# Brief Announcement: Collision Detection for Modular Robots – It Is Easy to Cause Collisions and Hard to Avoid Them

**Siddharth Gupta**
BITS Pilani, Goa Campus, India

**Marc van Kreveld**
Utrecht University, The Netherlands

**Othon Michail**
University of Liverpool, United Kingdom

**Andreas Padalkin**
Paderborn University, Germany

## Abstract

We consider geometric collision-detection problems for modular reconfigurable robots. Assuming the nodes (modules) are connected squares on a grid, we investigate the complexity of deciding whether collisions may occur, or can be avoided, if a set of expansion and contraction operations is executed. We study both discrete- and continuous-time models, and allow operations to be coupled into a single parallel group. Our algorithms to decide if a collision may occur run in $O(n^2 \log^2 n)$ time, $O(n^2)$ time, or $O(n \log^2 n)$ time, depending on the presence and type of coupled operations, in a continuous-time model for a modular robot with $n$ nodes. To decide if collisions can be avoided, we show that a very restricted version is already NP-complete in the discrete-time model, while the same problem is polynomial in the continuous-time model. A less restricted version is NP-hard in the continuous-time model.

**2012 ACM Subject Classification** Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

**Keywords and phrases** Modular robots, Collision detection, Computational Geometry, Complexity

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2024.26

**Related Version** *Full Version*: `https://arxiv.org/abs/2305.01015` [6]

**Funding** *Andreas Padalkin*: This author was supported by the DFG Project SCHE 1592/10-1.

**Acknowledgements** The authors thank all participants of the Bertinoro Workshop on Distributed Geometric Algorithms, in particular Peyman Afshani for suggesting the $O(n \log^2 n)$ time solution for detecting collisions when there are no couplings. We thank Irina Kostitsyna and Christian Scheideler for the organization, and the latter also for proposing the collision detection problem. Finally, we thank Jesper Nederlof for some useful observations.

## 1 Introduction

Modular reconfigurable robotics and the related concept of programmable matter concern systems composed of interconnected elementary entities, called modules. The collection of modules can coordinate its limited communication, computation, sensing, and local actuation to accomplish nontrivial global tasks. Local actuation of modules is enabled through a set of one or more mechanical operations that they can perform. An operation typically involves the module that applies it as well as modules in its local neighborhood. Examples of such operations are pushing, pulling, expanding, contracting, doubling, and rotating. Apart from


their induced local changes, these operations are often capable of causing a more global effect on the robotic structure within a limited period of time. An example is when a large part of the structure moves due to the simultaneous application of one or more local operations.

The ability of local operations to globally affect the robotic structure is a double-edged sword. On one hand, it is a convenient form of parallelism, where global structural changes can happen faster. On the other hand, if not properly orchestrated, it could cause small violations of the structure or even complete structural failure, such as uneven cycle growth, global connectivity breaking, and self-intersection of the structure. We, hereafter, shall call all structural violations and failures *collisions*. Operations that – when applied on individual modules – can globally affect the structure, are sometimes called *linear-strength operations*.

The positive effect of such operations has been studied from a theoretical point of view in a number of papers, for different underlying models and types of operations. In the crystalline model, square modules can expand and contract by extending and retracting their faces. In [2], Aloupis et al. gave a universal centralized reconfiguration algorithm that, for any pair of connected shapes $S_I, S_F$ of the same number of modules $n$, can transform $S_I$ into $S_F$ within $O(\log n)$ parallel time steps by performing $\Theta(n \log n)$ individual operations.

In [8], Woods et al. proposed the nubot model, motivated by the programmable self-assembly of molecules, such as DNA strands. The model allows insertion, deletion, and rotation of modules. Their main result is a distributed, asynchronous algorithm which, starting from a singleton, can grow any connected 2D shape and pattern of size $n$, within a polylogarithmic (in $n$) number of parallel time steps in expectation.

Almalki and Michail [1], building on the insertion operations of [8] and the growth processes on graphs by Mertzios et al. [7], investigated what families of shapes can be grown in time polylogarithmic in their size by using only growth operations. They gave centralized algorithms for growing a shape $S_F$ from a shape $S_I$ (possibly a singleton), which yield polylogarithmic parallel time-step schedules for large classes of shapes.

The amoebot model of Derakhshandeh et al. [4] –and its recent canonical extension [3]– is another model in which the main operations considered are expansions and contractions of modules. Shape formation algorithms in this model are usually designed in a way that operations are parallel but each is affecting only a local region around it and not larger parts of the shape. Recently, Feldmann et al. [5] have proposed to add linear-strength operations to the model, but they have left the details of such an extension for future work.

It is evident that most studies have restricted attention to those operations that are safe to perform in parallel. These are either linear-strength operations that cannot collide or operations that affect only the local region around them. In this paper, we explicitly pose the algorithmic question of determining when a set of operations may cause a collision and when a collision can be avoided. In particular, given a shape and a set of linear-strength operations on that shape we aim to give centralized algorithms that can compute a schedule of these (sets of) operations that would $(i)$ cause a collision or $(ii)$ avoid collisions. The former subquestion is motivated by asynchronous distributed algorithms, in which any of the possible interleavings of operations might be the one that the modules will actually realize; the latter by the need to design efficient reconfiguration algorithms that avoid collisions, instead of having collision-avoidance built into the model. To the best of our knowledge, the present is the first study to explicitly consider these types of questions.

## 2 Model

We assume a 2-dimensional square grid where each cell has integer coordinates $(x, y)$. Nodes (modules) occupy cells, defining a set of occupied integer points such that no two nodes occupy the same cell. We represent every node $u = (u_x, u_y)$ as a square of size equal to and

perfectly aligned with cell $(u_x, u_y)$ of the grid. A *shape* $S = (V, E)$ is a configuration of nodes $V$ together with their connectivity, represented by $E$. Only orthogonally adjacent nodes can be connected, but adjacent nodes are not necessarily connected. We use $n$ to denote $|V|$ and restrict our attention to connected shapes, throughout.

**Operations and collisions.** In general, applying one or more *operations* to a shape $S$ either causes a *collision* or yields a new shape $S'$. Collisions come in two types: *node collisions* and *cycle collisions*. Given that all collisions here will be "self-collisions" of a connected shape, we can assume without loss of generality (abbreviated "w.l.o.g." throughout) that there is an *anchor* node $u_0 \in V$ that is stationary and other nodes move relative to it. We begin with the simpler case where the shape is a tree $T = (V, E)$, where cycle collisions do not exist, and then generalize to any connected shape $S$.

We start by defining single *expansion* and *contraction* operations[1]. An *expansion* operation is applied to a pair of adjacent integer points $uv$, where either $(i)$ $u \in V$ and $v \notin V$, or $(ii)$ $u, v \in V$ and $uv \in E$ holds. The remaining case where $u, v \in V$ but $uv \notin E$ immediately gives a collision. In case (i), the expansion generates a node at the empty cell $v$ connected to $u$. In case (ii), assume w.l.o.g. that $u$ is closer to $u_0$ in $T$ than $v$. Let $T(v)$ denote the subtree of $T$ rooted at $v$. Then, the expansion generates a node between $u$ and $v$, connected to both, which translates $T(v)$ by one unit away from $u$ along the axis parallel to $uv$. In both cases, the new node starts as a unit-length segment that widens into a unit square. A *contraction* operation is applied to a pair of nodes $uv \in E$, $v$ being the furthest from the anchor. It merges $v$ with $u$ by translating $T(v)$ by one unit toward $u$ while $v$ narrows to a unit-length segment. In both types of operations, if after $T(v)$'s translation two nodes occupy the same cell then a collision has occurred. We call this type of collision a *node collision* and more generally define it as the non-empty intersection of the areas of any two nodes at any point in time. Otherwise, a new tree $T'$ has been obtained.

We assume that no node is ever adjacent to more than one operation.

**Coupling.** Let $Q$ be a set of operations to be applied *in parallel* to a connected shape $S$, each operation on a distinct pair of nodes or a node and an unoccupied cell. We call such a set a *coupling*, and the operations it contains are *coupled* or *parallel*. We assume that all operations in $Q$ are applied *concurrently*, have the same *constant execution speed*, and their *duration* is equal to one unit of time.

Let $T = (V, E)$ be a tree and $u_0 \in V$ its anchor. We set $u_0$ to be the root of $T$. We want to determine the displacement of every $v \in V \setminus \{u_0\}$ due to the parallel application of the operations in $Q$. As $u_0$ is stationary and each operation translates a subtree, only the operations on the unique $u_0v$ path contribute to $v$'s displacement. In particular, any such operation contributes one of the unit vectors $\langle -1, 0\rangle, \langle 0, -1\rangle, \langle +1, 0\rangle, \langle 0, +1\rangle$ to the motion vector $\vec{v}$ of $v$. Moreover, for any node $u \in V$ that expands toward an empty cell, we add a new node $v$ with a corresponding unit motion vector $\vec{v}$. We can use the set of motion vectors to determine whether the trajectories of any two nodes will collide at any point.

Now, let $S$ be any connected shape with at least one cycle and any node $u_0$ be its anchor. Then, a set of operations $Q$ on $S$ either causes a *cycle collision* or its effect is essentially equivalent to the application of $Q$ on any spanning tree of $S$ rooted at $u_0$. Let $u$, $v$ be any

[1] We believe that our definitions and techniques can be extended to alternative versions of expansion and contraction – including the case where the operations can be reversed – and to different geometries such as a triangular grid.

two nodes on a cycle. If $p_1$ and $p_2$ are the two $uv$ paths of the cycle, then $\vec{v}_{p_1} = \vec{v}_{p_2}$ must hold: the displacement vectors of $v$ along the paths $p_1$ and $p_2$ are equal. Otherwise, we cannot maintain all nodes or edges of the cycle. Such a violation is called a *cycle collision*. We call a set of operations that does not cause any node or cycle collisions *collision free*.

**Discrete and continuous time.** We consider two different models for the scheduling of the operations. In the *discrete-time model*, each operation or coupling starts at a different integer time and takes one unique unit of time. In other words, no two operations are active at the same time unless they are coupled. In the *continuous-time model*, we do not make the integer starting-time assumption. Operations can start at any time and their active times can overlap. Coupled operations start and finish at the same time. Our assumption that each operation takes one unit of time to complete and has constant execution speed holds for both timing models. In the discrete-time model, only the order of the operations (individual or coupled) matters for having collisions or not. In the continuous-time model, the precise starting times of the operations matter.

**Problem definitions.** We now define the problems considered. Given a shape $S$ and an assignment of operations on $S$ that involve any node at most once, a *coupling partition of operations on* $S$ is a collection of sets $\{Q_1, Q_2, \ldots, Q_k\}$, where each $Q_i$ (possibly a singleton) denotes a subset of the operations that should be performed in parallel.

Colliding Schedule**.** Given a shape $S = (V, E)$ from a given family of shapes and a coupling partition of operations $\{Q_1, Q_2, \ldots, Q_k\}$ on $S$, decide if a starting time $t_0(Q_i) \in \mathbb{R}$ for each coupled set $Q_i$ exists such that the application of the operations according to these starting times causes a collision.

Collision-free Schedule**.** Given a shape $S = (V, E)$ from a given family of shapes and a coupling partition of operations $\{Q_1, Q_2, \ldots, Q_k\}$ on $S$, decide if a starting time $t_0(Q_i) \in \mathbb{R}$ for each coupled set $Q_i$ exists such that the application of the operations according to these starting times is collision free.

The discrete special cases of these problems, Discrete Colliding Schedule and Discrete Collision-free Schedule, respectively, are obtained by requiring all $t_0(Q_i)$'s to be unique integers.

## 3 Algorithms for Colliding Schedule

In this section, we present algorithms to decide whether a connected shape can have collisions for some schedule of operations. We first consider the continuous model followed by the discrete model. We distinguish the cases based on the type of coupling.

We assume that the topology of $S$ is that of a tree. We refer the readers to [6] for details regarding general graphs. In the case of continuous model, we get the following results.

▶ **Theorem 1.** *Let $S$ be a shape consisting of $n$ unit square nodes with operations defined on the edges between adjacent nodes, and let the adjacency structure of $S$ be a single tree. Then we can solve* Colliding Schedule

- *in $O(n^2 \log^2 n)$ time if couplings exist;*
- *in $O(n^2)$ time if each coupling has constant size, or is horizontal-only or vertical-only;*
- *in $O(n \log^2 n)$ time if the operations are not coupled.*

We can also solve Discrete Colliding Schedule in polynomial time. The algorithm without coupling is still correct, but with coupling we need a different approach. Thus, we get the following results.

▶ **Theorem 2.** *Let $S$ be a shape consisting of $n$ unit square nodes with operations defined on the edges between adjacent nodes, and let the adjacency structure of $S$ be a single tree. Then we can solve* Discrete Colliding Schedule
- *in $O(n^{17/3})$ time if couplings exist;*
- *in $O(n^5)$ time if each coupling has constant size, or is horizontal-only or vertical-only;*
- *in $O(n \log^2 n)$ time if the operations are not coupled.*

## 4 Continuous and Discrete Collision-free Schedule

So far we considered detecting whether collisions might occur for an input instance. In this section, we consider the problem of deciding if all operations can be performed without any collisions, for a suitable choice of operation order or starting times. We show that, even if there are only expansions that are w.l.o.g. horizontal and couplings have size $O(1)$, in the discrete-time model the problem is NP-complete. Interestingly, the same problem is solvable in polynomial time in the continuous-time model. When we add vertical expansions, the problem is NP-hard in the continuous-time model.

▶ **Theorem 3.** *Discrete Collision-free Schedule is NP-complete even if all operations are horizontal expansions and all couplings have size $O(1)$.*

▶ **Theorem 4.** *Collision-free Schedule is solvable in linear time if all operations are horizontal.*

▶ **Theorem 5.** *Collision-free Schedule is NP-hard.*

## References


1 Nada Almalki and Othon Michail. On geometric shape construction via growth operations. *Theor. Comput. Sci.*, 984:114324, 2024.

2 Greg Aloupis, Sébastien Collette, Erik D. Demaine, Stefan Langerman, Vera Sacristán Adinolfi, and Stefanie Wuhrer. Reconfiguration of cube-style modular robots using O(log n) parallel moves. In *ISAAC*, volume 5369 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2008.

3 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The canonical amoebot model: algorithms and concurrency control. *Distributed Comput.*, 36(2):159–192, 2023.

4 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: amoebot - a new model for programmable matter. In *SPAA*, pages 220–222. ACM, 2014.

5 Michael Feldmann, Andreas Padalkin, Christian Scheideler, and Shlomi Dolev. Coordinating amoebots via reconfigurable circuits. *J. Comput. Biol.*, 29(4):317–343, 2022.

6 Siddharth Gupta, Marc J. van Kreveld, Othon Michail, and Andreas Padalkin. Collision detection for modular robots - it is easy to cause collisions and hard to avoid them. *CoRR*, abs/2305.01015, 2023.

7 George B. Mertzios, Othon Michail, George Skretas, Paul G. Spirakis, and Michail Theofilatos. The complexity of growing a graph. In *ALGOSENSORS*, volume 13707 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 2022.

8 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 *ITCS*, pages 353–354. ACM, 2013.