Small Turing Machines in Tile Automata

Jacob Hendricks* Derek Stedman†

May 14, 2022

Abstract

Tile Automata is a rich model of self-assembly that combines features from both cellular automata and the 2-Handed Model(2HAM) from self-assembly. In this paper, we explore the power of the Tile Automata model and its capabilities of perfoming computation with small assemblies. We first introduce the concept of assembly complexity and then show a construction for simulating Turing Machines that have low assembly complexity. We then analyze what computations in terms of time and space in which this construction has lower assembly complexity than traditional Turing-complete constructions in Tile Automata.

1 Introduction

Self-assembly is a rich area of research with its ability of its models to reflect real world systems while also serving as a grounds for purely theoretical results. Tile Automata is a recent model of self assembly that ties together traditional models of self assembly such as 2HAM with cellular automata by adding state change rules. This powerful new model captures the powerful aspects of self assembly while also simplifying it to allow one to avoid the implementation details of other models of self assembly. This allows Tile Automata to capture the essential aspects of self assembly and connect the different models within self assembly.

1.1 Previous Work

Tile Automata was first introduced in [4]. In this paper, the authors showed that in freezing Tile Automata, Tile Automata in which a tile cannot repeat its state, is able to simulate normal Tile Automata.

Another paper dealing with Tile Automata being simulated is [3]. The authors show that the Signal Passing Self-Assembly model is able to simulate Tile

*Department of Computer Science and Information Systems University of Wisconsin - River Falls, River Falls Email: jacob.hendricks@uwrf.edu

†Department of Computer Science and Information Systems University of Wisconsin - River Falls, River Falls Email: derek.stedman@my.uwrf.edu

Automata. Futhermore, they connect this result with a result from [1] that showed that models of programmable matter are able to be simulated with Tile Automata. Thus they were able to show that the Signal Passing Self-Assembly model is able to simulate the models of programmable matter.

In [2], the authors look at simulating Turing Machine with Tile Automata with various restrictions such as 1-Dimensional Tile Automata. They then use those results to show a connection between Unique Assembly Verification and the Busy Beaver problem.

2 Model and Definitions

In this paper use the definitions for self-assembly and Tile Automata terms from [4]

Furthermore, we define the concept of assembly complexity.

Definition 2.1 (Assembly Complexity)**.** *Let PA be the set of all producible assemblies in Tile Automata system T. Then the assembly complexity of T is defined as* $AC(T) = \{|x| \mid \forall y \in PA, x \in PA(|x| > |y|)\}$*. Basically, the assembly complexity is the size of the largest producible assembly in a certain Tile Automata system.*

3 Small Turing Machine

The power of Tile Automata allows it simulate a Turing Machine even with various restrictions. In this construction, we simulate a Turing machine in a way that minimizes assembly coplexity. First we provide the construction that can simulate any Turing Machine. Next, we will delve into the properties of this consruction.

Theorem 1 (Construction)**.** *Given any Turing Machine* $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$ *with time t and space s, there exists a Tile Automata system* $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta, \tau)$ *that can simulate M suct that* $|\Sigma_{TA}| = \mathcal{O}(|Q||\Gamma|)$ *and* $|\Delta| = \prime(|\delta|)$ *and such that the assembly complexity of T is log(t)+log(s).*

Proof. Given a Turing Machine Turing Machine $M = (Q, \Sigma, \Gamma, \delta, q_a, q_r, q_s)$ with time t and space s, we construct the Tile Automata system $T = (\Sigma_{TA}, \Pi, \Lambda, \Delta, \tau)$ with the following. We first take an informal look at the construction before giving rigorious proofs of its key properties and showing it capapable of simulating a Turing Machine. □

3.1 Assemblies

There are three main types of assemblies in this construction: tape assemblies, updater assemblies, and the head assembly. Each tape assembly is an assembly that represents one cell of the turing machine. It consists of the location of the cell, the time the turing machine is at, and the symbol that is on that cell of

the tape. The location and time are encoded using "bumps " where a bump in one place means a 1 and a bump in another means 0. An example of this is shown below(Insert picture). Number encoding is gone into further in Section 3.3. These bumps encode the time and location in binary. Also, these bumps only allow other assemblies to bind that having the corresponding complement bumps. Additionally, the encoded time on the tape assembly is able to be incremented by one.

The next type of assembly is the head assembly. The head assembly is an assembly that also has its location and the time of turing machine encoded but with the complement series of bumps as the tape assemblies. This allows the head assembly to bind with tape assemblies. Also, the head assembly has the state of the Turing Machine encoded in a tile. The head assembly is able to have its time incremented and its location incremented or decremented. Incrementing or decrementing its location allows the head assembly to move left or right on the tape.

The final type of assembly in this construction is the updater assembly. The function of the updater assemblies is to increment the tape assemblies after each step of the Turing Machine. The updater assemblies also have a location and time encoded with its encoding the complement of the tape assemblies. The location of the updater assemblies are able to be incremented and decremented . Two updater assemblies are set into motion by the head assembly after the head assembly is done at the current time. One is to update times to the left of the head and another is to update the assemblies to the right of the head. They are set to the current time and one is set to the location of head incremented by one and one is set to the current location decremented by one(if possible). These updater assemblies bind to the tape assembly with the corresponding location and time and start a series of state changes that increment the time on the tape assembly and increment or decrement themselves depending on if they are the left or right updater. In this way all the tape assemblies are at the correct time for the computation. Using these three types of assemblies , we are able to simulate a Turing Machine with Tile Automata with assembly complexity of log(t)+log(s).

3.2 Overview of Simulation

The beginning of the simulation starts with the input symbols encoded in the initial assemblies that are tape assemblies. The initial state of the head in encoded in the initial assembly that is a head assembly. At each step in the Turing Machine, a few things happen in the Tile Automata system. First the head assembly binds to the tape location with the corresponding time and location. The state tile of the head assembly interacts with the symbol tile of the tape assembly and through a transition of states, changes the symbol on the symbol tile and the state of the state tile. These changes correspond to the symbol written and change of the head state in the Turing Machine. The head assembly than sets into motion two updater assemblies with one updater assembly at the head location plus one and with one updater assembly at the head location

minus one. These assemblies are set to the current time of the head assembly. The updater location at the right of the head(at the head location plus one) attaches to the tape assembly at its time and location, starts a series of state changes that increments the tape assembly's time, then detaches and moves to the right(increments its location by one). The updater assembly to the left does the same but moving to the left instead. These updater assemblies continue until getting a location that does not have a tape cell at that location and time or puts itself into a condition in which it can no longer bind to other assemblies (this happens in an assembly at location 1(0?) that decrements). Going back to the head assembly, it start the process of its time and the tape's time being incremented before detaching and incrementing or decrementing it location depending on if the head of the Turing Machine moves left or right. Once the location and time of the head assembly are done being changed, that step of the Turing Machine is done and the whole process repeats. If the head tape ever moves to a location that does not have tape assembly with that location, it starts a process that creates a tape cell with that location, at the time of the head assembly, with a blank symbol. Once the tape assembly is finished being made, the interaction between the head assembly and tape assembly continues as usual. The simulation of the Turing Machine continues like this until the state tile of the head assembly is put into an accept or reject state. This ends the simulation of the Turing Machine.

3.3 Number Encoding

Numbers are encoded for time and location using a big-endian series of bumps. For the tape assembly, Each number consists of three tiles with two tiles as the floor with the third tile on top of one of the two tiles in order to create a "bump". Being on top of the left tile makes it a 0 and being on top of the right of the two tiles makes it a 1. This process is reversed to encode numbers for the head and updater assemblies.

Numbers are incremented by a one tile being placed by the rightmost series of tiles representing a number. Through state changes and tiles being allowed to bind, the one is carried through the series of bumps changing "1" into "0" until it reaches a "0" and changes it to a "1". If it reaches the end of the series of bumps without reaching a "0", the assembly is extended at that point and a "1" bump is placed. Decrementing is done in a similiar manner.

The tiles, states, affinifty functions, and state transition rules for the encoding and incrementing/decrementing are shown in Figure(INSERT FIGURE).

3.4 States

The states for encoding numbers and incrementing/decrementing are the same for all Tile Automata systems with this construction and found in Section 3.3. Next there are the head state states which we call H. Let $H = \{h_x \mid x \in Q\}$. Also, there are two states for each each head state, h_r and h_l, which correspond whether the head will be moving left or right. Finally there are the tile assembly

symbol tile states which we call S. Let $S = \{s_y \mid y \in \Sigma\}$ The tiles in H and S have sufficient afinity to bind with the tiles in the head assembly and tile assembly respectively.

3.5 Transition Rules

We create a transition rule TR for each element of δ as follows. Let $x \in \delta$. Then $TR = \{h_{x_{qb}}, h_{x_{qa}}, s_{x_{\Gamma b}}, s_{x_{\Gamma a}}, \perp\}$, with $h_{x_{qb}}$ is the head state tile that corresponds with the head state of the turing machine before being changed in the transition rule, etc.(Probably will rewrite this part).

There are also the transition rules for incrementing/decrementing, and the binding/debinding of assemblies.

3.6 Accept/Reject

TDB depending on how we want system to look at end(All tiles are either accept or reject tiles or some other ending)

3.7 Input

The set of initial assemblies can be split into two groups: the head assembly and the tile assemblies. The tile assemblies are constructed in the following manner. Say M has input string x. All initial tile assemblies will have time 0. Location and the symbol tile are encoded in the following way. The tile assembly with location 0 will have the leftmost symbol in x. Following tile assemblies will have increase in location and and go right that much on x for the state on the symbol tile of the tile assembly. The head assembly will have time 0, location where the head of the Turing Machine starts, and head state tile state of h_{q_0}.

3.8 Terminal Assemblies

TDB Accept/Reject section

3.9 Assembly Complexity

The size of assemblies in this Tile Automata system are based on the location of where the assembly is on the tape and the time step that the Turing Machine is on. The largest location is based on the space, s, used by the Turing Machine and the largest time is based on the time, t, of the Turing Machine. The numbers representing time and location are encoded with log that number of tiles. This means that the largest producible assembly has size $log(s) + log(s)$. This means that this construction has lower assembly complexity than a Tile Automata system that simulates a Turing Machine with s tiles when $t < \frac{2^s}{t}$.

4 Computational Complexity Stuff

4.1 Space

4.2 Time

5 Conclusion

In this paper we showed that there are classes of problems that can simulated by a Tile Automata system in assembly complexity less than the space used by the Turing Machine. We showed a construction of a Tile Automata system that simulates a Turing Machine with log(time) + log(space) assembly complexity. We also showed what class of problems this construction is able to simulate with better assembly complexity than space. Futher directions for future work can directly build on this work. Are there other ways of simulating a Turing Machine in Tile Automata that require even less assembly complexity for similiar classed of problems or with the same assembly complexity for a wider range of problems? The original goal of this paper was to find a way of simulating a Turing Machine in Tile Automata with an assembly complexity asymptotically less than the space of the Turing Machine. Clearly we have failed to do so, but is this possible? Or is there a proof of its impossibility? Another direction for future research is around the parallel nature of Tile Automata. This is a powerful component of Tile Automata that deserves further study. The parallel nature of Tile Automata was one of the main hinderences in making Tile Automata systems with small assembly complexity, but also allowed for the time Tile Automata system to be the similiar to the time of the simulated Turing Machine. Further exploring the parallel nature of Tile Automata would give further insight into its capabilities as well as its limitations.

References

[1] J. C. Alumbaugh, J. J. Daymude, E. D. Demaine, M. J. Patitz, and A. W. Richa. Simulation of programmable matter systems using active tile-based self-assembly. In *International Conference on DNA Computing and Molecular Programming*, pages 140–158. Springer, 2019.

[2] D. Caballero, T. Gomez, R. Schweller, and T. Wylie. Verification and computation in restricted tile automata. *Natural Computing*, pages 1–19, 2021.

[3] A. A. Cantu, A. Luchsinger, R. Schweller, and T. Wylie. Signal passing self-assembly simulates tile automata. 2020.

[4] C. Chalk, A. Luchsinger, E. Martinez, R. Schweller, A. Winslow, and T. Wylie. Freezing simulates non-freezing tile automata. In *International Conference on DNA Computing and Molecular Programming*, pages 155–172. Springer, 2018.