Parallel Assembly under Uniform Control Inputs

Sheryl Manzoor, Aaron T. Becker

***Abstract*— We present fundamental progress on parallel assembly using large swarms of micro- or nano-scale robots in complex environments, controlled not by individual navigation, but by a uniform global, external force. Consider a 2D grid world, in which all obstacles and robots are unit squares, and for each actuation, robots move maximally until they collide with an obstacle or another robot. In previous work, we demonstrated simulating arbitrary digital circuits. In this work we describe algorithms for designing obstacles to create arbitrary 2D structures**

I. INTRODUCTION

One of the exciting new directions of robotics is the design and development of micro- and nanorobot systems, with the goal of letting a massive swarm of robots perform complex operations in a complicated environment. Due to scaling issues, individual control of the involved robots becomes physically impossible: while energy storage capacity drops with the third power of robot size, medium resistance decreases much slower. As a consequence, current micro- and nanorobot systems with many robots are steered and directed by an external force that acts as a common control signal [1]–[7]. These common control signals include global magnetic or electric fields, chemical gradients, and turning a light source on and off.

Clearly, having only one global signal that uniformly affects all robots at once poses a strong restriction on the ability of the swarm to perform complex operations. The only hope for breaking symmetry is to use interactions between the robot swarm and obstacles in the environment. The key challenge is to establish if interactions with obstacles are sufficient to perform complex operations, ideally by analyzing the complexity of possible logical operations. In previous work [8]–[10], we were able to demonstrate how a subset of logical functions can be implemented; however, devising a fan-out gate (and thus the ability to replicate and copy information) appeared to be prohibitively challenging. In this paper, we resolve this crucial question by showing that only using unit-sized robots is insufficient for achieving computational universality. Remarkably, adding a limited number of domino-shaped objects *is sufficient* to let a common control signal, mobile particles, and unit-sized obstacles simulate a computer. While this does not imply that large-scale computational tasks should be run on these particle computers instead of current electronic devices, it establishes that future nano-scale systems are able to perform arbitrarily complex operations *as part of the physical system*, instead of having to go through external computational devices.

S. Manzoor is with the Dept. of Electrical and Computer Engineering, University of Houston, Houston, TX 70004, USA smanzoor2@uh.edu. A. Becker is with the Dept. of Electrical and Computer Engineering, University of Houston, Houston, TX 70004, USA atbecker@uh.edu.

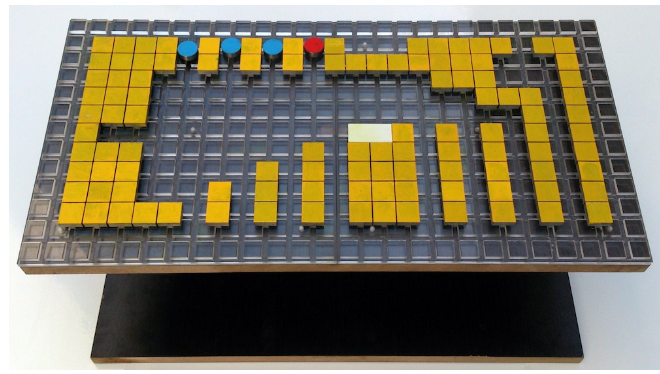

Fig. 1. Gravity-fed hardware implementation of particle computation. The reconfigurable prototype is setup as a FAN-OUT gate using a $2{\times}1$ robot (white). This paper proves that such a gate is impossible using only $1{\times}1$ robots. See the demonstrations in the video attachment http://youtu.be/EJSv8ny31r8.

A. Model

This paper builds on the techniques for controlling many simple robots with uniform control inputs presented in [8]–[10], using the following rules:

1) A planar grid *workspace* W is filled with a number of unit-square robots (each occupying one cell of the grid) and some fixed unit-square blocks. Each unit square in the workspace is either *free*, which a robot may occupy or *obstacle* which a robot may not occupy. Each square in the grid can be referenced by its Cartesian coordinates $\boldsymbol{x} = (x, y)$.
2) All robots are commanded in unison: the valid commands are "Go Up" (u), "Go Right" (r), "Go Down" (d), or "Go Left" (l).
3) Robots all move in the commanded direction until they
   a) hit an obstacle
   b) hit a stationary robot.
   c) share an edge with a compatible robot

   If a robot shares an edge with a compatible robot the two robots bond and from then on move as a unit. A *command sequence* $\boldsymbol{m}$ consists of an ordered sequence of moves m_k, where each $m_k \in \{u, d, r, l\}$ A representative command sequence is $\langle u, r, d, l, d, r, u, \ldots\rangle$. We assume the area of W is finite and issue each command long enough for the robots to reach their maximum extent.

II. Related Work

Our efforts have similarities with *mechanical computers*, computers constructed from mechanical, not electrical components. For a fascinating nontechnical review, see [11]. These devices have a rich history, from the *Pascaline*, an adding machine invented in 1642 by a nineteen-year old Blaise Pascal; 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 GridWorld implementations are rather basic, we require none of these precision elements—merely unit-size obstacles, and sliding particles sized 2×1 and 1×1 for achieving computational universality.

A. 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 [12]. 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 [13]. 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 [15]. 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.

B. 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 [16] and Demaine [17] showed tiles can be arranged to create logic gates, and used this technique to prove PSPACE complexity 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/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.

C. Other Related Work on 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 [18]–[20], which has sparked a wide range of theoretical and practical research. In this model, unit-sized pixels ("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]–[23] in the aTAM with only unit-sized pixels, very recent work [24] 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, while employing bigger particles suffices

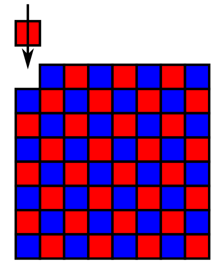

Fig. 2. Any polyomino can be constructed with two compatible robot species.

III. Construction

A. How many species are required to build an arbitrary 2D shape?

A *polyomino* is a 2D geometric figure formed by joining one or more equal squares edge to edge. Polyominoes have four-point connectivity.

Lemma 1: Any polyomino can be constructed using just two species

Proof: Label a grid with an alternating pattern like a checkerboard. Any desired polyomino can be constructed on this checkerboard, and all joints are between dissimilar species. An example shape is shown in Fig. 2. ■

The sufficiency of two species to construct any shape gives many options for implementation. The two species could correspond to any gendered connection, including electric charge, ionic charge, magnetic polarity, or hook-and-loop type fasteners.

B. Hopper Construction

Two-part adhesives react when the components mix. Placing the components in separate containers prevents mixing. Similarly, storing many particles of a single species in separate containers allows controlled mixing.

We can design *part hoppers*, containers that store similarly labelled particles. These particles will not bond with each other. The hopper shown in Fig. 3 releases one particle every cycle.

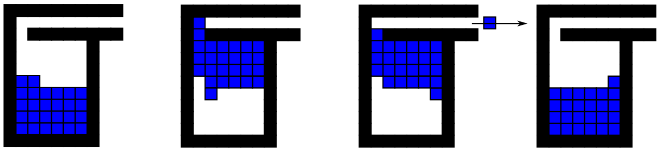

Fig. 3. This hopper is filled with similarly-labelled robots that will not combine. Every clockwise command sequence $\langle u, r, d, l \rangle$ releases one robot from the hopper.

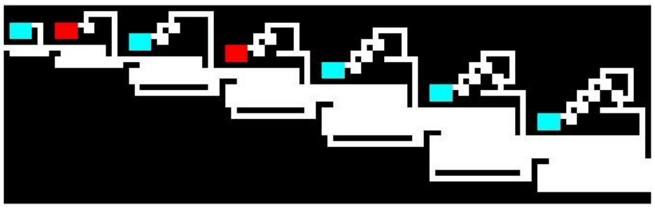

Fig. 4. A seven tile factory

C. *Part Assembly Jigs*

IV. Design Rules

```
Sheryl, add the algorithmic environment
for Build Factory
```

Different 2D part geometries are more difficult to construct than others. Fig. 8 shows three parts of varying complexity. The part of the left is shaped as a '#' symbol. Though it has an interior hole, any of the 16 particles could serve as the seed particle, and the shape could be constructed around it. The second shape is a spiral, and must be constructed from the inside-out. If the outer spiral was completed first, there would be no path to add particles to finish the interior because added particles would have to slide past compatible particles. Increasing the number of species would not solve this problem, because there is a narrow passage through the spiral that forces incoming parts to slide past the edges of all the bonded particles.

The third shape on the right is two mirrored spirals that are connected. This part cannot be assembled by adding one particle at a time, because each spiral must be constructed from the inside-out. Instead, this part must be divided into sub-assemblies that are each constructed, and then combined.

A polyomino is said to be *column convex* if each column has no holes. Similarly, a polyomino is said to be row convex if each row has no holes. A polyomino is said to be *convex* if it is row and column convex.

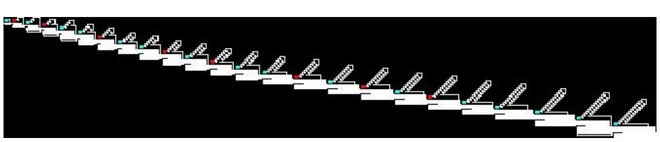

Fig. 5. A twenty four tile factory

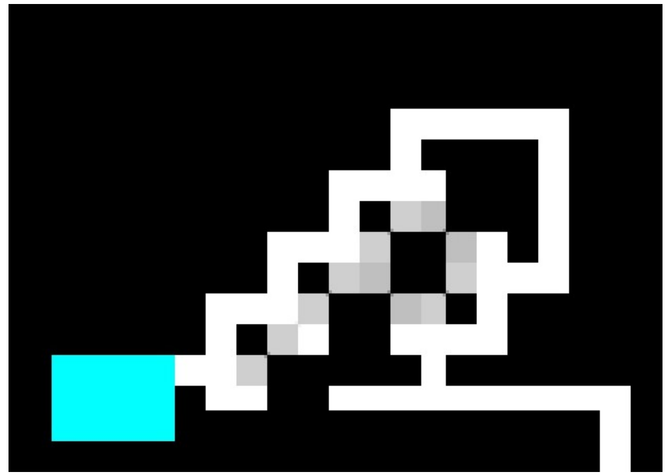

Fig. 6. Hopper with delays

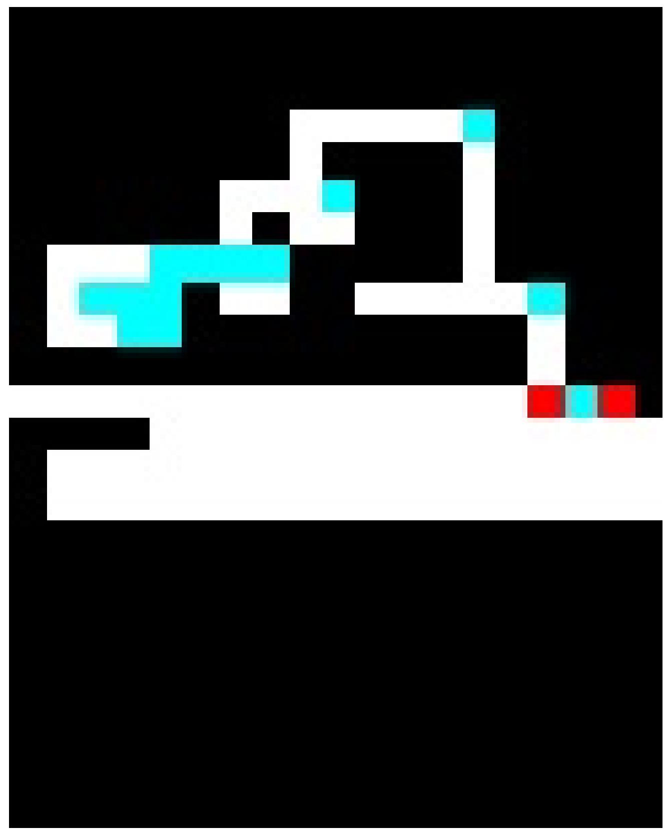

Fig. 7. Tile being attached to a three tile part by down move

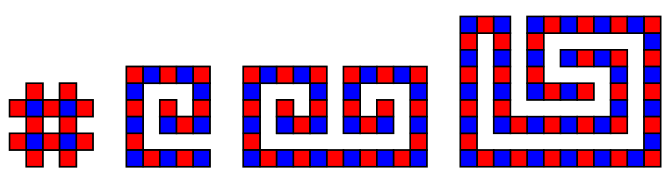

Fig. 8. Polyomino parts. Difficulty increases from left to right. The rightmost part cannot be built by additive construction.

Algorithm 1 The BuildFactory algorithm

function BUILDFACTORY($partXY$, $numCopies$)
    function FINDBUILDPATH($partXY$)
        if $false = IsPossible$ **then** return
        end if
    end function
    for $i \leftarrow 2, size(sequenceXY)$ **do**
        function FACTORYADDTILE(
$)partXYbuild$,$XYcoord$,$directions$,$partColoredArray$
return factoryObstacleAdditionArray, partXYbuild
        end function
        function CONCATFACTORIES(
$)factoryLayoutArray$,$factoryObstacleadditionArray$
return factoryLayoutArray
        end function
    end for
    function DISPLAYFACTORY(
$)factoryLayoutArray$
    end function
end function

Lemma 2: Any convex polyomino can be constructed by adding one particle at a time

Proof: Select any pixel as the *seed block*, or root node. Perform a breadth-first search starting at the seed block, labelling each block in the order they are expanded. Constructing the shape according to the ordering ensures that the polyomino is convex at every step of construction. ■

The proof of 2 assumes the existence of fixtures for assembly.

```
describe fixtures for adding one
particle at a time
```

A. Sub Assemblies for Additive Manufacture

Some non-convex polynominos cannot be constructed one particle at a time, as illustrated in Fig. 8. For instance, a polynomino consisting of a clockwise and a counterclockwise square spiral, joined at the ends with a gap of one unit between the spirals must be constructed by first assembling each spiral, and then combining the sub assemblies.

Lemma 3: Any non-convex polyomino can be disassembled into convex sub-assemblies.

Proof: How do you decompose an arbitrary polyomino into convex sub assemblies? ■

The proof of 3 requires

B. Combining Sub Assemblies

```
provide design rules for fixtures that
combine arbitrary sized sub-assemblies
```

Algorithm 2 The factoryAddTile algorithm

function FACTORYADDTILE($partXY$, $tileXY$, dir, $tileColor$, $numCopies$, pos)
    function HOPPER($tileColor$,$numCopies$,4,pos)
return hopper, hopper˙size
    end function
    if $dir =' d'$ **then** $maxpartx = max(partXY(:, 2));$
        if $tileXY(1, 2) <= maxpartx$ **then**
            function DOWNDIR($hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        else
            function LEFTDIR(
$)hopper$,$partXY$,$tileXY$
return factoryObstacleAdditionArray, align
            end function
        end if
    end if
    if $dir =' l'$ **then** $maxparty = max(partXY(:, 1));$
        if $tileXY(1, 1) <= maxparty$ **then**
            function LEFTDIR(
$)hopper$,$partXY$,$tileXY$
return factoryObstacleAdditionArray, align
            end function
        else
            function UPDIR($hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        end if
    end if
    if $dir =' u'$ **then** $minpartx = min(partXY(:, 2));$
        if $tileXY(1, 2) >= minpartx$ **then**
            function UPDIR($hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        else
            function RIGHTDIR(
$hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        end if
    end if
    if $dir =' r'$ **then** $minparty = min(partXY(:, 1));$
        if $tileXY(1, 1) >= minparty$ **then**
            function RIGHTDIR(
$hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        else
            function DOWNDIR($hopper$,$partXY$,$tileXY$)
return factoryObstacleAdditionArray, align
            end function
        end if
    end if
    return partXYupdated, factoryObstacleAdditionArray, align, hopper˙size
end function
[2]

Algorithm 3 The FindBuildPath algorithm

function FINDBUILDPATH($partXY$)
for $m = 1 : partXY(:, 1)$ **do**
$Start = partXY(m, :)$;
function DEPTHFIRSTSEARCH($partXY$,$Start$) return Output, Seq, Tmppart
end function
$partColored = labelColor(Tmppart(:, :, 1))$;
$partialAssembly = zeros(size(Tmppart(:, :, 1), 1), size(Tmppart(:, :, 1), 2))$;
$partialAssembly(Output(1, 1), Output(1, 2)) = 1$;
$dirsFinal = size(partXY, 1) - 1$;
$dirsFinal = char(dirsFinal)$;
$dirs2 = ['d';' l';' u';' r']$;
for $i = 2 : size(Output, 1)$ **do**
for $j = 1 : 4$ **do**
function CHECKPATH1TILE($partialAssembly$,$Output(i, :)$,$dirs2(j, :)$,$partColored$) return move
end function
if $strcmp(move,' true')$ **then** $partialAssembly(Output(i, 1), Output(i, 2)) = 1$; $dirsFinal(i - 1, :) = num2str(dirs2(j, :))$; $break$;
end if
end for
if $strcmp(move,' false')\&m = size(partXY, 1)$ **then** $clearoutputseqtmp_partpartColoredpartialAssemblydirsFinal$; $break$;
end if
if $strcmp(move,' true')\&i == size(Output, 1)$ **then** $foundPath = true$;
end if
end for
if $foundPath == true$ **then** $sequence = Output$; $dirs = dirsFinal$; $partColoredArray = partColored$; $break$;
end if
end for
return foundPath, sequence, dirs, partColoredArray
end function

[3]

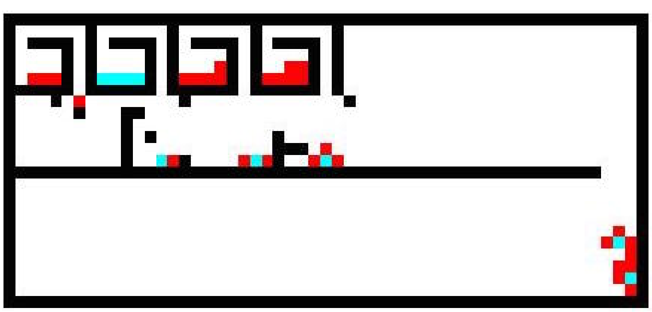

Fig. 9. A factory hand-designed for constructing a ⊥ tetromino using four hoppers and a clockwise global input.

V. EXPERIMENT

VI. CONCLUSION

In this paper we

This work, along with [8]–[10], introduces a new model for additive assembly. Interesting applications will aim at nanoscale and microfluidics work.

REFERENCES

[1] 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*, vol. 32, no. 2, pp. 218–246, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/2/218.abstract

[2] 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*, vol. 6, no. 1, pp. 592–597, Feb. 2011.

[3] H.-W. Tung, D. R. Frutiger, S. Panè, and B. J. Nelson, "Polymer-based wireless resonant magnetic microrobots," in *IEEE International Conference on Robotics and Automation*, May 2012, pp. 715–720.

[4] E. Diller, J. Giltinan, and M. Sitti, "Independent control of multiple magnetic microrobots in three dimensions," *The International Journal of Robotics Research*, vol. 32, no. 5, pp. 614–631, 2013. [Online]. Available: http://ijr.sagepub.com/content/32/5/614.abstract

[5] W. Jing, N. Pagano, and D. Cappelleri, "A tumbling magnetic microrobot with flexible operating modes," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 5514–5519.

[6] 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," *Int. J. Rob. Res.*, vol. 32, no. 1, pp. 129–139, Jan. 2013.

[7] D. de Lanauze, O. Felfoul, J.-P. Turcot, M. Mohammadi, and S. Martel, "Three-dimensional remote aggregation and steering of magnetotactic bacteria microrobots for drug delivery applications," *The International Journal of Robotics Research*, 11 2013. [Online]. Available: http://ijr.sagepub.com/content/early/2013/11/11/0278364913500543

[8] A. Becker, E. Demaine, S. 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)*, Sep. 2013.

[9] A. Becker, E. Demaine, S. Fekete, and J. McLurkin, "Particle computation: Controlling robot swarms with only global signals," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[10] A. Becker, E. D. Demaine, S. P. Fekete, G. Habibi, and J. McLurkin, "Reconfiguring massive particle swarms with limited, global control," in *Algorithms for Sensor Systems*, ser. Lecture Notes in Computer Science, P. Flocchini, J. Gao, E. Kranakis, and F. Meyer auf der Heide, Eds. Springer Berlin Heidelberg, 2014, pp. 51–66. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-45346-5_5

[11] S. McCourtney, *ENIAC, the triumphs and tragedies of the world's first computer*. United States of America: Walker Publishing, 1999.

[12] A. Adamatzky and J. Durand-Lose, "Collision-based computing," in *Handbook of Natural Computing*, G. Rozenberg, T. Bäck, and J. Kok, Eds. Springer Berlin Heidelberg, 2012, pp. 1949–1978. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-92910-9_58

[13] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, no. 3-4, pp. 219–253, 1982. [Online]. Available: http://dx.doi.org/10.1007/BF01857727

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

[15] A. Adamatzky and P. Rendell, *Turing Universality of the Game of Life*. Springer London, 2002, pp. 513–539. [Online]. Available: http://dx.doi.org/10.1007/978-1-4471-0129-1_18

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

[17] E. D. Demaine and R. A. Hearn, *Games of No Chance 3*. Mathematical Sciences Research Institute Publications, Cambridge University Press, 2009, vol. 56, ch. Playing Games with Algorithms: Algorithmic Combinatorial Game Theory, pp. 3–56. [Online]. Available: http://arXiv.org/abs/cs.CC/0106019

[18] E. Winfree, "Algorithmic self-assembly of DNA," Ph.D. dissertation, California Institute of Technology, June 1998.
[19] E. Winfree, F. Liu, L. Wenzler, and N. Seeman, "Design and self-assembly of two-dimensional DNA crystals," *Nature*, vol. 394, pp. 539–544, 1998.
[20] T. LaBean, E. Winfree, and J. Reif, "Experimental progress in computation by self-assembly of DNA tilings," *DNA Based Computers*, vol. 5, pp. 123–140, 1999.
[21] D. Doty, M. J. Patitz, and S. M. Summers, "Limitations of self-assembly at temperature 1," in *Proceedings of The Fifteenth International Meeting on DNA Computing and Molecular Programming (Fayetteville, Arkansas, USA, June 8-11, 2009)*, 2009, pp. 283–294.
[22] J. Maňuch, L. Stacho, and C. Stoll, "Two lower bounds for self-assemblies at temperature 1," *Journal of Computational Biology*, vol. 17, no. 6, pp. 841–852, 2010.
[23] 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 *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 2014), (Portland, OR, USA, January 5-7, 2014)*, 2014, pp. 752–771.
[24] 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 *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015, to appear.