Forming tile shapes with a single robot

Forming Tile Shapes with a Single Robot*

Robert Gmyr† Irina Kostitsyna‡ Fabian Kuhn§ Christian Scheideler† Thim Strothmann †

1 Introduction

We investigate the problem of *shape formation* with *robots* on *tiles* in which a collection of robots has to rearrange a set of movable tiles to form a desired shape. In this preliminary work we consider the case of a single robot operating on an arbitrary number of tiles and present first results towards the formation of simple shapes. Our ultimate goal is to investigate how multiple robots can cooperate to speed up the process of shape formation.

Model. We consider a single *robot* acting on a finite set of *hexagonal tiles*. The tiles are *passive*, i.e., they do not perform any computation and cannot move on their own. The tiles may form any structure so that their centers coincide with nodes of a triangular grid graph, as shown in Figure 1, and there is at most one tile per node.

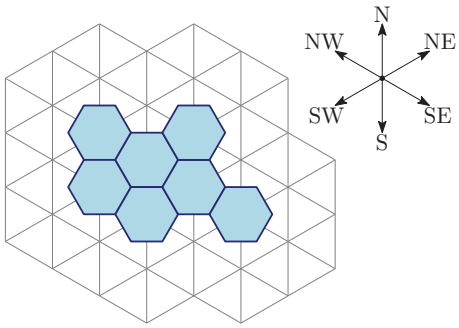

Figure 1: An example tile configuration. The top right part of the figure shows the compass directions we use to describe the movement of the robot.

The robot is *active* and may occupy any node of the grid graph. It is a *deterministic finite automaton* that operates in *look-compute-move* cycles. In the *look* phase the robot can observe the node it occupies and the six neighbors of that node. For each of these nodes it can determine whether there is a tile placed at that node. In the *compute* phase the robot can use this information together with its state to determine its next move and to change its state. In the *move* phase the robot can take a tile from its current node, place a tile it is carrying at that node, or move to an adjacent node while possibly carrying a tile with it. The robot can carry at most one tile.

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. For the algorithms presented in this paper, it is enough for the robot to be able to maintain its relative orientation with respect to its original orientation.

Problem Statement. A *configuration* consists of the positions (i.e., the occupied nodes) of the tiles and the position and state of the robot. We define a configuration to be *connected* if the subgraph induced by the nodes that are occupied by the tiles (including a tile carried by the robot) is connected. In the *triangle formation problem* we are given an arbitrary connected configuration in an infinite grid graph with a robot in the initial state and the goal is to rearrange the tiles into an equilateral triangle with the help of the robot while having a connected configuration at the beginning of every look-compute-move cycle.

We aim at maintaining connectivity of the tile structure. Consider a scenario where a tile structure floats in a liquid. Connected components of a disconnected structure might float apart. Thus, we want our techniques to be applicable to scenarios where it is important to maintain fixed tile positions (relative to each other). To be applicable also to nano-systems, we assume the robot just to have the computational power of a finite automaton.

Note that the requirement for connectivity effectively restricts the movement of the robot but there are no restrictions concerning the grid graph since it is assumed to be infinite in every direction. Also note that if the number of tiles is not a triangular number, one side of the triangle is only partially occupied by tiles.

Related Work. There is a number of approaches to shape formation in the literature that use agents that fall somewhere in the spectrum between passive and active. For example, *tile-based self-assembly* [8] uses passive tiles that bond to each other to form shapes. A variant of *population protocols* proposed in [7] uses agents that are partly passive (i.e., they cannot control their movement) and partly active (i.e., upon meeting another, they can perform a computation and decide whether they want to form a bond). Fi-

*This work was begun at the Dagstuhl Seminar on Algorithmic Foundations of Programmable Matter, July 3–8, 2016.

†University of Paderborn, Germany
‡Université libre de Bruxelles (ULB), Brussels, Belgium
§University of Freiburg, Germany

nally, the *amoebot model* [4], the *nubot model* [11], and the modular robotic model proposed in [6] use agents that are completely active in that they can compute and control their movement. All of these approaches have in common that they consider a *single type* of agent. In contrast, we investigate a model that uses a *combination* of *active* and *passive* agents.

When arguing about a robot that traverses a tile structure without moving tiles, our model essentially reduces to an instance of the ubiquitous *agents on graphs* model. The vast amount of research on this model covers many interesting problems such as Gathering and Rendezvous (e.g. [9]), Intruder Caption and Graph Searching (e.g. [1, 5]), and Graph Exploration (e.g. [2]). Some approaches are also known that allow agents to move tiles (e.g. [3, 10]) but these focus on computational complexity issues or agents that are more powerful than finite automata.

2 A Naive Approach

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 tile that can be safely removed, the following theorem shows that, in general, the robot cannot find it, which makes this naive approach infeasible.

Theorem 1 *The robot cannot find a tile that can be removed without disconnecting the tile structure.*

Proof. Suppose that there is an algorithm that allows the robot to find such a tile. Let s be the number of states used by the algorithm. Consider the execution of the algorithm on a hollow hexagon of side length ℓ where the robot is initially placed on a vertex of the hexagon as depicted in the left part of Figure 2. We subdivide the execution into *phases* where we define a new phase to start whenever the robot visits a vertex of the hexagon. Note that the algorithm runs for at most $6s$ phases before the robot chooses the tile because if it would run for more phases the robot would visit the same vertex twice in the same state and therefore the algorithm would enter an infinite loop.

The way the robot traverses the hexagon depends on the side length ℓ. We define the *traversal sequence* associated with ℓ as $((v_1, q_1), (v_2, q_2), \ldots, (v_k, q_k))$ where k is the number of phases the algorithm takes until the tile is chosen, v_i is the vertex occupied by the robot at the beginning of phase i, and q_i is the state of the robot at the beginning of phase i. Since the algorithm takes at most $6s$ phases to choose the tile (independently of ℓ), there are at most $(6s)^{6s}$ distinct traversal sequences. Hence, there is a finite number of traversal sequences and an infinite number of side lengths which implies, according to the pigeonhole principle, that there must be an infinite set of side lengths L that have the same traversal sequence.

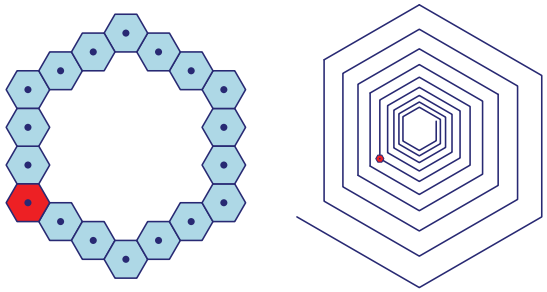

Figure 2: Left: The hollow hexagon of side length $\ell = 4$. Right: An example of the tile structure S. The red mark represents the initial position of the robot.

Based on this observation, we now define a tile structure S for which the algorithm fails to find a tile that can be safely removed. This tile structure essentially consists of a spiral as depicted in the right part of Figure 2. We start at an arbitrary node of the triangular grid graph and construct an outward spiral consisting of $24s$ line segments. The first line segment of the spiral goes north and each following line segment takes a $60°$ clockwise turn. The lengths of the line segments are chosen from L in such a way that the segments stay separated. This is possible since L is an infinite set and therefore we can always choose sufficiently large segment lengths. We initially place the robot at the end of the $12s$-th line segment.

It remains to show that the algorithm fails to find a tile that can be safely removed when being executed on S. As above, we subdivide the execution of the algorithm into phases where we define a new phase to start whenever the robot visits a vertex of the spiral (i.e., a tile where two line segments meet). It is easy to show using induction on the phases that the robot traverses S in a way that corresponds to the traversal sequence associated with the side lengths in L. Consequently, the robot chooses a tile that is neither the start tile nor the end tile of the spiral. Since these two tiles are the only tiles that can be safely removed from S, the algorithm fails. This contradicts the assumption that the algorithm works correctly and therefore shows that there is no such algorithm. □

3 Taking a Detour via a Line

We now present an approach that avoids the pitfall of the naive approach by first rearranging the tiles into a straight line. Whenever a tile that cannot be removed without disconnecting the structure is picked up by the robot during this process, the tile is placed at a neighboring node in a way that preserves connectivity.

Algorithm 1 Algorithm to form a straight line from any tile configuration by a single robot.

1: **procedure** MAKELINE
2: The robot moves S until there is no tile to step on.
3: **do**
4: Set flag *is_line* to TRUE.
5: *Tile searching phase:* at every step, until the robot can no longer move,
6: – if there is a neighboring tile at NW, SW, NE, or SE, set the flag *is_line* to FALSE;
7: – the robot repeatedly moves NW, SW or N (in this order of preference).
8: *Tile moving phase:* if *is_line* is FALSE, the robot picks up the tile at the current position, and moves it to the bottom of the adjacent column, starting at position SE.
9: **while** *is_line* is FALSE

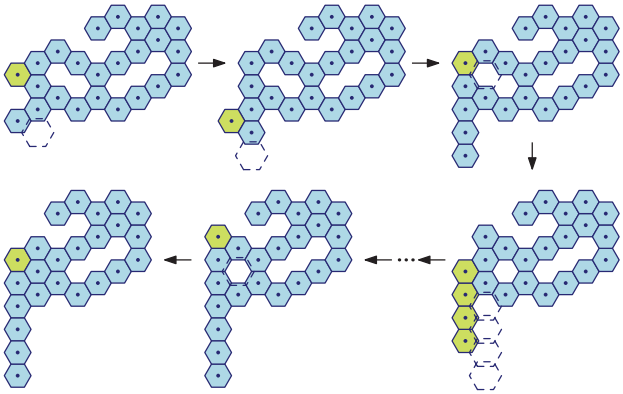

Figure 3: First several steps of the algorithm. The green tiles are moved to the positions marked by dashed frames.

3.1 Line Formation

We consider the problem of rearranging the tiles into a straight line. We will measure the efficiency of our algorithm in the number of *steps* (i.e., move actions) that the robot has to perform.

We present an algorithm for one robot to rearrange a tile configuration into a straight line in $O(n^2)$ steps. Throughout the algorithm 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). The pseudocode is given in Algorithm 1. At the beginning of every iteration of the algorithm, the robot is located at a locally most southern tile, i.e., there is no tile in the S direction. During one iteration of the algorithm, the robot finds a locally most north-western tile and moves it to the bottom of the column of tiles to the right from it. Figure 3 illustrates the first several iterations of the algorithm. To check whether the desired tile configuration has been achieved, the robot inspects neighboring tiles at each step in the search phase.

Theorem 2 *Following the procedure* MAKELINE*, a single robot can rearrange any tile configuration into a straight line in $O(n^2)$ steps.*

Proof. The correctness of the algorithm follows from the following observations: (i) the tile searching phase terminates in a locally most north-western tile, (ii) if there is more than one column in the tile configuration, the tile searching phase does not terminate in the topmost tile of the rightmost column, (iii) the tile moving phase does not disconnect the tile configuration and (iv) the algorithm terminates when a line is formed.

The first observation is obvious by the definition of the first phase of the algorithm. The second observation follows from the fact that the preference is given to the NW and SW directions when searching. If the target tile configuration has not yet been achieved, and the robot stops at some locally north-western tile, there must be tiles to the right from that position.

For the third observation, suppose that the tile moving phase disconnects the tile configuration. Let t be the locally most north-western tile being moved. The tile configuration can get disconnected after removing t only if there are neighboring tiles to NE and S of t, but no SE neighboring tile, since otherwise the neighboring tiles will still be locally connected after removing t. But in that case the tile t will be placed in the empty position at the SE neighbor and reconnect the neighboring NE and S tiles. Therefore, during the second phase of the algorithm the tile configuration does not get disconnected.

To prove the last claim, assign 2-dimensional coordinates to the centers of tiles. Let the x coordinate grow from left to right, and the y coordinate grow from top to bottom. Let 0 be the x-coordinate of a rightmost tile, thus the x-coordinate of any tile is not greater than 0. Consider the sum of the x-coordinates of all tiles $S = \sum_1^n x_t$. Initially, the value of S is negative, and it always increases by 1 after a tile is moved. The tile configuration is a straight line at $x = 0$, i.e., $S = 0$. No tiles will be moved to a position with an x-coordinate larger than 0. Therefore, the algorithm will terminate, and the terminal tile configuration will be a vertical straight line.

Finally, we show that the algorithm takes $O(n^2)$ steps. The preparation steps of the algorithm (line 2 of the Algorithm) take $O(n)$ steps. Consider the tile moving phase (line 8). Let the initial coordinates of some tile t be $(x_{t,0}, y_{t,0})$, and its final coordinates be $(0, y_{t,1})$. Each time the tile was moved, its coordinates were changed from some (x_t, y_t) to $(x_t + 1, y_t + \frac{1}{2} + c_t)$, where c_t is the number of tiles in the column, at the bottom of which the tile t was placed. The total number of steps the robot performed to move the tile from (x_t, y_t) to $(x_t + 1, y_t + \frac{1}{2} + c_t)$ is $1 + c_t$. Therefore, the total number of steps the robot performed to move the tile from its initial position to its final placement,

is $0-x_{t,0}+y_{t,1}-y_{t,0}-\frac{1}{2}(0-x_{t,0}) = -\frac{x_{t,0}}{2}+(y_{t,1}-y_{t,0})$. And the total number of steps the robot performed to move all the tiles is $s_{move} = \sum_t \left(-\frac{x_{t,0}}{2}+(y_{t,1}-y_{t,0})\right) \leq \sum_t \frac{3}{2}n = O(n^2)$. Now, consider the tile searching phase (lines 5–7). Whereas the sum of the coordinates of the robot was increasing at every step in the tile moving phase, in the tile searching phase, the sum of the coordinates of the robot is decreasing at every step. More specifically, at each step of the tile moving phase, the sum of the coordinates of the robot increases by at most $\frac{3}{2}$, and at every step of the tile searching phase, the sum of the coordinates decreases by at least $\frac{1}{2}$. Thus, the total number of steps in the tile searching phase can be bounded in the following way: $s_{search} < 3 \times s_{move} + (x_0 + y_0) - \min_i(x_i + y_i)$, where (x_0, y_0) is the initial coordinates of the robot, and the value $\min(x_i+y_i)$ is taken over all possible placements of all tiles. As the initial tile configuration is connected, $(x_0 + y_0) - \min_i(x_i + y_i) = O(n)$, and $s_{search} = O(n^2)$. Therefore, the total number of steps is $O(n^2)$. □

Note that it is not hard to see that $\Omega(n^2)$ steps are necessary to rearrange an arbitrary initial tile configuration into a straight line. If starting from a initial configuration with diameter $O(\sqrt{n})$, a constant fraction of the tiles have to be moved by a distance linear in n and thus, in total, $\Omega(n^2)$ move steps are necessary.

3.2 Triangle Formation

Once the robot has built a line, it can construct a triangle as follows: the robot picks up tiles from one end of the line and assembles them into a triangle at the other end following a zig-zag pattern, see Figure 4. It fills each *layer* of the triangle with tiles until it

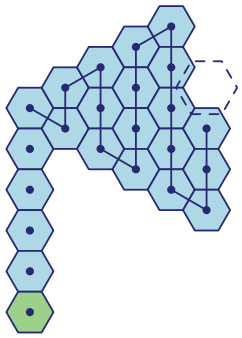

Figure 4: Triangle formation starting from a line

recognizes that the current position does not have a tile in the NW direction (when moving up), or that the current position does not have a tile in the SW direction (when moving down). In both cases, it starts a new layer of the triangle arrangement.

Theorem 3 *A single robot can rearrange any tile configuration into a triangle in $O(n^2)$ steps.*

It is not hard to see, using similar arguments as in the previous section, that this is asymptotically optimal.

4 Future Work

There are many directions for further research on shape formation with robots on tiles. First, we would be very interested to see how multiple robots can cooperate to speed up shape formation. Another obvious direction would be the formation of more complex shapes. Finally, it might be interesting to study an extension of the model in which each tile can have a state that can be read and modified by the robot.

Acknowledgments. This work was partially supported by DFG grant SCHE 1592/3-1. Irina Kostitsyna is supported by F.R.S.-FNRS and Fabian Kuhn is supported by ERC Grant No. 336495 (ACDC).

References

[1] A. Bonato and R. J. Nowakowski. *The Game of Cops and Robbers on Graphs*. AMS, 2011.

[2] S. Das. Mobile agents in distributed computing: Network exploration. *Bulletin of the European Association for Theoretical Computer Science*, 109:54–69, 2013.

[3] E. Demaine, M. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *Computational Geometry*, 26(1):21–36, 2003.

[4] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *28th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016.

[5] F. V. Fomin and D. M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.

[6] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguraiton of 2D lattice-based modular robotic systems. *Autonomous Robots*, 38(4):383–413, 2015.

[7] O. Michail and P. G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81-82:1–10, 2015.

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

[9] A. Pelc. Deterministic rendezvous in networks: A comprehensive survey. *Networks*, 59(3):331–347, 2012.

[10] Y. Terada and S. Murata. Automatic modular assembly system and its distributed control. *International Journal of Robotics Research*, 27(3–4):445–462, 2008.

[11] D. Woods, H. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Innovations in Theoretical Computer Science (ITCS)*, 2013.