures that it will find its way back (see Figure 3c). The robot alternates between moving $N$ as long as there is a tile at $N$ (`get_tile_N` phase), and moving $NW$ as long as there is a tile at $NW$ and no

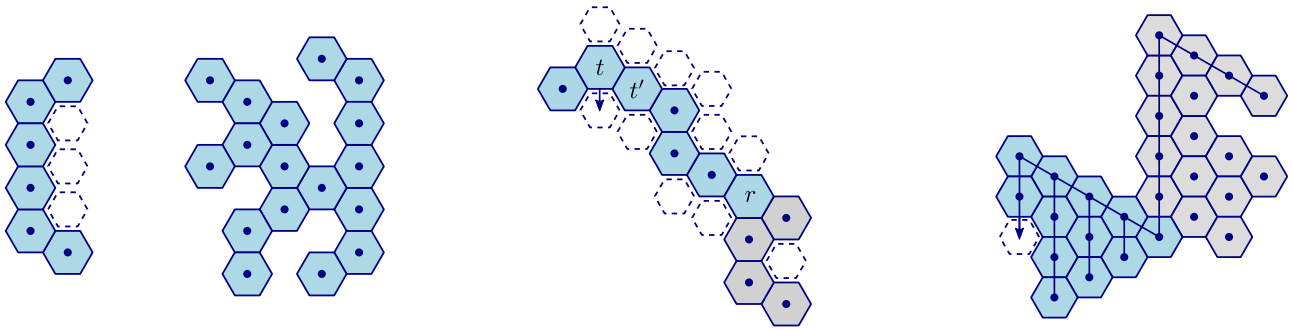


**Figure 3:** (a) An overhang. (b) A tree. (c) The search path of the robot starting at $r$ to fill an overhang (gray tiles). All positions marked by dashed frames must be empty. (d) Triangle formation (in blue) starting from a block (in gray).

tile at $SW$ or $N$ (`get_tile_NW` phase). The robot's path either ends in the `get_tile_N` phase at a tile that does not have a neighbor at $NW$ or $SW$, in which case it is picked up, or in the `get_tile_NW` phase at a locally most north-western tile, which would also be taken, or at a tile $t$ that has a $SW$ neighbor. In the latter case, which is depicted in the figure, $t$ is moved one step $S$. If thereafter $t$ has a neighbor at $S$ or $SW$, the robot takes $t$'s $NE$ neighbor $t'$. Otherwise, it moves onto $t'$ and continues its search. Once the robot has picked up a tile, it returns to its originating column by moving $S$ or $SE$ (in this order of precedence), until it reaches the overhang. The robot continues to bring tiles as described until the overhang is filled, in which case it restarts the algorithm.

## 4 Forming a Triangle

All presented intermediate structures can easily be transformed into simple shapes such as a triangle, a hexagon, or a parallelogram. For example, starting from a block, the robot can build a triangle in an additional $O(nD)$ rounds by repeatedly taking the last tile of the northernmost row of the block, carrying it to the bottom of the westernmost column, and placing it at the next position of the westernmost *layer* of the triangle, see Figure 3d. More specifically, the robot first creates the tip of the triangle by placing the first tile below the westernmost column of the block. The second tile is placed $NW$ of the tip. Every other tile of the triangle is then placed as follows. The robot first brings a tile to the triangle's tip. It then walks $NW$ and $S$ (in that precedence) until there is no tile in these directions anymore. If there is a tile at $SE$, the robot moves one step $S$ and drops the tile. Otherwise, the robot moves to the top of the layer, takes one step in $NW$ direction and places the tile there. In this manner, the robot continues to extend the triangle tile by tile until the whole block has been disassembled.

## References

[1] Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

[2] Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal shape formation for programmable matter. In *Proc. 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.

[3] Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proc. 4th Conference of Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013.

[4] Ferran Hurtado, Enrique Molina, Suneeta Ramaswami, and Vera Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.