Particle Computation: Complexity, Algorithms, and Logic

Aaron T. Becker[a], Erik D. Demaine[b], Sándor P. Fekete[c,*], Jarrett Lonsford[a], Rose Morris-Wright[d]

[a]*Department of Electrical and Computer Engineering, University of Houston, TX, USA,*
[b]*Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA,*
[c]*Department of Computer Science, TU Braunschweig, Germany,*
[d]*Mathematics, Brandeis University, Waltham, MA, USA*

Abstract

We investigate algorithmic control of a large swarm of mobile particles (such as robots, sensors, or building material) that move in a 2D workspace using a global input signal (such as gravity or a magnetic field). Upon activation of the field, each particle moves maximally in the same direction until forward progress is blocked by a stationary obstacle or another stationary particle. In an open workspace, this system model is of limited use because it has only two controllable degrees of freedom—all particles receive the same inputs and move uniformly. We show that adding a maze of obstacles to the environment can make the system drastically more complex but also more useful.

We provide a wide range of results for a wide range of questions. These can be subdivided into *external* algorithmic problems, in which particle configurations serve as input for computations that are performed elsewhere, and *internal* logic problems, in which the particle configurations themselves are used for carrying out computations.

For external algorithms, we give both negative and positive results. If we are *given* a set of stationary obstacles, we prove that it is NP-hard to decide whether a given initial configuration of unit-sized particles can be transformed into a desired target configuration. Moreover, we show that finding a control sequence of minimum length is PSPACE-complete. We also work on the inverse problem, providing constructive algorithms to *design* workspaces that efficiently implement arbitrary permutations between different configurations.

For internal logic, we investigate how arbitrary computations can be implemented. We demonstrate how to encode *dual-rail logic* to build a universal logic gate that concurrently evaluates AND, NAND, NOR, and OR operations. Using many of these gates and appropriate interconnects, we can evaluate any logical expression. However, we establish that simulating the full range of complex interactions present in arbitrary digital circuits encounters a fundamental difficulty: a FAN-OUT gate cannot be generated. We resolve this missing component with the help of $2{\times}1$ particles, which can create FAN-OUT gates that produce multiple copies of the inputs. Using these gates we provide rules for replicating arbitrary digital circuits.

Keywords: Programmable matter, robot swarms, nano-particles, uniform inputs, parallel motion planning, complexity, array permutations, NP-completeness, PSPACE-completeness, efficient algorithms, logic gates, universal computation.

[*]Corresponding author
Email addresses: `atbecker@uh.edu` (Aaron T. Becker), `edemaine@mit.edu` (Erik D. Demaine), `s.fekete@tu-bs.de` (Sándor P. Fekete), `JLLonsford@uh.edu` (Jarrett Lonsford), `rmorriswright@brandeis.edu` (Rose Morris-Wright)
This paper provides full details for and combines results of a number of different extended abstracts that have appeared in the International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013) [7], IEEE International Conference on Robotics and Automation (ICRA 2014) [8] and ICRA 2015 [58]. See video from the 31st International Symposium on Computational Geometry (SoCG'15) [9] for illustrations and animations.

1. Introduction

"Programmable matter refers to a substance that has the ability to change its physical properties (shape, density, moduli, conductivity, optical properties, etc.) in a programmable fashion, based upon user input or autonomous sensing. The potential applications are endless, e.g., smart materials, autonomous monitoring and repair, or minimal invasive surgery. Thus, there is a high relevance of this topic to industry and society in general, and much research has been invested in the past decade to fabricate programmable matter. However, fabrication is only part of the story: without a proper understanding of how to program that matter, complex tasks such as minimal invasive surgery will be out of reach." [29]

Since the first visions of massive sensor swarms, more than ten years of work on sensor networks have yielded considerable progress with respect to hardware miniaturization. The original visions of "Smart Paint" [2] or "Smart Dust" [41] have triggered a considerable amount of theoretical research on swarms of *stationary* processors, e.g., the work in [26, 27, 28, 43]. Recent developments in the ability to design, produce, and control particles at the micro and nanoscale and the rise of possible applications, e.g., targeted drug delivery, micro and nanoscale construction, and Lab-on-a-Chip, motivate the study of large swarms of *mobile* objects. But how can we control such a swarm with only limited computational power and a lack of individual control by a central authority? Local, robotics-style motion control by the particles themselves appears hopeless because the capacity for storing energy for computation, communication, and motion control is proportional to the volume, but volume shrinks with the third power of particle length.

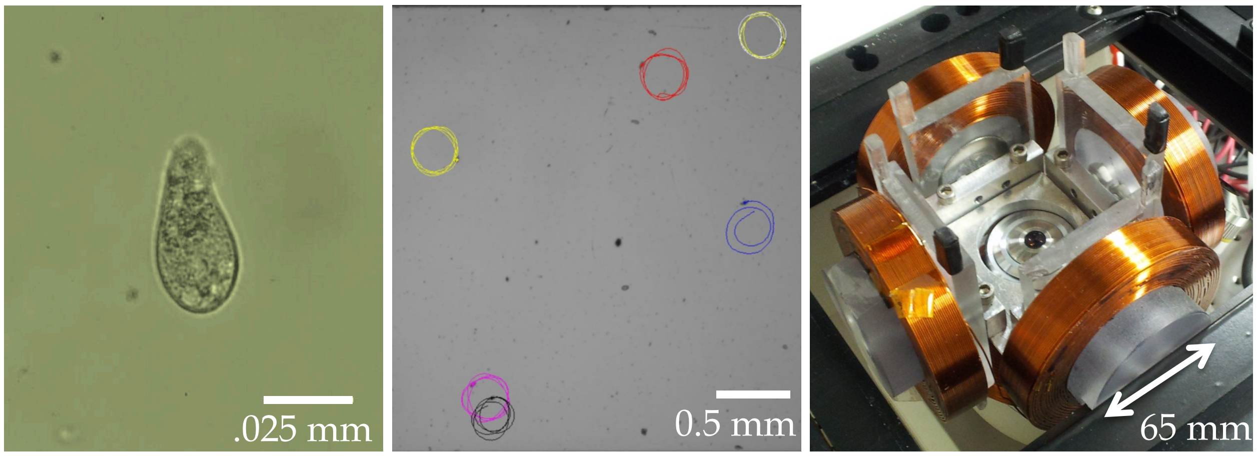
A possible answer lies in applying a global, external force to all particles in the swarm. This resembles the logic puzzle Tilt [59], slide and merge games such as the 2048 puzzle [1], and dexterity ball-in-a-maze puzzles such as Pigs in Clover and Labyrinth, which involve tilting a board to cause all mobile pieces to roll or slide in a desired direction. Problems of this type are also similar to sliding-block puzzles with fixed obstacles [17, 38, 39, 37], except that all particles receive the same control inputs, as in the Tilt puzzle. In the real world, driving ferromagnetic particles with a magnetic resonance imaging (MRI) scanner gives a milli-scale example of this challenge [61]. At the micro-scale, Becker et al. [13] demonstrate how to apply a magnetic field to simultaneously move cells containing iron particles in a specific direction within a fabricated workspace (see Fig. 1a). Other recent examples include using the global magnetic field from an MRI to guide magneto-tactic bacteria through a vascular network to deliver payloads at specific locations [15] and using electromagnets to steer a magneto-tactic bacterium through a micro-fabricated maze [42]; however, this still involves only individual particles at a time, not the parallel motion of a whole, massive swarm. How can we manipulate the overall swarm with coarse global control, such that individual particles arrive at multiple different destinations in a (known) complex vascular network such as the one in Fig. 1b? And how can we use the complex interaction of the particles to carry out complex computations *within* the swarm?

All this gives rise to the following two families of problems, which we denote by *External Computation* and *Internal Computation.*

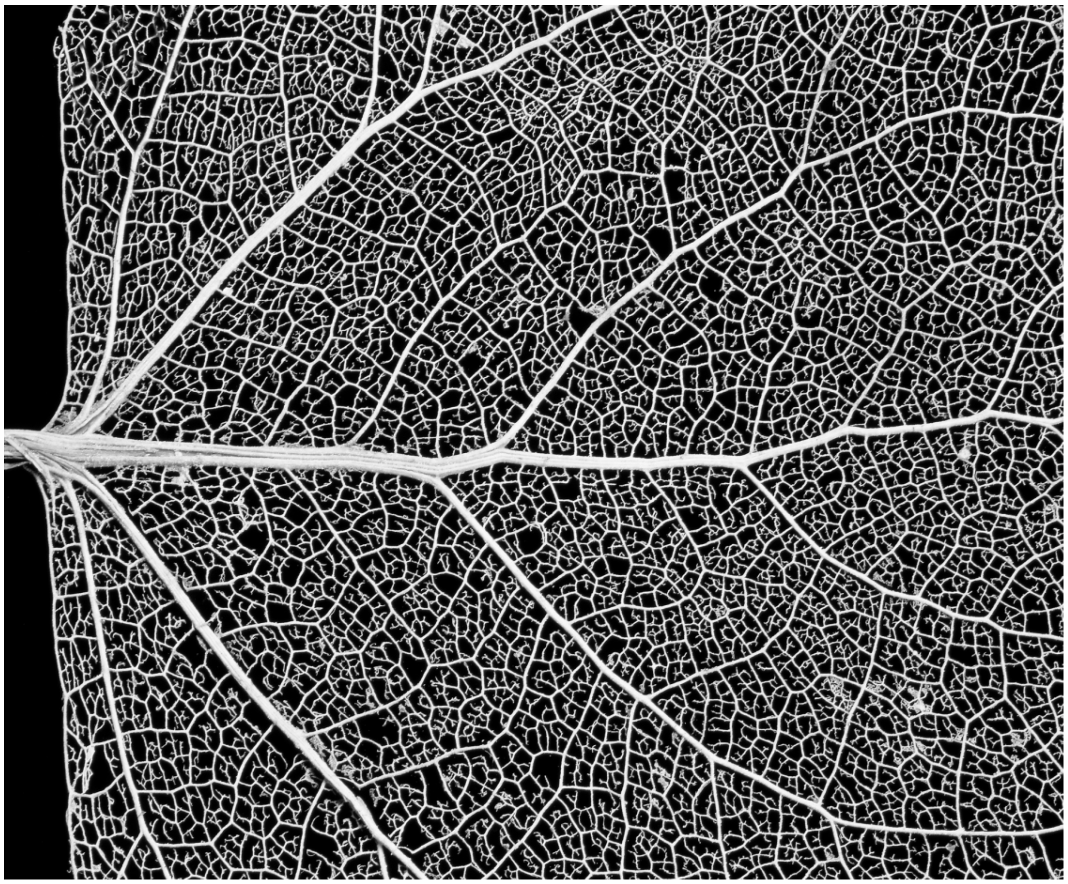
External Computation. Considering the particle swarm as input for a given algorithmic problem, we are faced with a number of questions that need to be resolved externally, such as the following:

1. Given a map of an environment, such as the vascular network shown in Fig. 1b, along with initial and goal positions for each particle, does there exist a sequence of inputs that will bring each particle to its goal position?
2. Given a map of an environment, such as the vascular network shown in Fig. 1b, along with initial and goal positions for each particle, what is the shortest sequence of moves that will bring each particle to its goal position?
3. Given initial and goal positions for each particle in a swarm, how can we design a set of obstacles and a sequence of moves, such that each particle reaches its goal position?

Deliberate use of existing stationary obstacles leads to a wide range of possible particle configurations. In the first part of the paper (Section 4 and Section 5), we give answers to the first two questions by showing



(a) (Left) After feeding iron particles to ciliate eukaryon (*Tetrahymena pyriformis*) and magnetizing the particles with a permanent magnet, the cells can be turned by changing the orientation of an external magnetic field (see colored paths in the center image). (Right) Using two orthogonal Helmholz electromagnets, Becker et al. [13] demonstrated steering many living magnetized *T. pyriformis* cells. All cells are steered by the same global field.



(b) Biological vascular network (cottonwood leaf). (Photo: Royce Bair/Flickr/Getty Images.) Given such a network along with initial and goal positions of N particles, is it possible to bring each particle to its goal position using a global control signal? Note that this arrangement is *not* a tree, but a graph structure with many cycles. MATLAB code for driving N particles through this network is available at http://www.mathworks.com/matlabcentral/fileexchange/42892.

Figure 1: (Top) State of the art in controlling small objects by force fields. (Bottom) A complex vascular network, forming a typical environment for the parallel navigation of small objects. This paper investigates parallel navigation in discretized 2D environments.

that they may lead to computationally difficult situations. We also develop several positive results for the third question (again in Section 5). The underlying idea is to construct artificial obstacles (such as walls) that allow arbitrary rearrangements of a given two-dimensional particle swarm. For clearer notation, we will formulate the relevant statements in the language of matrix operations, which is easily translated into plain geometric language. This paper investigates these problems in 2D discretized environments, leaving 3D and continuous environments for future work.

Internal Computation. Considering the particle swarm as a complex system that can be reconfigured in various ways, we are faced with issues of the computational power of the swarm itself, such as the following:

1. Can the complexity of particle interaction be exploited to model logical operations?
2. Are there limits to the computational power of the particle swarm?
3. How can we achieve computational universality with particle computation?

In the second part of the paper (Section 6), we give precise answers to all of these questions. In particular, we show that the logical operations AND, NAND, NOR, and OR can be implemented in our model using dual-rail logic. Using terminology from electrical engineering, we call these components that calculate logical operations *gates*. We establish a fundamental limitation for particle interactions: we cannot duplicate the output of a chain of gates without also duplicating the chain of gates. This means a FAN-OUT gate cannot be generated. We resolve this missing component with the help of 2×1 particles, which can be used to create FAN-OUT gates that produce multiple copies of the inputs without needing duplicate gates. Using FAN-OUT gates, we provide rules for replicating arbitrary digital circuits.

In the following, we start by a brief formal problem definition (Section 2) and a discussion of related work (Section 3), then continue to provide the results on External Computation (Sections 4 and 5) and Internal Computation (Section 6 and 7). We conclude with future work (Section 8).

2. Problem Definition

2.1. Model

Our model is based on the following rules:

1. A planar grid *workspace* W contains a number of unit-size particles and some fixed unit-square obstacles. A grid cell is referenced by its Cartesian coordinates $\mathbf{x} = (x, y)$, and is either *free* for possible occupation by a particle, or a permanent *obstacle*, which may never be occupied by a particle. Each particle occupies one grid cell.
2. All particles are commanded in unison: the valid commands are "Go Up" (u), "Go Right" (r), "Go Down" (d), and "Go Left" (ℓ).
3. The particles all move in the commanded direction until forward progress is blocked by a stationary obstacle or another blocked particle. A *command sequence* $\mathbf{m}$ consists of an ordered sequence of moves m_k, where each $m_k \in \{u, d, r, \ell\}$. A representative command sequence is $\langle u, r, d, \ell, d, r, u, \ldots\rangle$. We assume that W is bounded by obstacles and issue each command long enough for the particles to move to their maximum extent.

The algorithmic decision problem GLOBALCONTROL-MANYPARTICLES is to decide whether a given puzzle is solvable. In other words, given a fixed workspace and a start and goal location for each particle, the algorithm determines the existence of a sequence of moves that move the particles to their goal locations. See Fig. 2 for two simple instances.

3. Related Work

Related work is categorized into underactuated control, manipulation, and computation.

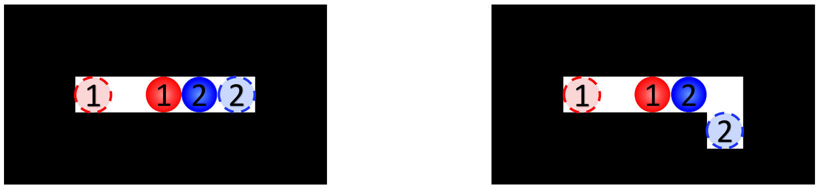


Figure 2: In this image, black cells are fixed, white cells are free, solid discs are individual particles, and goal positions are dashed circles. For the simple instance on the left, it is impossible to maneuver both particles to end at their goals. The instance on the right has a finite solution: $\langle r, d, \ell \rangle$.

3.1. Underactuated Control

Large Robot Populations. Due to the efforts of roboticists, biologists, and chemists (e.g., [57, 53, 16]), it is now possible to make and field large ($N = 10^3$–10^{14}) populations of simple robots. Potential applications for these robots include targeted medical therapy, sensing, and actuation. With large populations come two fundamental challenges: how to (1) perform state estimation for the robots and (2) control these robots.

Traditional approaches often assume independent control signals for each robot, but each additional independent signal requires bandwidth and engineering. These bandwidth requirements grow at $O(N)$. Using independent signals becomes more challenging as the robot size decreases. Especially at the micro- and nano-scales, it is not practical to encode autonomy in the robots. Instead, the robots are controlled and interacted with using global control signals. For this reason, it may be more appropriate to call the moving agents *particles* and label the external control system as the robot.

More recently, robots have been constructed with physical heterogeneity so that they respond differently to a global, broadcast control signal. Examples include *scratch-drive microrobots*, actuated and controlled by a DC voltage signal from a substrate [19]; magnetic structures with different cross-sections that could be independently steered [30]; *MagMite* microrobots with different resonant frequencies and a global magnetic field [32]; and magnetically controlled nanoscale helical screws constructed to stop movement at different cutoff frequencies of a global magnetic field [54]. In previous work with robots modeled as nonholonomic unicycles, we showed that an inhomogeneity in turning speed is enough to make even an infinite number of robots controllable with regard to position. All these approaches show promise, but they require precise state estimation and heterogeneous robots. At the molecular scale, there is a bounded number of modifications that can be made to differentiate robots. In addition, the control law computation requires at best a summation over all the robot states $O(N)$ [11, 12] and at worst a matrix inversion $O(N^{2.373})$[6].

In this paper we take a different approach. We assume a population of approximately identical planar particles (which could be small robots) and one global control signal that contains the direction all particles should move. In an open environment, this system is not controllable because the particles move uniformly—implementing any control signal translates the entire group identically; however, an obstacle-filled workspace allows us to break this symmetry. In previous practical work [10], we showed that if we can command the particles to move one unit distance at a time, some goal configurations have easy solutions. Given a large free space, we have an algorithm showing that a single obstacle is sufficient for position control of N particles (video of position control: `http://youtu.be/5p_XIad5-Cw`). This result required incremental position control of the group of particles, i.e. the ability to advance them a uniform, fixed distance. This is a strong assumption and one that we relax in this work.

Dexterity Games. The problem we investigate is strongly related to dexterity puzzles—games that typically involve a maze and several balls that should be maneuvered to goal positions. Such games have a long history. *Pigs in Clover*, involving steering four balls through three concentric incomplete circles, was invented in 1880 by Charles Martin Crandall. Dexterity games are dynamic and depend on the manual skill of the player. Our problem formulation also applies the same input to every agent but imposes only kinematic restrictions on agents. This is most similar to the gravity-fed logic maze *Tilt*$^{\text{TM}}$, invented by Vesa Timonen and Timo Jokitalo and distributed by ThinkFun since 2010 [59].

Sliding-Block Puzzles. Sliding-block puzzles use rectangular tiles that are constrained to move in a 2D workspace. The objective is to move one or more tiles to desired locations. They have a long history. Hearn [36] and Demaine et al. [18] showed that tiles can be arranged to create logic gates and used this technique to prove PSPACE-completeness for a variety of sliding-block puzzles. Hearn expressed the idea of building computers from the sliding blocks—many of the logic gates could be connected together, and the user could propagate a signal from one gate to the next by sliding intermediate tiles. This requires the user to know precisely which sequence of gates to enable and disable. In contrast to such a hands-on approach, with our architecture we can build circuits, store parameters in memory, and then actuate the entire system in parallel using a global control signal.

3.2. Manipulation

Computational Geometry: Robot Box-Pushing. Many variations of block-pushing puzzles have been explored from a computational complexity viewpoint with a seminal paper proving NP-hardness by Gordon Wilfong in 1991 [64]. The general case of motion-planning when each command moves particles a single unit in a world composed of even a single robot and both *fixed* and *moveable* squares is in the complexity class PSPACE-complete [18, 20, 38].

Ricochet Robots [23], Atomix [39], and PushPush [17] have the same constraint that particles must move to their full extent, once they have been set in motion. This constraint reflects physical realities where, due to uncertainties in sensing, control application, and dynamic models, precise quantified movements in a specified direction are not possible. Instead, the input can be applied for a long period of time, guaranteeing that all particles move to their fullest extent. In these games the particles move to their full extent with each input, but each particle can be actuated individually. The problem complexity with global inputs to all particles is addressed in this paper.

Sensorless Manipulation. The algorithms in the second half of our paper do not require feedback, and we have drawn inspiration from work on sensorless manipulation [24]. Sensorless manipulation explicitly maintains the set of all possible part configurations and selects a sequence of actions to reduce the size of this set and drive it toward some goal configuration. Carefully selected primitive operations can make this easier. Sensorless manipulation strategies often use a sequential composition of primitive operations, "squeezing" a part either virtually with a programmable force field or simply between two flat, parallel plates [35]. Some sensorless manipulation strategies take advantage of limit cycle behavior, for example, engineering fixed points and basins of attraction so that parts only exit a feeder when they reach the correct orientation [46, 51]. These two strategies have been applied to a much wider array of mechanisms such as vibratory bowls and tables [33, 62, 63] or assembly lines [4, 35, 60], and have also been extended to situations with stochastic uncertainty [34, 50] and closed-loop feedback [5, 52].

3.3. Computation

Parallel Algorithms: SIMD. Another related area of research is Single Instruction Multiple Data (SIMD) parallel algorithms [45]. In this model, multiple processors are all fed the same instructions to execute, but they do so on different data. This model has some flexibility, for example, allowing command execution selectively only on certain processors and no operations (NOPs) on the remaining processors.

Our model is actually more extreme. The particles all respond in effectively the same way to the same instruction. The only difference is their location and which obstacles or particles will block them. In some sense, our model is essentially Single Instruction, Single Data, Multiple Locations.

Our efforts have similarities with *mechanical computers*, computers constructed from mechanical, not electrical components. For a fascinating nontechnical review, see [48]. These devices have a rich history, from the *Pascaline*, an adding machine invented in 1642 by a nineteen-year old Blaise Pascal, to Herman Hollerith's punch-card tabulator in 1890, to the mechanical devices of IBM culminating in the 1940s. These devices used precision gears, pulleys, or electric motors to carry out calculations. Though our implementations in this paper are rather basic, we require none of these precision elements to achieve computational universality—merely unit-size obstacles and sliding particles sized 2×1 and 1×1.

Collision-Based Computing. Collision-based computing has been defined as *"computation in a structureless medium populated with mobile objects."* For a survey of this area, see the excellent collection [3]. Early examples include the billiard-ball computer proposed by Fredkin and Toffoli using only spherical balls and a frictionless environment composed of elastic collisions with other balls and with angled walls [31]. Another popular example is Conway's *Game of Life*, a cellular automaton governed by four simple rules [14]. Cells live or die based on the number of neighbors. These rules have been examined in depth and used to design a Turing-complete computer [55, 56]. Game of life scenarios and billiard-ball computers are fascinating but lack a physical implementation. In this paper we present a collision-based system for computation and provide a physical implementation.

Programmable Matter. Clearly there is a wide range of interesting scenarios for developing approaches to programmable matter. One such model is the *abstract Tile-Assembly Model* (aTAM) by Winfree [65, 66, 44], which has sparked a wide range of theoretical and practical research. In this model, unit-sized tiles interact and bond with the help of differently labeled edges, eventually composing complex assemblies. Even though the operations and final objectives in this model are quite different from our particle computation with global inputs (e.g., key features of the aTAM are that tiles can have a wide range of different edge types, and that they keep sticking together after bonding), there is a remarkable geometric parallelism to a key result of our present paper: while it is widely believed that at the most basic level of interaction (called *temperature 1*), computational universality *cannot* be achieved [21, 47, 49] in the aTAM with only unit-sized tiles, recent work [25] shows that computational universality *can* be achieved as soon as even slightly bigger tiles are used. This resembles the results of our paper, which shows that unit-size particles are insufficient for universal computation, but employing bigger particles suffices.

4. Mazes

We prove that the general problem defined in Section 2 is computationally intractable.

Theorem 1. GLOBALCONTROL-MANYPARTICLES *is NP-hard: given a specified goal location and an initial configuration of movable particles and fixed obstacles, it is NP-hard to decide if a move sequence exists that ends with some particle at the goal location.*

Proof. We prove hardness by a reduction from 3SAT. Suppose we are given n Boolean variables $x_1, x_2, \ldots, x_n$, and m disjunctive clauses $C_j = U_j \vee V_j \vee W_j$, where each literal U_j, V_j, W_j is of the form x_i or $\neg x_i$. We construct an instance of GLOBALCONTROL-MANYPARTICLES that has a solution if and only if all clauses can be satisfied by a truth assignment to the variables. This instance is composed of *variable gadgets* for setting individual variables TRUE or FALSE, *clause gadgets* that construct the logical OR of groupings of three variables, and a *check gadget* that constructs the logical AND of all the clauses. A particle is only delivered to the goal location if the variables have been set in such a way that the formula evaluates to TRUE.

Variable gadgets. For each variable x_i that appears in k_i literals, we construct k_i instances of the *variable gadget* i shown in Fig. 3, with a particle initially at the top of the gadget. The gadget consists of a tower of n levels, designed for the overall construction to make n total variable choices. These choices are intended to be made by a move sequence of the form $\langle d, l/r, d, l/r, \ldots, d, l/r, d, r\rangle$, where the ith l/r choice corresponds to setting variable x_i to either TRUE (l) or FALSE (r). Thus variable gadget i ignores all but the ith choice by making all other levels lead to the same destination via both l and r. The ith level branches into two destinations chosen by either l or r, which correspond to x_i being set TRUE or FALSE, respectively.

In fact, the command sequence may include multiple l and r commands in a row, in which case the last l/r before a vertical u/d command specifies the final decision made at that level, and the others can be ignored. The command sequence may also include a u command, which undoes a d command if done immediately after or else does nothing; thus we can simply ignore the u command and the immediately preceding d, if it exists. We can also ignore duplicate commands (e.g., d, d becomes d) and remove any initial l/r command. After ignoring these superfluous commands, assuming a particle reaches one of the output channels, we obtain a sequence in the canonical form $\langle d,\ l/r,\ d,\ l/r,\ \ldots,\ d,\ r\rangle$ as desired, corresponding uniquely to a truth assignment to the n variables. (If no particle reaches the output port, it is as if the variable is neither True nor False, satisfying no clauses.) Note that all particles arrive at their output ports at exactly the same time.

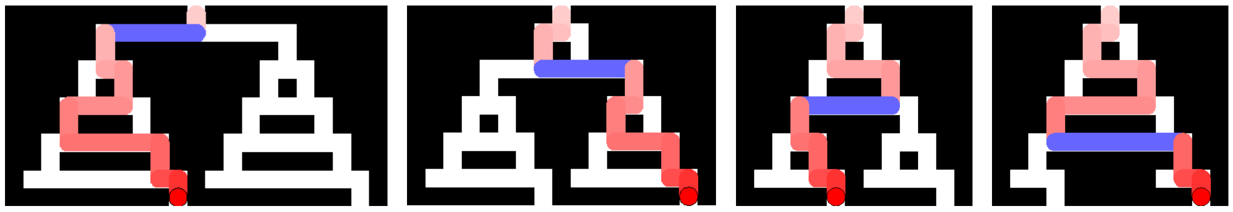
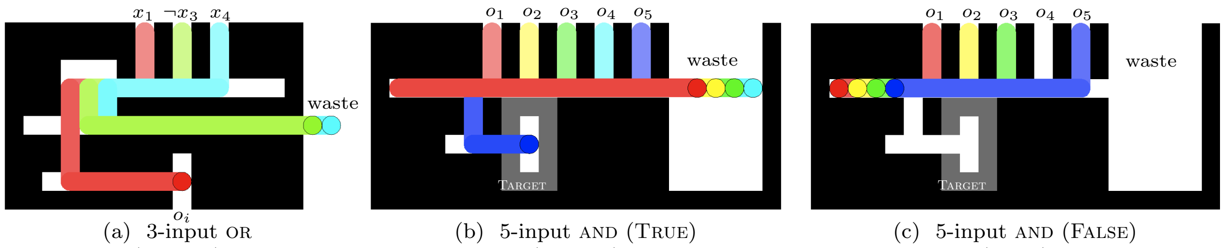

(a) Variable x_1 set True for $i = 1$ (b) x_2 set False, $i = 2$ (c) x_3 set True, $i = 3$ (d) x_4 set False, $i = 4$

Figure 3: Variable gadgets that are assigned a truth value by executing a sequence of $\langle d, \ell/r\rangle$ moves. The ith ℓ/r choice sets the variable x_i to True or False by putting the particle in the left or right column. This selection move is shown in blue. Each gadget is designed to respond to the ith choice but ignore all others. This lets us make several copies of the same variable by making multiple gadgets with the same i. In the figure $n = 4$, and the input sequence $\langle d, \ell, d, r, d, \ell, d, r, d, r, d\rangle$ sets (x_1=True, x_2=False, x_3=True, x_4=False).

Clause gadgets. For each clause, we use the OR gadget shown in Fig. 4a. The OR gadget has three inputs corresponding to the three literals, and input particles are initially at the top of these inputs. For each positive literal of the form x_i, we connect the corresponding input to the left output of an unused instance of variable gadget i. For each negative literal of the form $\neg x_i$, we connect the corresponding input to the right output of an unused instance of a variable gadget i. (In this way, each variable gadget gets used exactly once.)

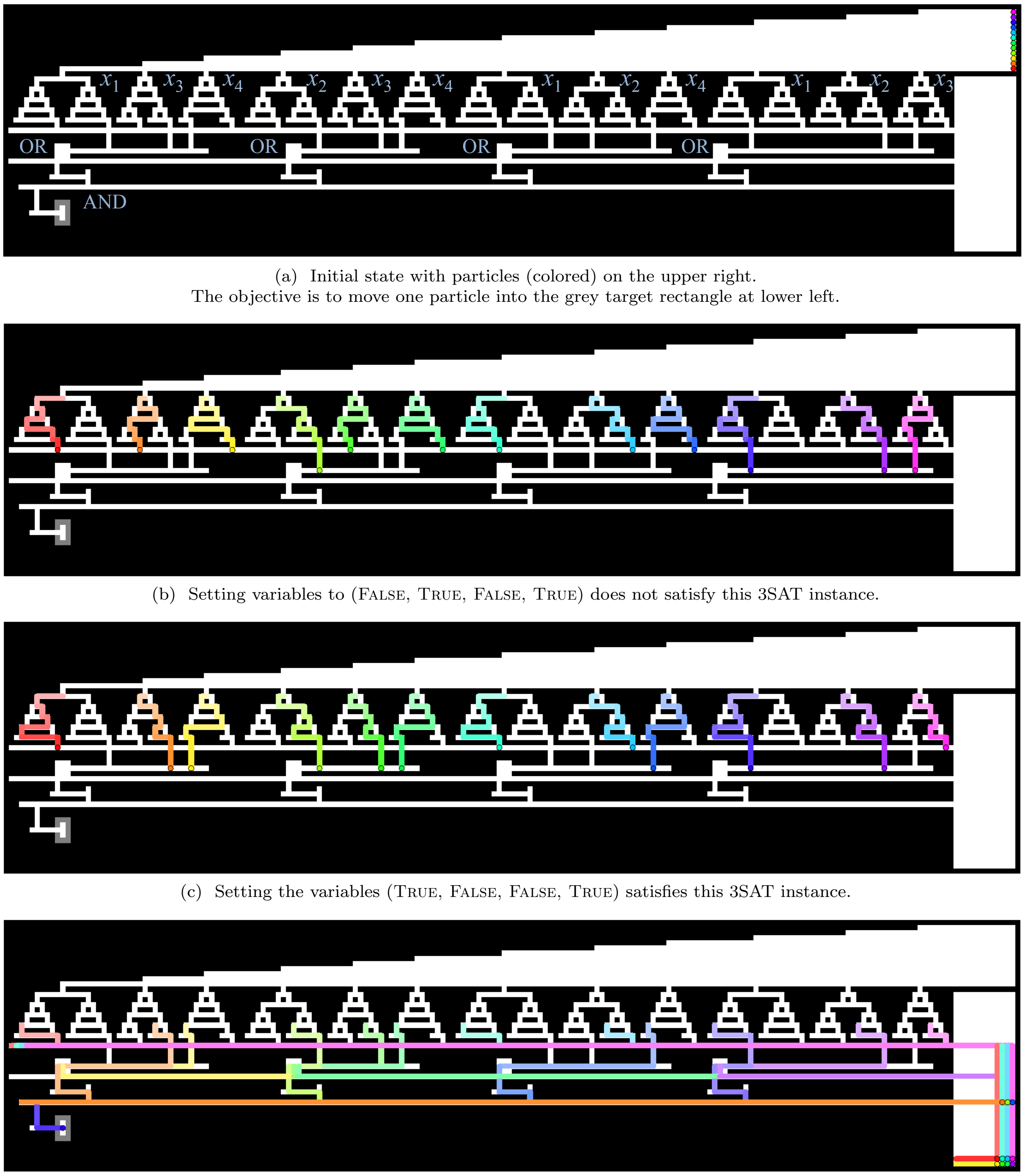
We connect the variable gadget to the OR gadget as shown in Fig. 5: place the variable gadget above the clause so as to align the vertical output and input channels, and join them into a common channel. To make room for the three variable gadgets, we simply extend the black areas separating the three input channels in the OR gadget. The unused output channel of each variable gadget is connected to a waste receptacle. Any particle reaching that end cannot return to the logic.

If any input channel of the OR gadget has a particle, then it can reach the output port by the move sequence $\langle d, \ell, d, r\rangle$. Furthermore, because variable gadgets place all particles on their output ports at the same time, if more than one particle reaches the OR gadget, they will move in unison as drawn in Fig. 4a, and only one can make it to the output port; the others will be stuck in the "waste" row, even if extra $\langle \ell, r, u, d\rangle$ commands are interjected into the intended sequence. Hence, a single particle can reach the output of a clause if and only if that clause (i.e., at least one of its literals) is satisfied by the variable assignment.



(a) 3-input OR $\langle d, \ell, d, r\rangle$ (b) 5-input AND (True) $\langle d, \ell, d, r\rangle$ (c) 5-input AND (False) $\langle d, \ell, d\rangle$

Figure 4: Gadgets that use the cycle $\langle d, \ell, d, r\rangle$. The 3-input OR gadget outputs one particle if at least one particle enters in an input line and sends any extra particle(s) to a waste receptacle. The 5-input AND gadget outputs one particle to the Target Location, marked in gray, if at least 5 inputs are True. Excess particles are sent to a waste receptacle.



(a) Initial state with particles (colored) on the upper right.
The objective is to move one particle into the grey target rectangle at lower left.

(b) Setting variables to (False, True, False, True) does not satisfy this 3SAT instance.

(c) Setting the variables (True, False, False, True) satisfies this 3SAT instance.

(d) Successful outcome. (True, False, False, True) moves a single particle into the target region.

Figure 5: Combining 12 variable gadgets, four 3-input OR gadgets, and a 4-input AND gadget to realize the 3SAT expression $(\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee \neg x_2 \vee x_3)$.

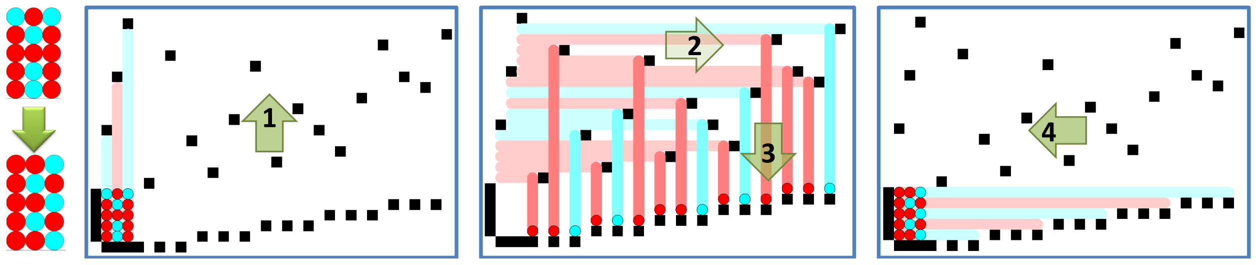


Figure 6: In this image for $N = 15$, black cells are obstacles, white cells are free, and colored discs are individual particles. The world has been designed to permute the particles between 'A' into 'B' every four steps: $\langle u, r, d, \ell \rangle$. See video at http://youtu.be/3tJdRrNShXM. Visually, the distinction between particles of the same color does not matter; however, the arrangement of obstacles induces a specific permutation of individual particles.

Check gadget. As the final stage of the computation, we check that all clauses were simultaneously satisfied by the variable assignment, using the m-input AND gadget shown in Fig. 4b and 4c. Specifically, we place the clause gadgets along a horizontal line and connect their vertical output channels to the vertical input channels of the check gadget. Again we can align the channels by extending the black areas that separate the input channels of the AND gadget, as shown in the composite diagram Fig. 5.

The intended solution sequence for the AND gadget is $\langle d, \ell, d, r \rangle$. The AND gadget is designed with the downward channel exactly m units to the right from the left wall and more than $2m$ units from the right wall, so for any particle to reach the downward channel (and ultimately, the target location), at least m particles must be presented as input. Because each input channel will present at most one particle (as argued in a clause), a particle can reach the final destination if and only if all m clauses output a particle, which is possible exactly when all clauses are satisfied by the variable assignment.

Clearly, the size of all parts of the construction is polynomial in the size of the original 3SAT instance. This completes the reduction and the NP-hardness proof.

□

We conjecture that GlobalControl-ManyParticles is in fact PSPACE-complete. One approach would be to simulate nondeterministic constraint logic [37], perhaps using a unique move sequence of the form $\langle d, \ell/r, d, \ell/r, \ldots \rangle$ to identify and "activate" a component. One challenge is that all gadgets must properly reset to their initial state without permanently trapping any particles. However, we are able to prove that a variant of this problem is PSPACE-complete in Section 5.3.

5. Matrices

The previous section investigated pathologically difficult configurations. This section investigates a complementary problem. Given the same particle and world constraints as before, what types of control are possible and economical if we are free to design the environment?

First, we describe an arrangement of obstacles that implement an arbitrary matrix permutation in four commands. Then we provide efficient algorithms for sorting matrices. We finish with potential applications.

5.1. A Workspace for a Single Permutation

For our purposes, a *matrix* is a 2D array of particles (each possibly of a different color). For an $a_r \times a_c$ matrix A and a $b_r \times b_c$ matrix B, of equal total size $N = a_r a_c = b_r b_c$, a *matrix permutation* assigns each element in A a unique position in B. Example constructions that execute matrix permutations of total size $N = 15$ and 100 are shown, respectively, in Fig. 6 and 7.

Theorem 2. *Let A and B be matrices with dimensions as above. Any matrix permutation that transforms A into B can be executed by a set of obstacles in just four moves. For N particles, the constructed arrangement of obstacles requires* $(3N + 1)^2$ *space and* $4N + 1$ *obstacles. If particles move with a speed of* v*, the required time for those four moves is* $12N/v$*. and in time* $12N/$speed.

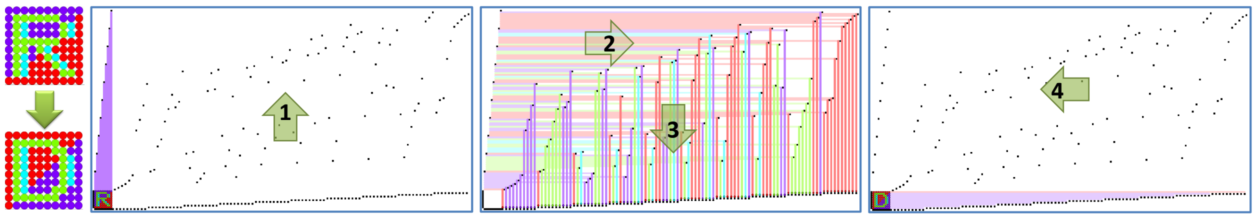


Figure 7: In this larger example with $N = 100$, the different control sections are easier to see than in Fig. 6. (1) The staggered obstacles on the left spread the matrix vertically, (2) the scattered obstacles on the upper right permute each element, and (3) the staggered obstacles along the bottom reform each row, which are collected by (4). The cycle resets every 740 iterations. See `http://youtu.be/eExZO0HrWRQ` for an animation of this gadget.

Proof. The reader is referred to Fig. 6 and 7 for examples. The move sequence is $\langle u, r, d, \ell\rangle$. The lower-left particle starts at $(0, 0)$.

Move 1: We place a_c obstacles, one for each column of A, spaced vertically $a_r - 1$ units apart, such that moving u spreads the particle array into a staggered vertical line. Each particle now has its own row. **Move 2:** We place N obstacles to stop each particle during the move r. Because each particle has a unique row, it can be stopped at any arbitrary column by its obstacle. We leave an empty column between each obstacle to prevent collisions during the next move. **Move 3:** We place N particles to stop particles during the move d, which arranges the particles in their final rows. These rows are spread in a staggered horizontal line. **Move 4:** We place a_r obstacles in a vertical line from $(-1, 1)$ to $(-1, a_c)$. Moving ℓ stacks the staggered rows into the desired permutation and returns the array to the initial position. □

By reapplying the same permutation enough times, we can return to the original configuration. The permutations shown in Fig. 6 return to the original image in two cycles, while Fig. 7 requires 740 cycles. In fact, any permutation of N elements will return to its original position after it is repeated a finite number of times [22].

For a two-color image, we can always construct a permutation that resets in 2 cycles. If the matrix has only two colors then for each entry in the matrix a permutation of the particles will either flip the color or keep the color constant in that given entry. If the permutation flips the color of a particular entry, then doing the permutation twice will flip this entry back to its original color. If the permutation keeps the color of a particular entry constant, then doing the permutation twice will also preserve the color of that entry. Performing the permutation twice always results in the original matrix. Such a permutation is an *involution*, a function that is its own inverse. An involution often does not exist for permutations on images with more than two colors.

5.2. A Workspace for Arbitrary Permutations

Theorem 2 can be exploited to generate larger sets of permutations or even all possible permutations. There is a tradeoff between the number of introduced obstacles and the number of moves required for realizing a permutation.

We start with obstacle sets that require only a small number of moves.

Theorem 3. *For an arbitrary set of k permutations of N particles, we can construct a set of $O(kN)$ obstacles, such that we can switch from a start arrangement into any of the k permutations using at most $O(\log k)$ force-field moves.*

Proof. See Fig. 8. Build a binary tree of depth $\log k$ for choosing between the permutations by a sequence of $\langle r,\ d,\ (r/\ell),\ d,\ (r/\ell),\ \ldots,\ d,\ (r/\ell),\ d,\ \ell,\ u\rangle$ with $\log k$ decisions between r and ℓ, from the initial prefix $\langle r, d\rangle$ to the final suffix $\langle d, \ell, u\rangle$. This gets the particles to the set of obstacles for performing the appropriate permutation. □

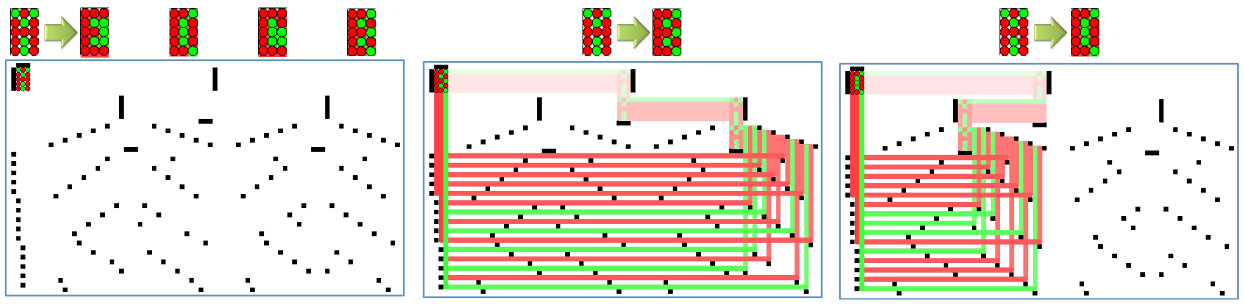


Figure 8: For any set of k fixed, but arbitrary permutations of N particles, we can construct a set of $O(kN)$ obstacles, such that we can switch from a start arrangement into any of the k permutations using at most $O(\log k)$ force-field moves. Here $k = 4$ and 'A' is transformed into 'B', C', 'D', or 'E' in eight moves: $\langle r, d, (r/\ell), d, (r/\ell), d, \ell, u\rangle$.

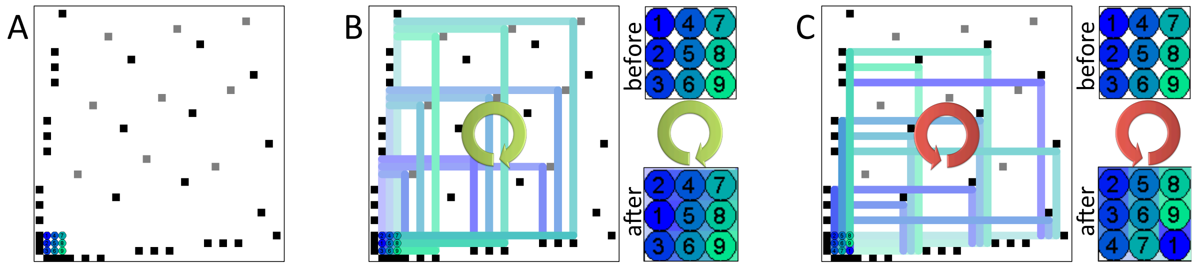


Figure 9: Repeated application of two base permutations can generate any permutation, when used in a manner similar to Bubble Sort. The obstacles in (Fig. 9A) generate the base permutation $p = (1, 2)$ in the clockwise direction $\langle u, r, d, \ell\rangle$ (Fig. 9B) and $q = (1, 2, \ldots, N)$ in the counter-clockwise direction $\langle r, u, \ell, d\rangle$ (Fig. 9C).

Corollary 4. *For any N and an arbitrary but fixed $\varepsilon > 0$, we can construct a set of $(N!)^\varepsilon$ obstacles such that any permutation of N particles can be achieved by at most $O(N \log N)$ force-field moves.*

Proof. This follows from Theorem 3. With $k = (N!)^\varepsilon/N$, $\log k$ becomes $\varepsilon \log(N!) - \log N$, i.e., $O(N \log N)$. □

Now we proceed to more economical sets of obstacles, with arbitrary permutations realized by clockwise and counterclockwise move sequences. We make use of the following easy lemma, which shows that two base permutations are enough to generate any desired rearrangement.

Lemma 5. *Any permutation of N objects can be generated by the two base permutations $p = (1, 2)$ and $q = (1, 2, \ldots, N)$. Moreover, any permutation can be generated by a sequence of length at most N^2 that consists of p and q.*

The proof is elementary and left to the reader. This allows us to establish the following result.

Theorem 6. *We can construct a set of $O(N)$ obstacles such that any $a_r \times a_c$ arrangement of N particles can be rearranged into any other $a_r \times a_c$ arrangement π of the same particles, using at most $O(N^2)$ force-field moves.*

Proof. See Fig. 9. Use Theorem 2 to build two sets of obstacles, one each for p and q, such that p is realized by the sequence $\langle u, r, d, \ell\rangle$ (clockwise) and q is realized by $\langle r, u, \ell, d\rangle$ (counterclockwise). Then we use the appropriate sequence for generating π in $O(N^2)$ moves. □

Using a larger set of base permutations allows us to reduce the number of necessary moves. Again, we make use of a simple base set for generating arbitrary permutations.

Lemma 7. *Any permutation of N objects can be generated by the N base permutations $p_1 = (1,2), p_2 = (1,3), \ldots, p_N = (1,(N-1))$ and $q = (1,2,\ldots,N)$. Moreover, any permutation can be generated by a sequence of length at most N that consists of p_i and q.*

The proof is again completely straightforward and left to the reader.

Theorem 8. *We can construct a set of $O(N^2)$ obstacles such that any $a_r \times a_c$ arrangement of N particles can be rearranged into any other $b_r \times b_c$ arrangement π of the same particles, using at most $O(N \log N)$ force-field moves.*

Proof. Use Theorem 2 to build N sets of obstacles, one each for $p_1, \ldots, p_{N-1}, q$. Furthermore, use Lemma 7 for generating all permutations with at most N different of these base permutations, and Theorem 3 for switching between these $k = N$ permutations. Then we can get π with at most N cycles, each consisting of at most $O(\log N)$ force-field moves. □

This is the best possible with respect to the number of moves in the following sense:

Theorem 9. *Suppose we have a set of obstacles such that any permutation of a rectangular arrangement of N particles can be achieved by at most M force-field moves. Then M is at least $\Omega(N \log N)$.*

Proof. Each permutation must be achieved by a sequence of force-field moves. Because each decision for a force-field move $\langle u, d, \ell, r\rangle$ partitions the remaining set of possible permutations into at most four different subsets, we need at least $\Omega(\log(N!)) = \Omega(N \log N)$ such moves. □

5.3. PSPACE*-Completeness*

In Section 4, we showed that the problem GlobalControl-ManyParticles is computationally intractable in a particular sense: given an initial configuration of movable particles and fixed obstacles, it is NP-hard to decide whether *any* individual particle can be moved to a specified location. In the following, we show that minimizing the number of moves for achieving a desired goal configuration for *all* particles is PSPACE-complete.

Theorem 10. *Given an initial configuration of (labeled) movable particles and fixed obstacles, it is* PSPACE*-complete to compute a shortest sequence of force-field moves to achieve another (labeled) configuration.*

Proof. The proof is largely based on a complexity result by Jerrum [40], who considered the following problem: Given a permutation group specified by a set of generators and a single target permutation π, which is a member of the group, what is the shortest expression for the target permutation in terms of the generator? This problem was shown to be PSPACE-complete in [40], even when the generator set consists of only two permutations, say, π_1 and π_2.

As shown in Subsection 5.2, we can realize any matrix permutation π_i of a rectangular arrangement of particles by a set of obstacles, such that this permutation π_i is carried out by a quadruple of force-field moves. We can combine the sets of obstacles for the two different permutations π_1 and π_2, such that π_1 is realized by going through a clockwise sequence $\langle u, r, d, \ell\rangle$, while π_2 is realized by a counterclockwise sequence $\langle r, u, \ell, d\rangle$. We now argue that a target permutation π of the matrix can be realized by a minimum-length sequence of m force-field moves if and only if π can be decomposed into a sequence of a total of n applications of permutations π_1 and π_2, where $m = 4n$.

The "if" part is easy: simply carry out the sequence of n permutations, each realized by a (clockwise or counterclockwise) quadruple of force-field moves. For the "only if" part, suppose we have a shortest sequence of m force-field moves to achieve permutation π, and consider an arbitrary subsequence that starts from the base position in which the particles form a rectangular arrangement in the lower left-hand corner. It is easy to see that a minimum-length sequence cannot contain two consecutive moves that are both horizontal or both vertical: these moves would have to be be in opposite directions, and we could shorten the sequence by omitting the first move. Furthermore, by construction of the obstacle set, the first move must be u or

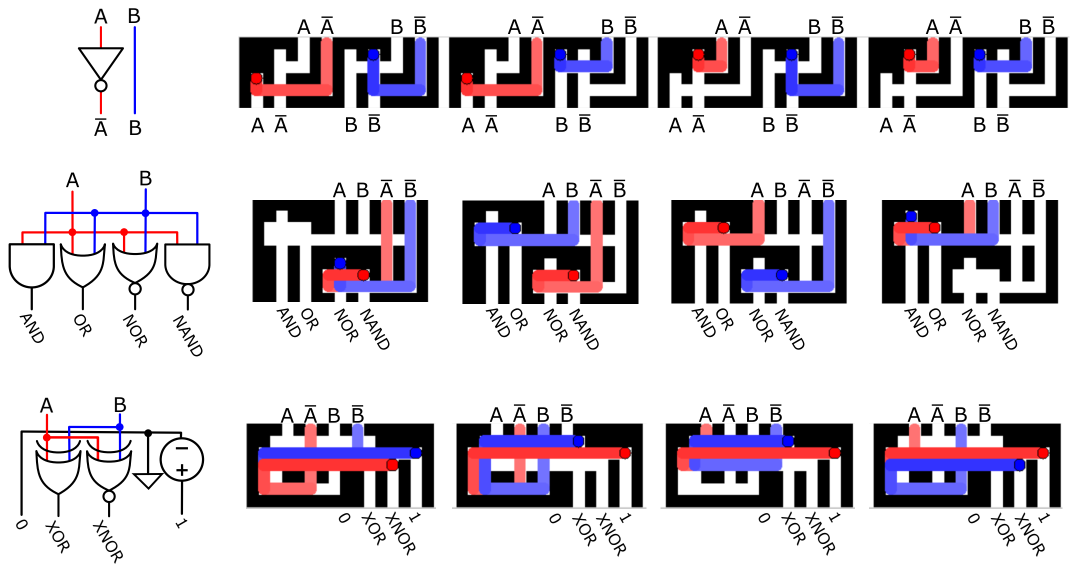


Figure 10: Schematic and diagram of dual-rail logic gates. Each gate employs the same clock sequence $\langle d, \ell, u, r\rangle$, the four inputs correspond to $A, \bar{A}, B, \bar{B}$. The top row is a NOT gate and a connector. The middle row is a universal logic gate whose four outputs are AND, NAND, OR, NOR. The bottom row gate outputs the XOR, XNOR of the inputs and constants 1 and 0. See video at `http://youtu.be/mJWl-Pgfos0` for a hardware demonstration.

r. Now it is easy to check that the choice of the first move determines the next three ones: u must be followed by $\langle r, d, \ell\rangle$; similarly, r must be followed by $\langle u, \ell, d\rangle$. Any other choice for moves 2–4 would produce a longer overall sequence or destroy the matrix by leading to an arrangement from which no recovery to a rectangular matrix is possible. Therefore, the overall sequence can be decomposed into $m = 4n$ clockwise or counterclockwise quadruples. As described, each of these quadruples represents either π_1 or π_2, so π can be decomposed into n applications of permutations π_1 and π_2. This completes the proof. □

Note that the result also implies the existence of solutions of exponential length, which can occur with polynomial space. Binary counters are particular examples of such long sequences that are useful for many purposes.

6. Limitations of Particle Logic

After considering the complexity of rearranging given arrangements of particles, i.e., *external computation*, we now turn to using the particles themselves for performing logic operations, i.e., *internal computation.* We first establish the limitations of 1×1 particles; details on designing the full range of logic gates with the help of 2×1 particles are described in the following Section 7.

6.1. Dual-Rail Logic and FAN-OUT *Gates*

In Section 4 we showed that given only obstacles and particles that move maximally in response to an input, we can construct a variety of logic elements. These include variable gadgets that enable setting multiple copies of up to n variables to be TRUE or FALSE (Fig. 3) and m-input OR and AND gates (Fig. 4). Unfortunately, we cannot build NOT gates because our system of particles and obstacles is conservative—we cannot create a new particle at the output when no particle is supplied to the input. A NOT gate is necessary to construct a logically complete set of gates. To do this, we rely on a form of *dual-rail logic*, where both the state and inverse (A and $\bar{A}$) of each signal are propagated throughout the computation. Dual-rail logic is often used in low-power electronics to increase the signal-to-noise ratio without increasing the voltage [67]. With dual-rail logic we can now construct the missing NOT gate, as shown in Fig. 10. The command sequence $\langle d, \ell, u, r\rangle$ inverts the input.

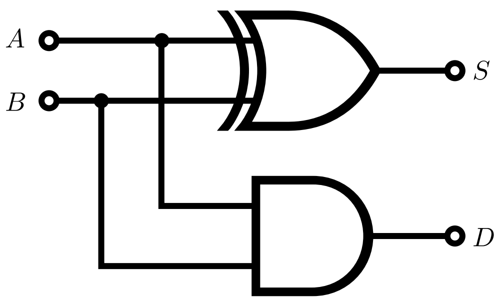


Figure 11: The half adder shown above requires two copies of A and B.

We now revisit the OR and AND gates of Fig. 4, using dual-rail logic and the four inputs $A, \bar{A}, B, \bar{B}$. The gate in the middle row of Fig. 10 can simultaneously compute AND, OR, NOR, and NAND, using the same command sequence $\langle d, \ell, u, r \rangle$ as the NOT gate. Outputs can be piped for further logic using the interconnections in Fig. 10. Unused outputs can be piped into a storage area and recycled for later use.

Dual-rail devices open up new opportunities, including XOR and XNOR gates, which are not conservative using single-rail logic. This gate, shown in the bottom row of Fig. 10 also outputs a constant 1 and 0.

Consider the half adder shown in Fig. 11. With an AND and XOR we can compactly construct a half adder. We are hindered by an inability to construct a FAN-OUT device. The *fan out* of a logic gate output is the number of gate inputs it can feed or connect to. In the particle logic demonstrated thus far, each logic gate output could fan out to only one gate. This is sufficient for *sum of products* and *product of sums* operations in CPLDs (complex programmable logic devices) but insufficient for more flexible architectures. Instead, we must take any logical expression and create multiple copies of each input. For example, a half adder requires only one XOR and one AND gate, but our particle computation requires two A and two B inputs. In the rest of this section we prove the insufficiency of unit-sized particles for the implementation of FAN-OUT gates and design a FAN-OUT gate using 2×1 particles.

6.2. Only 1×1 Particles Are Insufficient

First we provide terminology to define how particles interact with each other. We say that particle q *blocks* particle p during a move m_k, if p is prevented from reaching location $\mathbf{x} = (x, y)$ because particle q occupies this location. As a consequence, at the end of m_k, q's location is $\mathbf{x}$, while particle p's location is adjacent to $\mathbf{x}$, depending on the direction of m_k. Furthermore, the sequence of locations $\langle \mathbf{s}, \ldots, \mathbf{g} \rangle$ of a particle p from a start location $\mathbf{s}$ to its goal location $\mathbf{g}$ describes its *path.* For an unchanged sequence of force-field moves and obstacles, a particle p can only be prevented from reaching its destination by adding additional particles, or removing existing ones. We argue in the following that this will still lead to g being occupied at the end of the sequence, possibly by a different particle.

Lemma 11. *If given a fixed workspace W and a command sequence $\mathbf{m}$ that moves a particle p from start location $\mathbf{s}$ to goal location $\mathbf{g}$, then adding additional particles anywhere in W at any stage of the command sequence cannot prevent $\mathbf{g}$ from being occupied at the conclusion of sequence $\mathbf{m}$.*

Proof. Consider the effect of adding a particle to workspace W. If p never gets blocked by any particle q, then p's path remains the same. Therefore, at the conclusion of $\mathbf{m}$, p occupies $\mathbf{g}$.

Now suppose p gets blocked by q. By the definition of blocking, q prevents p from reaching some location $\mathbf{x}$ because q already occupies this location. After the blocking, the command sequence will continue and so particle q will continue on p's original path, following the same instructions and therefore ending up in the same location, $\mathbf{g}$, unless q gets blocked by yet another particle. By induction, additional particles will have the same effect. If q gets blocked by any other particle, then this particle will continue on p's original path. Thus by adding more particles, it is impossible to prevent some particle from occupying $\mathbf{g}$ at the conclusion of $\mathbf{m}$. $\square$

Corollary 12. *A* NOT *gate without dual-rail inputs cannot be constructed.*

Inputs			*Outputs*			
A	$\overline{A}$	1	A	A	$\overline{A}$	$\overline{A}$
0	1	1	0	0	1	1
1	0	1	1	1	0	0

Table 1: FAN-OUT operation. This cannot be implemented with 1×1 particles and obstacles. Our technique uses 2×1 particles.

Proof. By contradiction. A particle logic NOT gate without dual-rail inputs has one input at $\mathbf{s}$, one output at $\mathbf{g}$, an arbitrary (possibly zero) number of asserted inputs, which are all initially occupied, and an arbitrary (possibly zero) number of waste outputs.

To satisfy the NOT gate conditions given a command sequence $\mathbf{m}$, the following conditions must be satisfied.

1. If $\mathbf{s}$ is initially unoccupied, $\mathbf{g}$ must be occupied at the conclusion of $\mathbf{m}$.
2. If $\mathbf{s}$ is initially occupied, $\mathbf{g}$ must be unoccupied at the conclusion of $\mathbf{m}$.

By Lemma 11, if $\mathbf{s}$ initially unoccupied results in $\mathbf{g}$ being occupied by some particle p at the conclusion of $\mathbf{m}$, then the addition of a particle q at $\mathbf{s}$ cannot prevent $\mathbf{g}$ from being filled, resulting in a contradiction. □

This shows that dual-rail logic is necessary for the formation of NOT gates.

Additionally, we show that 1×1 particles are insufficient to produce FAN-OUT gates. To this end, we must examine the possibilities both when we add additional particles to the scenario and when we remove them.

Lemma 13. *Consider a given workspace W with a number of particles, two of which are p_1 and p_2, initially at $\mathbf{s}_1$ and $\mathbf{s}_2$. Let $\mathbf{m}$ be a command sequence that moves p_1 and p_2 to the respective goal locations $\mathbf{g}_1$ and $\mathbf{g}_2$. Then deleting either p_1 or p_2 from the original configuration results in at least one of $\mathbf{g}_1$ or $\mathbf{g}_2$ being occupied at the conclusion of $\mathbf{m}$.*

Proof. Without loss of generality, suppose we remove particle p_1.

First suppose p_2 never gets blocked by p_1. Then the removal of p_1 will not affect the path of p_2. Particle p_2 has the same number of blockings that it had before the removal of p_1 and so p_2 will follow the same path and occupy $\mathbf{g}_2$ at the conclusion.

Alternatively, suppose p_2 gets blocked by p_1 when p_1 is occupying location $\mathbf{x}$. Because p_1 is removed, it no longer occupies $\mathbf{x}$ during this move; because it was stopped in the common direction when blocking p_2, particle p_2 gets stopped by this obstacle at location $\mathbf{x}$, previously occupied by p_1. Particle p_2 now proceeds along the path previously traveled by p_1. Effectively, p_2 has replaced p_1 and follows the path until it reaches $\mathbf{g}_1$. Successive blockngs between p_2 and p_1 in the original scenario are resolved in the same manner. □

In the context of programmable matter, it is natural to consider systems in which particles are moved around, but neither created nor destroyed; such a system is called *conservative.* As it turns out, this has important consequences.

Theorem 14. *A conservative dual-logic FAN-OUT gate cannot be constructed using only 1×1 particles.*

Proof. We assume such a FAN-OUT gate exists and reach a contradiction. Consider a FAN-OUT gate W, dual-rail input locations $\mathbf{s}_a$, $\mathbf{s}_{\overline{a}}$, and dual-rail output locations $\mathbf{g}_{a_1}, \mathbf{g}_{a_2}, \mathbf{g}_{\overline{a}_1}, \mathbf{g}_{\overline{a}_2}$. Because particle logic is conservative, there must be at least one additional input location $\mathbf{s}_p$ and particle p. A FAN-OUT gate implements the truth table shown in Table 1. Given an arbitrary command sequence $\mathbf{m}$:

1. If $\mathbf{s}_a$ and $\mathbf{s}_p$ are initially occupied and $\mathbf{s}_{\overline{a}}$ vacant at the conclusion of $\mathbf{m}$, then $\mathbf{g}_{a_1}$ and $\mathbf{g}_{a_2}$ are occupied and the locations $\mathbf{g}_{\overline{a}_1}$ and $\mathbf{g}_{\overline{a}_2}$ are vacant.
2. If $\mathbf{s}_a$ is initially vacant and $\mathbf{s}_{\overline{a}}$ and $\mathbf{s}_p$ are occupied at the conclusion of $\mathbf{m}$, then $\mathbf{g}_{a_1}$ and $\mathbf{g}_{a_2}$ are vacant and the locations $\mathbf{g}_{\overline{a}_1}$ and $\mathbf{g}_{\overline{a}_2}$ are occupied.

We will now assume that condition 1, above, is the original scenario and add and subtract particles, applying Lemmas 11 and 13, to show that it is impossible to meet condition 2.

Assume condition 1. Particles a and p start at $\mathbf{s}_a$ and $\mathbf{s}_p$ respectively and at the conclusion of $\mathbf{m}$, the locations $\mathbf{g}_{a1}$ and $\mathbf{g}_{a2}$ are occupied. Now remove particle a. According to Lemma 13, either $\mathbf{g}_{a_1}$ or $\mathbf{g}_{a_2}$ must be occupied at the conclusion of $\mathbf{m}$. Suppose without loss of generality that $\mathbf{g}_{a_1}$ is filled. By Lemma 11, adding an additional particle at location $\mathbf{s}_{\overline{a}}$ cannot prevent $\mathbf{g}_{a_1}$ from being filled. However, to meet condition 2, $\mathbf{g}_{a_1}$ must be vacant, thus no such gate is possible. □

7. Device and Gate Design

Now we consider actually designing clock sequence, logic gates, and wiring, making use of 2×1 particles.

Choosing a clock sequence. The *clock sequence* is the ordered set of moves that are simultaneously applied to every particle in our workspace. We call this the clock sequence because, as in digital computers, this sequence is universally applied and keeps all logic synchronized.

A clock sequence determines the basic functionality of each gate. To simplify implementation in the spirit of Reduced Instruction Set Computing (RISC), which uses a simplified set of instructions that run at the same rate, we want to use the same clock cycle for each gate and for *all* wiring. Our early work in [8] used a standard sequence $\langle d, \ell, d, r\rangle$. This sequence can be used to make AND, OR, and XOR gates, and any of their inverses. This sequence can also be used for *wiring* to connect arbitrary inputs and outputs, as long as the outputs are below the inputs. Unfortunately, $\langle d, \ell, d, r\rangle$ cannot move any particles upwards. To connect outputs as inputs to higher-level logic requires an additional reset sequence that contains a $\langle u\rangle$ command. Therefore, including all four directions is a necessary condition for a valid clock sequence for computation that reuses gates. The shortest sequence has four commands, each appearing once. We choose the sequence $\langle d, \ell, u, r\rangle$ and by designing examples, prove that this sequence is sufficient for logic gates, FAN-OUT gates, and wiring.

This clock sequence has the attractive property of being a clockwise (CW) rotation through the possible input sequences. One could imagine our particle logic circuit mounted on a wheel rotating about an axis parallel to the ground. If the particles were moved by the pull of gravity, each counter-clockwise revolution would advance the circuit by one clock cycle. A gravity-fed hardware implementation of particle computation is shown in Fig. 12.

Limitations: The clock sequence imposes constraints on the set of reachable positions after one cycle, as illustrated in Fig. 13. If at the completion of a $\langle d, \ell, u, r\rangle$ cycle a particle is located at (s_x, s_y), the potential locations at the end of the next cycle are any locations except $([s_x+1, \infty], s_y)$, and $(s_x-1, [-\infty, s_y-1])$. For the particle to start at (s_x, s_y) after a r move, it must have been stopped by an obstacle at (s_x+1, s_y), so (s_x+1, s_y) is unreachable. Moving to the right of this obstacle requires a move of length $\lambda \neq 0$ in the down direction, followed by a r move, followed by a move of λ in the up direction. However, r is not the second move in the sequence. Because r is the final move in the clock sequence, locations directly to the right of the start location are unreachable in one cycle. Similarly, to end at any location (s_x-1, g_y) with $g_y \leq s_y$ requires both an obstacle at (s_x, g_y) and an initial down move to at least (s_x, g_y-1), but these requirements are contradictory.

A FAN-OUT *Gate.* A FAN-OUT gate with two outputs implements the truth table in Table 1. This cannot be implemented with 1×1 particles and obstacles by Corollary 14. Our technique uses 2×1 particles. A single-input, two-output FAN-OUT gate is shown in Fig. 14. This gate requires a dual-rail input, a supply particle, and a 2×1 slider. The *clockwise* control sequence $\langle d, \ell, u, r\rangle$ duplicates the dual-rail input.

The FAN-OUT gate can drive multiple outputs. In Fig. 15 a single input drives four outputs. This gate requires a dual-rail input, three supply particles, and a 2×1 slider. The *clockwise* control sequence $\langle d, \ell, u, r\rangle$ quadruples the dual-rail input. In general, an n-output FAN-OUT gate with control sequence $\langle d, \ell, u, r\rangle$ requires a dual-rail input, $n-1$ supply particles, and one 2×1 slider. It requires an area of size $4n+7\times 2n+4$.

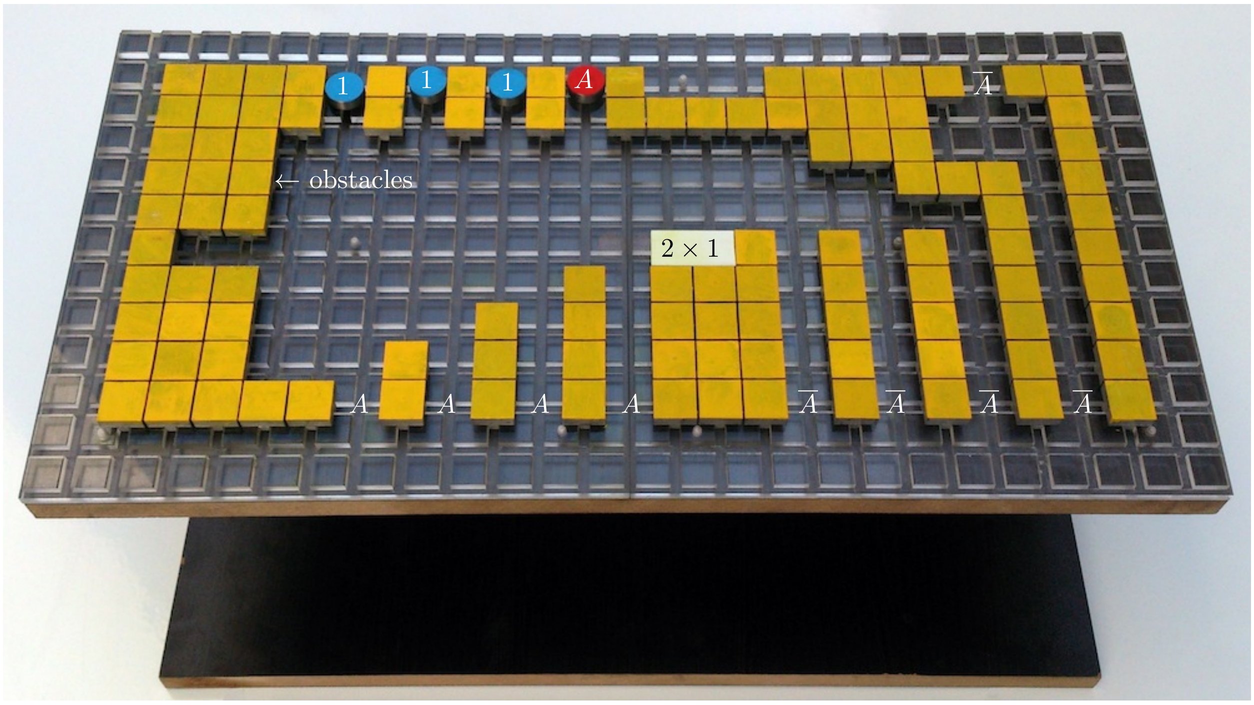


Figure 12: Gravity-fed hardware implementation of particle computation. The reconfigurable prototype is arranged using obstacle blocks (yellow) as a FAN-OUT gate using a 2×1 particle (white), three supply particles (blue), and one red dual-rail input (red). This paper proves that such a gate is impossible using only 1×1 particles. See the demonstrations in the video [9], `https://youtu.be/H6o9DTIfkn0`.

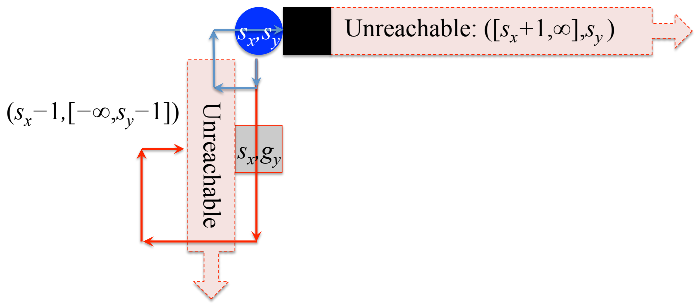
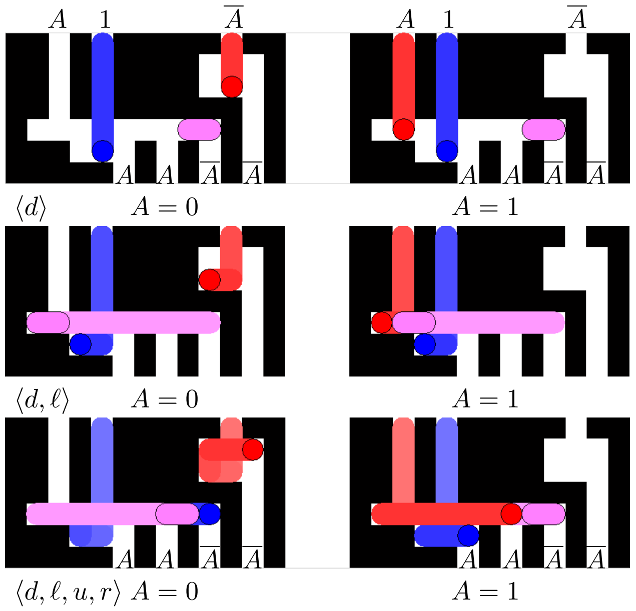

s_x, s_y
Unreachable: $([s_x+1,\infty], s_y)$
$(s_x-1, [-\infty, s_y-1])$
Unreachable
s_x, g_y

Figure 13: Two regions are unreachable in one $\langle d, \ell, u, r\rangle$ cycle.

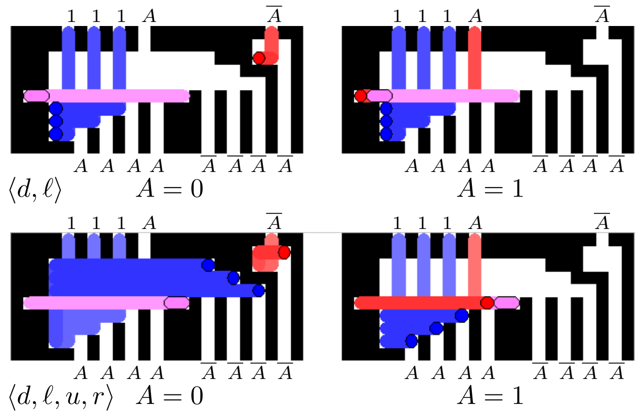
Data Storage. A general-purpose computer must be able to store data. A 2×1 particle enables us to construct a read/writable data storage for one bit. A single-bit data storage latch is shown in Fig. 16. This gate is conservative, and the memory state is given by the position of the 2×1 slider: If the slider is low the memory state is true, if the slider is high the memory state is false. This gate implements the truth table in Table 2, and has three inputs *Set*, *Clear*, or *Read*. Only one input should be true, making this a *tri-rail* input. This input can be generated by logic on two dual-rail inputs: a Set/Clear input S and a Read/Write input R, where $Set = S \wedge \overline{R}$, $Clear = \overline{S} \wedge \overline{R}$, and $Read = R$. Depending on which input is active, the *clockwise* control sequence $\langle d, \ell, u, r\rangle$ will read, set, or clear the memory. The gate has a single output M that reports the memory state after the inputs have been computed. The entire gadget requires a 16×8 area. By combining an n-out FAN-OUT gate shown in Fig. 15 with n data storage devices, we can read from an n-bit memory.

A Binary Counter. Using the FAN-OUT gate from Section 7 we can generate arbitrary Boolean logic. The half adder from Fig. 11 requires a single FAN-OUT gate, one AND, and one XOR gate.



A
1
$\overline{A}$
A
1
$\overline{A}$
A A $\overline{A}$ $\overline{A}$
A A $\overline{A}$ $\overline{A}$
$\langle d \rangle$
$A = 0$
$A = 1$
$\langle d, \ell \rangle$
$A = 0$
$A = 1$
A A $\overline{A}$ $\overline{A}$
A A $\overline{A}$ $\overline{A}$
$\langle d, \ell, u, r \rangle$
$A = 0$
$A = 1$

Figure 14: A single input, two-output FAN-OUT gate. This gate requires a dual-rail input, a supply particle, and a 2×1 slider. The *clockwise* control sequence $\langle d, \ell, u, r \rangle$ duplicates the dual-rail input.



1 1 1 A
$\overline{A}$
A A A A
$\overline{A}$ $\overline{A}$ $\overline{A}$ $\overline{A}$
$\langle d, \ell \rangle$
$A = 0$
$A = 1$
$\langle d, \ell, u, r \rangle$
$A = 0$
$A = 1$

Figure 15: The FAN-OUT gate can drive multiple outputs. Here a single input drives four outputs. This gate requires a dual-rail input, three supply particles, and a 2×1 slider. The *clockwise* control sequence $\langle d, \ell, u, r \rangle$ quadruples the dual-rail input.

Q	S	C	R	Q	M	$\overline{M}$
0	1	0	0	1	1	0
0	0	1	0	0	0	1
0	0	0	1	0	0	1
1	1	0	0	1	1	0
1	0	1	0	0	0	1
1	0	0	1	1	1	0

Table 2: A single-bit data storage latch with state Q, inputs *Set*, *Clear*, or *Read*, and outputs representing the memory state M and the inverse $\overline{M}$.

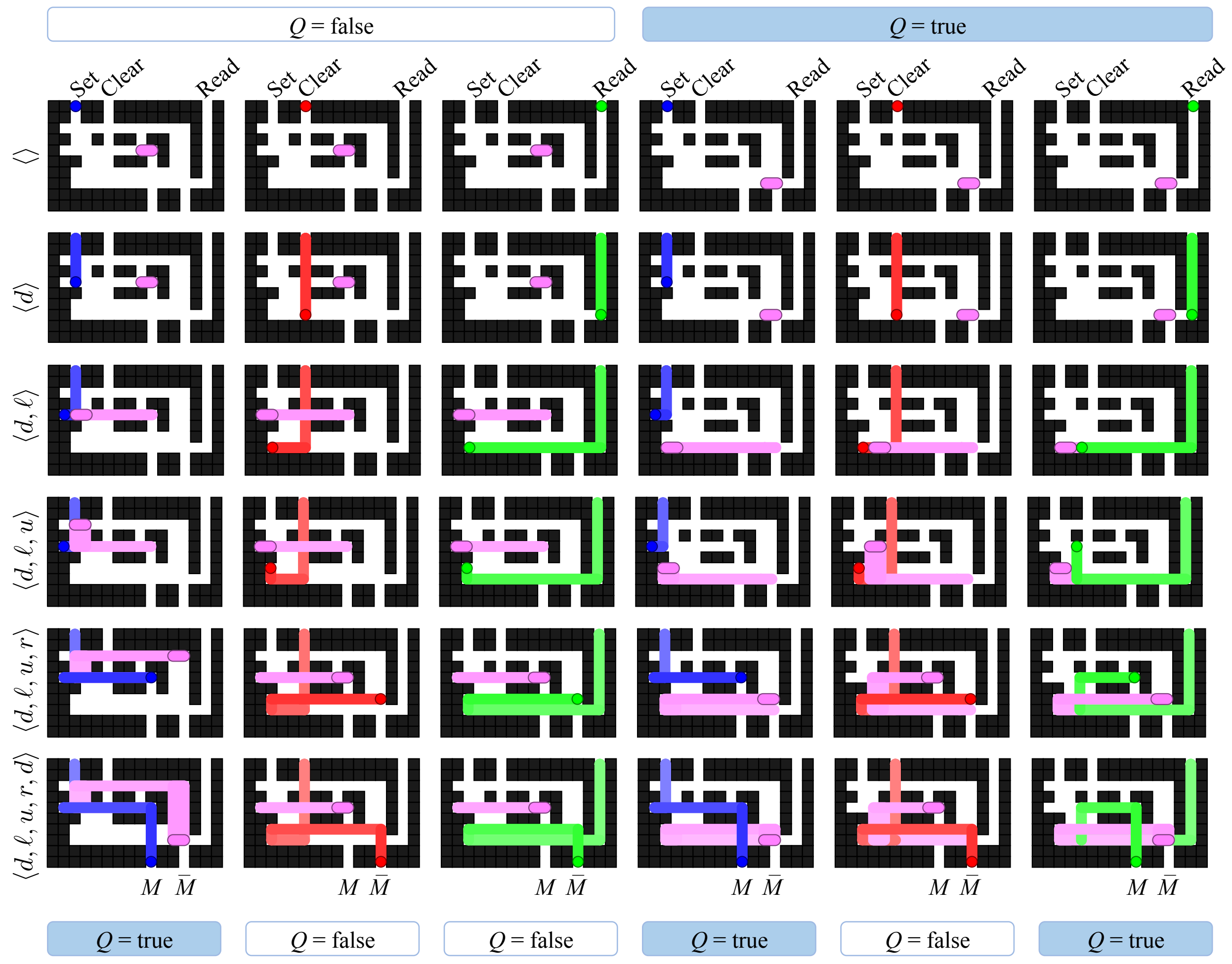


Figure 16: A conservative, flip-flop memory gadget. This gadget has a tri-rail input of *Set*, *Clear*, or *Read*; and a 2×1 state variable. The memory state Q is given by the position of the 2×1 slider: If the slider is low the memory state is true, if the slider is high the memory state is false. Depending on which input is active, the *clockwise* control sequence $\langle d, \ell, u, r\rangle$ will read, set, or clear the memory. The gate has a single output M that reports the memory state Q after the inputs have been computed. The entire gadget requires a 16×8 area.

We illustrate how several gates can be combined by constructing a binary counter, shown in Fig. 17. Six logic gates are used to implement a 3-bit counter. A block diagram of the device is shown in Fig. 18. The counter requires three FAN-OUT gates, two summers (XOR) gates, and one carry (AND) gate. Six 1×1 particles and three 2×1 particles are used. The counter has three levels of gates actuated by CW sequence $\langle d, \ell, u, r\rangle$ and requires three interconnection moves $\langle d, \ell, u, r\rangle$, for a total of 24 moves (6 cycles) per count.

Scaling Issues. Particle computation requires multiple clock cycles, workspace area for gates and interconnections, and many particles. In this section we analyze how these scale with the size of the counter, using Fig. 18 as a reference.

Gates. An n-bit counter requires $3(n-1)$ gates: n FAN-OUT gates, $n-1$ summers (XOR) gates, and $n-2$ carry (AND) gates.

Particles. We require n 1×1 particles, one for each bit, and n 2×1 particles, one for each FAN-OUT gate.

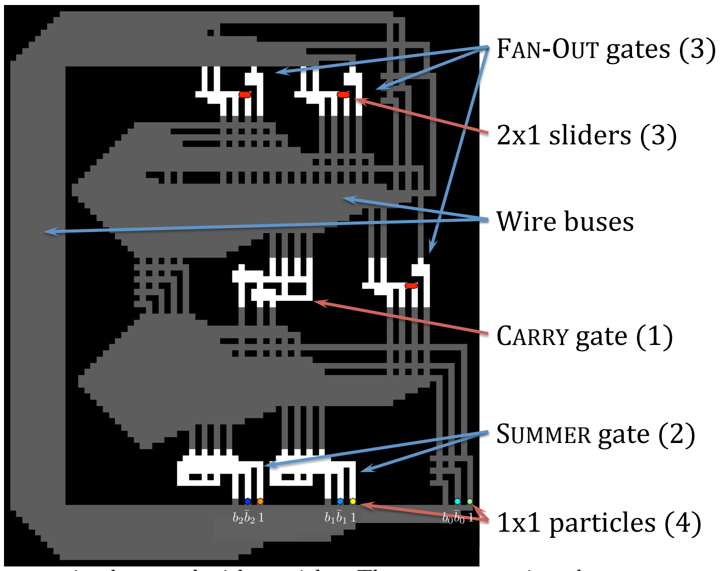


Figure 17: A three-bit counter implemented with particles. The counter requires three FAN-OUT gates, two summer gates, and one carry gate. Six 1×1 particles and three 2×1 particles are used. The gates and wire buses are actuated by the CW sequence $\langle d, \ell, u, r\rangle$. See video at `https://youtu.be/QRAOaLZjuBY?t=4m9s` for animation.

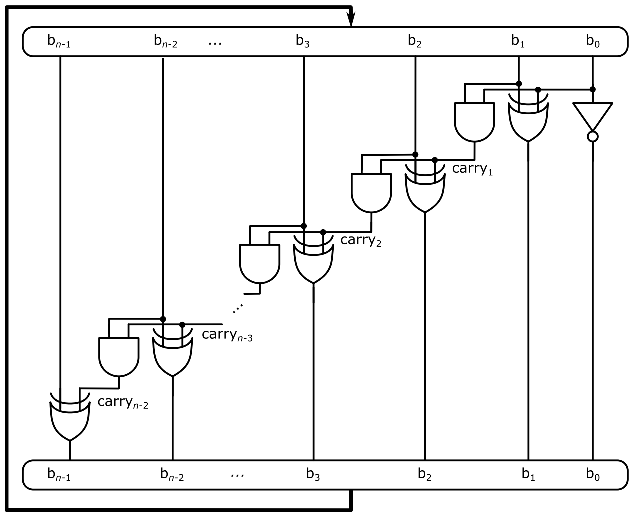


Figure 18: Gate-level diagram for an n-bit counter.

Propagation delay. The counter requires n stages of logic and n corresponding wiring stages. Each stage requires a complete clock cycle $\langle d, \ell, u, r\rangle$ for a total of $8n$ moves.

These requirements are comparable to a ripple-carry adder: the delay for n bits is n delays and requires $5(n-1)+2$ gates. Numerous other schemes exist to speed up the computation; however, using discrete gates allows us to use standard methods for translating a Boolean expression into gate-level logic. If speed were critical, instead of using discrete gates, we could engineer the workspace to compute the desired logic directly.

Optimal Wiring Schemes. With our current CW clock cycle, we cannot have outputs from the same column as inputs—outputs must be either one to the right or three to the left. Choosing one of these results in horizontal shifts at each stage and thus requires spreading out the logic gates. A more compact wiring scheme cycles through three layers that each shift right by one, followed by one layer that shifts left by three.

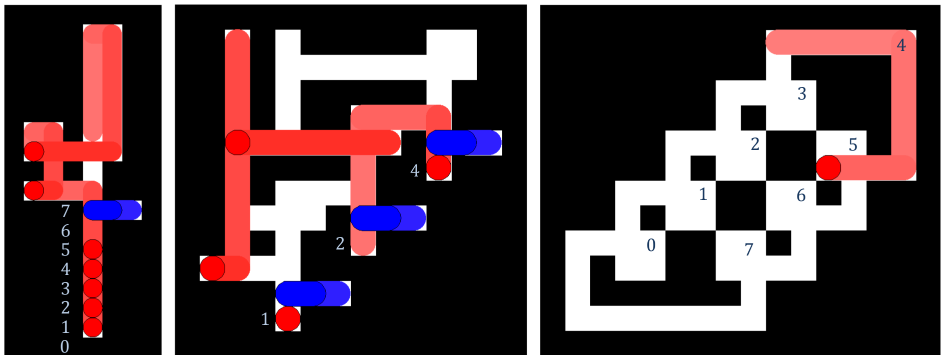


Figure 19: Custom logic can be compact. Each counter above uses the clock sequence $\langle d, \ell, u, r\rangle$ and resets after 8 cycles (32 moves). (Left) an arrangement that cyclically counts from zero to seven: seven particles, one 2×1 slider, 8×18 area. (Center) a binary counter with three bits: three particles, three 2×1 sliders, 14×14 area. (Right) a gadget that resets every 32 moves: one particle, 16×14 area.

We also want the wiring to be tight left-to-right. If our height is also limited, *wire buses*, shown in Fig. 17 provide a compact solution.

Optimized logic. The particle-computation presented in this section is general purpose, and Fig. 17 illustrates how a set of gate components can be composed to compute arbitrary logic. Single-purpose logic can often be more compact, as shown by Fig. 19, which shows three counters that use less area and fewer particles than Fig. 17.

8. Conclusion

This paper analyzes the problem of steering many particles with uniform inputs in a 2D environment containing obstacles. We design environments that can efficiently perform matrix operations on groups of particles in parallel, including a matrix permutation requiring only four moves for any number of particles. These matrix operations enable us to prove that the general motion planning problem is PSPACE-complete.

We also introduce a new model for mechanical computation. We (1) prove the insufficiency of unit-size particles for gate fan-out; (2) establish the necessity of dual-rail logic for Boolean logic; (3) design FAN-OUT gates and memory latches by employing slightly different particles; and (4) present an architecture for device integration, a common clock sequence, and a binary counter.

There remain many interesting problems to solve. We are motivated by practical challenges in steering micro-particles through vascular networks, which are common in biology. Though some are two-dimensional, including the leaf example in Fig. 1b and endothelial networks on the surface of organs, many of these networks are three dimensional. Magnetically actuated systems are capable of providing 3D control inputs, but control design poses additional challenges.

We investigate a subset of control in which all particles move maximally. Future work should investigate more general motion—what happens if we can move all the particles a discrete distance or along an arbitrary curve? We also abstracted important practical constraints, e.g., ferromagnetic objects tend to clump in a magnetic field, and most magnetic fields are not perfectly uniform.

Finally, our research has potential applications in micro-construction, microfluidics, and nano-assembly. These applications require additional theoretical analysis to model heterogeneous objects and objects that bond when forced together, e.g., MEMS components and molecular chains.

Acknowledgments

We thank Hamed Mohtasham Shad for building and testing the first experimental tilt tables that brought these algorithms to life. We acknowledge the helpful discussion and motivating experimental efforts with *T. pyriformis* cells by Yan Ou and Agung Julius at RPI and Paul Kim and MinJun Kim at Drexel University.

Preliminary versions of Section 4 and Section 5 are main topics of our paper [7] with an extra result proving the system to give rise to PSPACE-completeness in Section 5.3 from paper [8]. The particle logic in Section 6 was introduced in [8] and completed in paper [58].

This work has been partially supported by the National Science Foundation (Grant No. [IIS-1553063] and [IIS-1619278]).

References

[1] A. Abdelkader, A. Acharya, and P. Dasler. 2048 Without New Tiles Is Still Hard. In E. D. Demaine and F. Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, J. Thomas F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43(5):74–82, 2000.

[3] A. Adamatzky and J. Durand-Lose. Collision-based computing. In G. Rozenberg, T. Bäck, and J. Kok, editors, *Handbook of Natural Computing*, pages 1949–1978. Springer Berlin Heidelberg, 2012.

[4] S. Akella, W. H. Huang, K. M. Lynch, and M. T. Mason. Parts feeding on a conveyor with a one joint robot. *Algorithmica*, 26(3):313–344, 2000.

[5] S. Akella and M. T. Mason. Using partial sensor information to orient parts. *The International Journal of Robotics Research*, 18(10):963–997, 1999.

[6] A. T. Becker and T. W. Bretl. Approximate steering of a unicycle under bounded model perturbation using ensemble control. *The International Journal of Robotics Research*, 28(3):580–591, 2012.

[7] A. T. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin. Reconfiguring massive particle swarms with limited, global control. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics (ALGOSENSORS 2013)*, Lecture Notes in Computer Science, pages 51–66, 2014.

[8] A. T. Becker, E. D. Demaine, S. P. Fekete, and J. McLurkin. Particle computation: Designing worlds to control robot swarms with only global signals. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6751–6756, 2014.

[9] A. T. Becker, E. D. Demaine, S. P. Fekete, S. H. M. Shad, and R. Morris-Wright. Tilt: The Video. Designing worlds to control robot swarms with only global signals. In *31st International Symposium on Computational Geometry (SoCG'15)*, pages 16–18, 2015. Video available at `https://youtu.be/H6o9DTIfkn0`.

[10] A. T. Becker, G. Habibi, J. Werfel, M. Rubenstein, and J. McLurkin. Massive uniform manipulation: Controlling large populations of simple robots with a common input signal. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 520–527, 2013.

[11] A. T. Becker, C. Onyuksel, and T. W. Bretl. Feedback control of many differential-drive robots with uniform control inputs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2256–2262, 2012.

[12] A. T. Becker, C. Onyuksel, T. W. Bretl, and J. McLurkin. Controlling many differential-drive robots with uniform control inputs. *The International Journal of Robotics Research*, 33(13):1626–1644, 2014.

[13] A. T. Becker, Y. Ou, P. Kim, M. J. Kim, and A. Julius. Feedback control of many magnetized: Tetrahymena pyriformis cells by exploiting phase inhomogeneity. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3317–3323, 2013.

[14] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays, 2nd edition (Vol.1–4)*. A. K. Peters Ltd., 2001–2004.

[15] A. Chanu, O. Felfoul, G. Beaudoin, and S. Martel. Adapting the clinical MRI software environment for real-time navigation of an endovascular untethered ferromagnetic bead for future endovascular interventions. *Magnetic Resonance in Medicine*, 59(6):1287–1297, 2008.

[16] P.-T. Chiang, J. Mielke, J. Godoy, J. M. Guerrero, L. B. Alemany, C. J. Villagómez, A. Saywell, L. Grill, and J. M. Tour. Toward a light-driven motorized nanocar: Synthesis and initial imaging of single molecules. *ACS Nano*, 6(1):592–597, 2011.

[17] E. D. Demaine, M. L. Demaine, and J. O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *12th Annual Canadian Conference on Computational Geometry (CCCG)*, pages 211–219, 2000.

[18] E. D. Demaine and R. A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In M. H. Albert and R. J. Nowakowski, editors, *Games of No Chance 3*, pages 3–56. MSRI Publications 56, Cambridge University Press, 2009.

[19] B. R. Donald, C. G. Levey, I. Paprotny, and D. Rus. Planning and control for microassembly of structures composed of stress-engineered mems microrobots. *The International Journal of Robotics Research*, 32(2):218–246, 2013.

[20] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

[21] D. Doty, M. J. Patitz, and S. M. Summers. Limitations of self-assembly at temperature 1. In *15th International Meeting on DNA Computing and Molecular Programming (DNA15)*, pages 283–294, 2009.

[22] D. Dummit and R. Foote. *Abstract Algebra*, pages 29–31. Wiley, 3rd edition, 2009.

[23] B. Engels and T. Kamphans. On the complexity of Randolph's robot game. Technical report, Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Informatik I, 2005.

[24] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Journal on Robotics and Automation*, 4(4):369–379, 1988.

[25] S. P. Fekete, J. Hendricks, M. J. Patitz, T. A. Rogers, and R. T. Schweller. Universal computation with arbitrary polyomino tiles in non-cooperative self-assembly. In *26th ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 148–167. Society for Industrial and Applied Mathematics, 2015.

[26] S. P. Fekete and A. Kröller. Geometry-based reasoning for a large sensor network. In *22nd International Symposium on Computational Geometry (SoCG 2006)*, pages 475–476, 2006. Video available at http://compgeom.poly.edu/acmvideos/socg06video/index.html.

[27] S. P. Fekete and A. Kröller. Topology and routing in sensor networks. In *3rd International Workshop Algorithmic Aspects Wireless Sensor Networks (ALGOSENSORS 2007)*, volume 4837 of *Lecture Notes in Computer Science*, pages 6–15. Springer, 2007.

[28] S. P. Fekete, A. Kröller, D. Pfisterer, S. Fischer, and C. Buschmann. Neighborhood-based topology recognition in sensor networks. In *1st International Workshop Algorithmic Aspects Wireless Sensor Networks (ALGOSENSORS 2004)*, volume 3121 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2004.

[29] S. P. Fekete, A. Richa, K. Römer, and C. Scheideler. Algorithmic foundations of programmable matter. In *Dagstuhl Seminar 16271*, 2015.

[30] S. Floyd, E. Diller, C. Pawashe, and M. Sitti. Control methodologies for a heterogeneous group of untethered magnetic micro-robots. *The International Journal of Robotics Research*, 30(13):1553–1565, 2011.

[31] E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, 1982.

[32] D. R. Frutiger, B. E. Kratochvi, K. Vollmers, and B. J. Nelson. Magmites - wireless resonant magnetic microrobots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1770–1771, 2008.

[33] O. C. Goemans, K. Goldberg, and A. F. van der Stappen. Blades: a new class of geometric primitives for feeding 3d parts on vibratory tracks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, 2006.

[34] K. Goldberg, B. V. Mirtich, Y. Zhuang, J. Craig, B. R. Carlisle, and J. Canny. Part pose statistics: estimators and experiments. *IEEE Transactions on Robotics and Automation*, 15(5):849–857, 1999.

[35] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica*, 10(2):201–225, 1993.

[36] R. A. Hearn. The complexity of sliding-block puzzles and plank puzzles. *Tribute to a Mathemagician*, pages 173–183, 2005.

[37] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1):72–96, 2005.

[38] M. Hoffmann. Motion planning amidst movable square blocks: Push-* is NP-hard. In *Canadian Conference on Computational Geometry (CCCG 2000)*, pages 205–210, 2000.

[39] M. Holzer and S. Schwoon. Assembling molecules in ATOMIX is hard. *Theoretical Computer Science*, 313(3):447–462, 2004.

[40] M. R. Jerrum. The complexity of finding minimum-length generator sequences. *Theoretical Computer Science*, 36:265–289, 1985.

[41] J. Kahn, R. Katz, and K. Pister. Emerging challenges: Mobile networking for smart dust. *Journal of Communications and Networks*, pages 188–196, 2000.

[42] I. S. M. Khalil, M. P. Pichel, B. A. Reefman, O. S. Sukas, L. Abelmann, and S. Misra. Control of magnetotactic bacterium in a micro-fabricated maze. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5488–5493, 2013.

[43] A. Kröller, S. P. Fekete, D. Pfisterer, and S. Fischer. Deterministic boundary recognition and topology extraction for large sensor networks. In *17th ACM-SIAM Symposium on Discrete Algorithms (SODA 2006)*, pages 1000–1009, 2006.

[44] T. LaBean, E. Winfree, and J. Reif. Experimental progress in computation by self-assembly of DNA tilings. *DNA Based Computers*, 5:123–140, 1999.

[45] F. T. Leighton. *Introduction to parallel algorithms and architectures: arrays, trees, hypercubes.* Morgan Kaufmann, 1991.

[46] K. M. Lynch, M. Northrop, and P. Pan. Stable limit sets in a dynamic parts feeder. *IEEE Transactions on Robotics and Automation*, 18(4):608–615, 2002.

[47] J. Maňuch, L. Stacho, and C. Stoll. Two lower bounds for self-assemblies at temperature 1. *Journal of Computational Biology*, 17(6):841–852, 2010.

[48] S. McCourtney. *ENIAC, the triumphs and tragedies of the world's first computer*. Berkeley Trade, 2nd edition, 2001.

[49] P.-E. Meunier, M. J. Patitz, S. M. Summers, G. Theyssier, A. Winslow, and D. Woods. Intrinsic universality in tile self-assembly requires cooperation. In *25th ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 752–771, 2014.

[50] M. Moll and M. Erdmann. Manipulation of pose distributions. *The International Journal of Robotics Research*, 21(3):277–292, 2002.

[51] T. D. Murphey, J. Bernheisel, D. Choi, and K. M. Lynch. An example of parts handling and self-assembly using stable limit sets. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1624–1629, 2005.

[52] T. D. Murphey and J. W. Burdick. Feedback control methods for distributed manipulation systems that involve mechanical contacts. *The International Journal of Robotics Research*, 23(7-8):763–781, 2004.

[53] Y. Ou, D. H. Kim, P. Kim, M. J. Kim, and A. A. Julius. Motion control of magnetized tetrahymena pyriformis cells by magnetic field with model predictive control. *The International Journal of Robotics Research*, 32(1):129–139, 2013.

[54] K. E. Peyer, L. Zhang, and B. J. Nelson. Bio-inspired magnetic swimming microrobots for biomedical applications. *Nanoscale*, 5:1259–1272, 2013.

[55] P. Rendell. Turing universality of the game of life. In A. Adamatzky, editor, *Collision-Based Computing*, chapter 18, pages 513–539. Springer, London, 2002.

[56] P. Rendell. A universal Turing machine in Conway's game of life. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 764–772. IEEE, 2011.

[57] M. Rubenstein, C. Ahler, and R. Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3293–3298, 2012.

[58] H. M. Shad, R. Morris-Wright, E. D. Demaine, S. P. Fekete, and A. T. Becker. Particle computation: Device fan-out and binary memory. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5384–5389. IEEE, 2015.

[59] ThinkFun. Tilt: Gravity fed logic maze. `http://www.thinkfun.com/tilt`.

[60] A. van der Stappen, R.-P. Berretty, K. Goldberg, and M. Overmars. Geometry and part feeding. *Sensor Based Intelligent Robots*, pages 259–281, 2002.

[61] P. Vartholomeos, M. Akhavan-Sharif, and P. E. Dupont. Motion planning for multiple millimeter-scale magnetic capsules in a fluid environment. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1927–1932, 2012.

[62] T. H. Vose, P. Umbanhowar, and K. M. Lynch. Friction-induced velocity fields for point parts sliding on a rigid oscillated plate. *The International Journal of Robotics Research*, 28(8):1020–1039, 2009.

[63] T. H. Vose, P. Umbanhowar, and K. M. Lynch. Sliding manipulation of rigid bodies on a controlled 6-DoF plate. *The International Journal of Robotics Research*, 31(7):819–838, 2012.

[64] G. Wilfong. Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence*, 3(1):131–150, 1991.

[65] E. Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[66] E. Winfree, F. Liu, L. Wenzler, and N. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

[67] R. Zimmermann and W. Fichtner. Low-power logic styles: CMOS versus pass-transistor logic. *Solid-State Circuits, IEEE Journal of*, 32(7):1079–1090, 1997.