Fuel Efficient Computation in Passive Self-Assembly

Robert Schweller*  Michael Sherman†

Abstract
In this paper we show that passive self-assembly in the context of the tile self-assembly model is capable of performing fuel efficient, universal computation. The tile self-assembly model is a premiere model of self-assembly in which particles are modeled by four-sided squares with glue types assigned to each tile edge. The assembly process is driven by positive and negative force interactions between glue types, allowing for tile assemblies floating in the plane to combine and break apart over time. We refer to this type of assembly model as passive in that the constituent parts remain unchanged throughout the assembly process regardless of their interactions. A computationally universal system is said to be fuel efficient if the number of tiles used up per computation step is bounded by a constant. Work within this model has shown how fuel guzzling tile systems can perform universal computation with only positive strength glue interactions [33]. Recent work has introduced space-efficient, fuel-guzzling universal computation with the addition of negative glue interactions and the use of a powerful non-diagonal class of glue interactions [20]. Other recent work has shown how to achieve fuel efficient computation [28] within active tile self-assembly. In this paper we utilize negative interactions in the tile self-assembly model to achieve the first computationally universal passive tile self-assembly system that is both space and fuel-efficient. In addition, we achieve this result using a limited diagonal class of glue interactions.

1 Introduction
Self-assembly is the process by which systems of simple objects organize themselves through local interactions into larger, more complex objects. There are perhaps two categories of self-assembly: passive and active. In passive self-assembly the objects of the system are simple, stagnant particles that interact simply by surface chemistry and geometry. In contrast, objects within an active self-assembly model may be permitted to move, rotate, adjust state, or add and remove bonding domains based on local interactions. Put another way, active self-assembly extends traditional passive self-assembly by considering the objects of the system to be simple robots with abilities that vary according to the particular model considered.

The study of passive self-assembly is important for multiple reasons. First, many theoretical models of active self-assembly do not currently have satisfactory implementations at the nanoscale. In contrast, many computationally interesting passive tile self-assembly constructions are seeing experimental success based on DNA implementations [7, 24, 30]. Further, it is plausible that many active self-assembly models will see implementation through constructions built up within passive self-assembly models. For example, the system proposed in this paper which implements a space-efficient, fuel-efficient universal computation system might be modified to serve as the internal machinery for a larger, active component, thus potentially allowing for a passive self-assembly system to simulate the behavior of a more powerful active system. Finally, it is important to understand the power and limits of passive self-assembly to better understand when active components are truly necessary.

In this paper we focus on an established model of passive self-assembly, the tile assembly model (TAM). Monomers in the TAM are unit squares with glue types assigned to edges. Self-assembly is driven by a large (infinite) number of copies of a set of tile types floating about, bumping into one another in the plane, and potentially sticking together when glue affinities exceed some set threshold. While simple, the TAM has been extensively studied [16, 18, 14, 16, 10, 22, 23, 25, 27, 31, 32] and shown to be capable of universal computation [33], and even recently shown to be intrinsically universal [21] in the case of the abstract TAM.

*Department of Computer Science, University of Texas - Pan American, rtschweller@utpa.edu This author’s research was supported in part by National Science Foundation Grant CCF-1117672.
†Department of Computer Science, University of Texas - Pan American, mjsherman@utpa.edu This author’s research was supported in part by National Science Foundation Grant CCF-1117672.
Within the TAM, we consider the problem of improving existing universal computation results by obtaining fuel-efficiency. A TAM system that simulates a Turing machine is said to be fuel-efficient if only a constant number of tiles are used up per computation step. This requirement, or close approximations to it, are of fundamental importance for the implementation of scalable molecular computers. Unfortunately, all passive variants of the TAM have failed to yield fuel-efficient universal computation.

To address this fuel-deficiency, we consider the TAM with the added power of negative glue interactions. The power of negative force interactions stems from the possibility of stable assemblies coming together with strong affinities and yet producing unstable products that subsequently break apart into pieces that are different from the original pair of assemblies. By careful engineering, this property can be harnessed to allow assemblies to attach to specific locations of large assemblies, surgically pry off and replace small portions of the assembly, and finally detach the replacement mechanism and repeat the process. If you imagine that the large assembly represents the tape of a Turing machine, and that the location of attachment occurs at the current head location of the machine, you get a high level overview of our Turing machine simulation that only uses up a fixed number of tile types per computation step. This stands in contrast to previous fuel-guzzling work in which each computation step is simulated by the assembly of a complete, slightly modified, copy of the entire tape.

1.1 Related Work Doty et. al. [20] first considered the effect of negative interactions in self-assembly and showed how to design space-efficient computational self-assembly systems by utilizing negative glues to slice off a previous assembly state after computing the next state. This state transition construction can be applied to the simulation of a Turing machine. However, their construction does not achieve fuel efficiency in that each computational step uses up a number of tiles on the order of the current size of the tape. Their result is also not applicable to our graph traversal system in that their technique, modified to the graph traversal problem, would lead to dead end assemblies that violate the required dynamics of the graph traversal. Finally, their paper utilizes a very powerful non-diagonal version of the tile self-assembly glue function. We restrict ourselves to the much more restricted diagonal class of glue functions in which glues may only interact with similarly labeled glues. Also, this restricted diagonal class of glue functions allows us to maintain a strict separation between positive and negative glues which is likely more feasible for experimental implementation. Additionally, Doty et. al. provide an elegant amortized analysis proof showing a limit to the extent to which tile reuse can be achieved with negative force interactions within the TAM. We implicitly use their result in that it allows us to reasonably simplify our definition of fuel usage to not deal with tile reuse.

Some recent work has been done that shows provable increases in power of active self-assembly systems over passive TAM systems. Recent work by Woods et. al. [34] and Nadine Dabby and Ho-Lin Chen [15] has shown that local rule based active self-assembly systems are able to assemble large shapes in poly-logarithmic time when the monomers of the system are able to perform operations such as pushing a row of placed monomers a unit of distance in some direction. Another recent work by Padilla et. al. [28] considers an active tile self-assembly model motivated by a DNA strand replacement mechanism in which tiles can pass simple, fire-once signals from one tile edge to another. They show that this simple mechanism allows for the implementation of fuel-efficient universal computation, as well as additional efficiencies that are provably impossible within the passive TAM.

Additional investigations have looked into self-assembly models that allow self-assembly detachment, a key requirement for the implementation of space-efficient computations [1,6,25,32]. Other recent work has considered negative glues within the TAM to achieve universal computation with a very limited set of glues at temperature-1 [29].

Paper layout. In Section 2 we define the tile assembly model and present definitions that define our graph traversal and Turing machine simulation problems. In Section 3 we present our construction for fuel efficient graph walking. In Section 4 we present a simple example graph walking system that implements a base-4 oscillator. In Section 5 we extend our graph walking construction to simulate Turing machines. Finally, in Section 6 we conclude with a discussion of future research directions.

2 Definitions and Model
In this section we first define the two-handed tile self-assembly model with both negative and positive strength glue types and diagonal glue functions. We also formulate the problem of designing a tile assembly system that walks a given input graph, as
2.1 Tile Self-Assembly Model

Tiles. Consider some alphabet of glue types $\Pi$. A tile is a finite edge polygon with some finite subset of border points each assigned some glue type from $\Pi$. Further, each glue type $g \in \Pi$ has some rational number strength $\text{str}(g)$. Finally, each tile may be assigned a finite length string $\text{label}$. In this paper we consider a special class of tiles that are unit squares of the same orientation with at most one glue type per face, with each glue being placed exactly in the center of the tile’s face.

Assemblies. An assembly is a set of tiles whose interiors do not overlap. For a given assembly $\Upsilon$, define the bond graph $G_{\Upsilon}$ to be the weighted graph in which each element of $\Upsilon$ is a vertex, and each edge weight between tiles is the sum of the strengths of the overlapping, matching glue points of the two tiles. An assembly $\mathcal{C}$ is said to be stable if the bond graph $G_{\mathcal{C}}$ has min-cut at least 1, and unstable if the min-cut is less than 1. Note that only overlapping glues that are the same type contribute a non-zero weight, whereas overlapping, non-equal glues always contribute zero weight to the bond graph. The property that only equal glue types interact with each other is referred to as the diagonal glue function property and is perhaps more feasible for experimental implementation than more general glue functions. Additionally, with the square tiles considered in this paper, stable assemblies will necessarily consist of tiles stacked face to face, forming a subset of the 2D grid.

For an assembly $\mathcal{A}$, let $\mathcal{A}^*$ denote the set of all assemblies that are equal to $\mathcal{A}$ up to translation. For a set of assemblies $\mathcal{T}$, let $\mathcal{T}^*$ denote the set of all assembly sets $\mathcal{A}^*$ such that $\mathcal{A} \in \mathcal{T}$. Put another way, $\mathcal{T}^*$ is a set that represents a set of assemblies $\mathcal{T}$ if we do not care about translation.

Breakable Assemblies. For an (unstable) assembly $\mathcal{C}$ whose bond graph $G_{\mathcal{C}}$ has a cut into assemblies $\mathcal{A}$ and $\mathcal{B}$ with weight less than 1, we say that $\mathcal{C}$ is breakable into $\mathcal{A}$ and $\mathcal{B}$.

Combinable Assemblies. Informally, two assemblies $\mathcal{A}$ and $\mathcal{B}$ are said to be combinable into an assembly $\mathcal{C}$ if the assemblies can be translated together in such a way that they do not overlap and the sum of the matched glue strengths between the two assemblies is at least 1. Formally, consider two assemblies $\mathcal{A}$ and $\mathcal{B}$. If $\mathcal{B}$ can be translated into $\mathcal{B}'$ such that $\mathcal{C} = \mathcal{A} \cup \mathcal{B}'$ is a valid (not overlapping) assembly such that the cut of $G_{\mathcal{A} \cup \mathcal{B}'}$ into $\mathcal{A}$ and $\mathcal{B}'$ has strength at least 1, then we say that $\mathcal{A}$ and $\mathcal{B}$ are combinable into $\mathcal{C}$.

Note that $\mathcal{A}$ and $\mathcal{B}$ may be combinable into an assembly that is not stable. This is a key property that is leveraged throughout our constructions. See Figure 1 for an example.

Producible Assemblies. A set of initial assemblies $\mathcal{T}$ has an associated set of producible assemblies, $\mathcal{PROD}_T$, which define what assemblies can grow from the initial set $\mathcal{T}$ by any sequence of combination and break events. Formally, $\mathcal{T} \subseteq \mathcal{PROD}_T$ as a base case set of producible assemblies. Further, given
any $A, B \in \text{PROD}_T$ with $A$ and $B$ combinable into $C$, then $C \in \text{PROD}_T$, and for any $C \in \text{PROD}_T$ where $C$ is breakable into $A$ and $B$, then $A, B \in \text{PROD}_T$. Put another way, the set of producible assemblies contains any assembly that can be obtained by a valid sequence of combinations and breaks, starting from the initial set of assemblies $T$. We further define the set of terminal assemblies, $\text{TERM}_T$, to be the subset of $\text{PROD}_T$ such that no element of $\text{TERM}_T$ can undergo a break or combination transition with another element of $\text{PROD}_T$.

Typically, we require that an initial assembly set consist only of stable assemblies. When $T$ consists only of assemblies that are singleton tiles, $T$ is called a tile set.

2.2 Additional Definitions

Valid Assembly Sequence. For an initial assembly set $T$, a valid assembly sequence for $T$ is any sequence of assemblies $S = \langle a_1, \ldots, a_k \rangle$ such that each $a_i \in \text{PROD}_T$, and for each $i$ from $1$ to $k-1$, $a_i$ is either combinable with some $b \in \text{PROD}_T$ to form $a_{i+1}$, or $a_i$ is breakable into $a_{i+1}$ and $b$ for some assembly $b$. In addition, a valid assembly sequence $S$ is said to be $\ell$-focused for label $\ell$ if each assembly in $S$ contains at least one tile with label $\ell$. A valid assembly sequence is said to be nascent if $a_1 \in T$.

For a given partial function $f : \text{PROD}_T \rightarrow V$ for a set $V$, define function $f'((a_1, \ldots, a_k))$ to be a function from assembly sequences over $\text{PROD}_T$ to sequences over $V$ defined by replacing each $a_i$ with $f(a_i)$ if $f(a_i)$ is defined, and deleting $a_i$ if $f(a_i)$ is not defined.

Graph Walking Assemblies. Consider a directed graph $G = (V, E)$, a start vertex $s \in V$, and an initial assembly set $T$ in which some tiles of some assemblies are labeled with $\ell$. $T$ is said to walk graph $G$ if there exists a partial function $f : \text{PROD}_T \rightarrow V$ such that:

1. (assembly sequences are walks) For any $\ell$-focused nascent assembly sequence $S$ of $T$, $f'(S)$ is a valid walk of $G$ starting at vertex $s$.

2. (walks are assembly sequences) For any walk $W$ of $G$ starting at vertex $s$, there exists a nascent, $\ell$-focused assembly sequence $S$ such that $f'(S) = W$.

3. (no undesired dead-ends) Consider an edge $(u, v) \in E$. Then for any $\ell$-focused assembly sequence $S$ such that $f'(S) = (u)$, there exists an $\ell$-focused assembly sequence $R$ such that $S$ is a prefix of $R$ and $f'(R) = (u, v)$.

In this definition, the $\ell$ label is meant to denote tiles and assemblies that are meant to be part of the output, and not garbage. That is, to walk a graph efficiently there will necessarily be dead-end garbage assemblies. Our definition thus requires that all such garbage assemblies be free of the $\ell$ label and thus can be distinguished from the assemblies that represent the graph walking assemblies. In a sense, our definition states that if you stare at one $\ell$-labeled tile within an assembly, over time the assembly containing that tile will be guaranteed to walk the graph correctly, whereas if you stare at a portion of the assembly that is not labeled with the $\ell$, it is possible that portion may detach and become garbage at some point.

Fuel Efficiency. Consider two assemblies $a$ and $b$. Define $\text{fuel}(a, b)$ to be $|b| - |a|$ if $|b| \geq |a|$, and zero otherwise. For an assembly sequence $A = (a_1, \ldots, a_k)$, define $\text{fuel}(A)$ to be $\sum \text{fuel}(a_i, a_{i+1})$.

For a tile set $T$ that walks a directed graph $G$, $T$ is said to be fuel efficient if for all $\ell$-focused, nascent assembly sequences $S$, $\text{fuel}(S)/|f'(S)| = O(1)$. That is, for all possible walks, the average fuel per vertex transition is bounded by a constant.

3 Graph Walking

In this section we construct a tile assembly set for the fuel-efficient walk of a given input directed graph:

**Theorem 3.1.** For any given finite directed graph $G = (V, E)$, there exists a fuel efficient initial assembly set $T$ that walks $G$. Further, $T$ contains at most $O(|V| + |E|)$ distinct assemblies each of $O(1)$ size.

This theorem can be slightly strengthened in that it is possible to modify the construction so that $T$ consists of only singleton tiles. However, for clarity we do not present this modification.

The construction for this result serves two purposes. First, we believe the result itself is interesting. One example corollary is the ability to implement oscillator self-assembly systems with an example given in Section 4. The second purpose is that this construction presents some of the key techniques in a simplified form that we will extend to construct our main result: fuel efficient Turing machines.

The remainder of this section describes the construction for the proof of this theorem.

3.1 Template Assembly Set

The following tile assembly template allows one to create an assembly set that can non-deterministically walk across a di-
erected graph. This template creates an assembly set that has $O(|V| + |E|)$ assembly types. Also, the template uses a modest 9 fuel tiles per transition step. An example of constructing an assembly set using this template follows in Section 4 where we build a base four oscillator.

Figure 2: These four tiles are needed in order to represent a vertex in a graph. All glues labeled with some form of an $a$ require unique glues to the specific vertex these tiles represent and may not be shared amongst other vertices. The tile in Figure 2a allows an edge gadget to detach its corresponding vertex. The tile in Figure 2b allows an edge gadget to know when its vertex has finished attaching. Finally, the pair of tiles in Figure 2c represent a single vertex in a graph.

**Vertex Representation.** For each vertex in the graph you must have the set of two tiles shown in Figure 2a, a vertex detachment tile shown in Figure 2a, and finally a vertex attachment tile which is shown in Figure 2c. The five glues labeled $a'$, $a^*$, $a_+$ and $a_-$ on the duple of Figure 2b are unique to some vertex $a$ while the two southern glues are the same for all pairs of tiles.

Figure 3: The edge gadget and its detachment tile. The assembly in Figure 3a is the edge gadget and represents a directed edge from vertex $a$ to vertex $b$. The tile in Figure 3b is the tile that will allow the edge gadget to detach itself once done.

**Edge Gadget.** Figure 3a is the edge gadget which exchanges a vertex $a$ with the vertex $b$ on the cell assembly. When the edge gadget detaches itself from the cell assembly all tiles that detach with the gadget are no longer usable. In Figure 3a the glues labeled with some $a$ represent glues unique to some vertex $a$ while the glues labeled with some $b$ represent glues specific to some vertex $b$. The glue labeled with a $z$ is unique to the edge detachment tile shown in Figure 3a. This tile allows the edge gadget including its junk to detach from the cell assembly.

Cell Assembly. This cell assembly is assumed to be pre-built and contains the initial starting vertex $v$ already placed on top of the cell assembly. As you can see in Figure 4 the glues $c$ and $d$ are not discriminatory and allow any two tiles representing a vertex to attach to the cell assembly with the aid of an edge gadget. The $6^{th}$ glue along with the $a_*$ glue allow the edge gadget to attach. After the attachment the $5^{th}$ glue allows for the widget to get a better “grip” on the assembly cell so that it may detach the current vertex. The final $5^{th}$ glue on the assembly allows the edge gadget to pry itself off.

**Correctness** As can be seen from Figure 5 the possible transitions made by changing from one state to another are straightforward and verifying that each single tile addition must happen in the order shown is left as a short exercise to the reader. We will concentrate on the two negative glue detachment events which are more complex and need to be verified so as to ensure that no other cuts in the produced assemblies are unstable other than those depicted. We shall assume that each vertical column of the image represents a stage in the transition and so there are a total of 9 stages from start to finish.

**Correct Gadget Removal.** After the gadget is done a tile may attach to the negative $5^{th}$ glue on the cell assembly which in turn pops off the edge gadget along with any junk. Two scenarios are brought about by the infinite binding within the cell assembly.
and edge gadget independently. The first scenario occurs when the cell assembly and edge gadget end up on opposite sides of the detachment. The other happens when both the cell assembly and edge gadget end up on the same side of the detachment. We will analyze both of these scenarios and show that from stage 8 in Figure 5 the only valid detachment is what results in stage 9.

Suppose both the cell and the gadget detach to opposite sides of the cut. We must include a negative glue to get an unstable cut and so the edge detachment tile must be on the same side as the gadget. This gives us a cut with weight $\frac{1}{8}$ not taking into account any of the other four labeled tiles. For the cut to be unstable we can only increase the weight by $\frac{6}{8}$ so as to keep it below 1. To stay unstable both the vertex detachment tile and the vertex attached tile must be on the side of the edge gadget because of their $\frac{7}{8}$ attachment. Also, the left vertex tile must stay on the side of the cell assembly for the same reason. With this new information the total weight of the cut is now $\frac{6}{8}$ and we only have the right vertex tile to place. The right vertex tile must now stay on the side of the cell because to keep the cut unstable we may only increase the weight of the cut by $\frac{1}{8}$ more. Therefore, it is easy to see that the only possible unstable cut if both the cell and gadget are on opposite sides is the one displayed in Figure 5.

Now, suppose that both the cell assembly and the edge gadget detach to the same side of the cut. In order to include the negative strength of the edge detachment tile it must be on the opposite side of both the edge and the gadget. This immediately shows us that our cut is $\frac{9}{8}$ which is stable. With no other negative glue interactions, there is no cut with both the cell assembly and the edge gadget on the same side that would lead to a detachment.

Therefore, since we have exhaustively explored both scenarios using the fact that the cell assembly and edge gadget are infinitely bound we can say that the only unstable cut at stage 8 is the one that results in stage 9.

**Correct Vertex Replacement.** Assume we are in stage 3 of Figure 5. We will show that the transition to stage 4 is the only valid detachment event that may happen. If the cell and edge gadget were to detach to opposite sides of the cut this would give us a cut with weight $\frac{5}{8}$. Then, no matter which side of the cut you place the vertex removal tile all cuts will be above 1. If both the edge gadget and the cell are on the same side of the cut, the left vertex tile must be on the opposite side of the cut since the vertex removal tile will not fall out. This means that either the left vertex tile or both vertex tiles must be on the opposite side of the cut. If both vertex tiles are on the opposite side of the cut we have a cut with weight $\frac{11}{8}$ which is stable. On the other hand, if it is only the left vertex tile on the opposite side of the cut then the weight of the cut is $\frac{7}{8}$ which enables it to detach. Therefore, the only detachment event that can occur after stage 3 is the single left vertex tile detaching from the assembly leaving the assembly in stage 4. Once in stage 4 the right vertex tile destabilizes and may detach, leaving room for the next vertex to attach and stage 6 to come about.

**4 Quaternary Oscillator**

A quaternary oscillator can be drawn as a directed graph with vertices representing each digit in the base-4 numeral system and the directed edges marking the transition from one numeral to the next. Figure 6 shows a directed graph representing a quaternary oscillator. The Graph Walking Template Assembly Set from Figures 2, 3, and 4 is non-deterministic when any vertex in the graph has more than one outgoing edge because any one of the edges
may attach and continue to transition. The quaternary oscillator in Figure 8 on the other hand, has only a single outgoing edge for any vertex making the template deterministic. Figure 7 represents the complete assembly set necessary to represent the graph in Figure 8. This is a straightforward example of how to create a concrete assembly set using the assembly template presented in Section 3.1.

5 Fuel Efficient Universal Computation

In this section we describe how to construct an initial assembly set $T$ that will simulate a given Turing machine $M$. To formally define simulation of a Turing machine, we utilize our graph walking formulation. For a given Turing machine $M$, we consider all possible tape configurations, states, and head locations the machine may enter over time and form a directed graph over these possible configurations depicting which configurations the machine may transition between within a single computation step. With this definition we say that an assembly set $T$ simulates $M$ if it walks this configuration graph.

Formally, we define the simulation as follows:

**Turing Machine Simulation.** Consider some Turing Machine $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$, an initial tape $t$, and head position $h$. Let a possible status of $M$ be a 3-tuple $S = (\text{state}, \text{tape}, \text{head})$ where \text{state} represents some current state from $Q$, \text{tape} is some string of symbols over $\Gamma$, and \text{head} specifies a location on the tape where the Turing machine head currently sits. Additionally define the \text{start} status for a Turing machine $M$ and an input tape $r$ as the status corresponding to $\text{tape} = r$, $\text{head} = h$, and $\text{state} = q_0$.

Given a Turing machine $M$, define a directed graph $G_M$ where the set of all statuses for $M$ are vertices, and there exists an edge from any status $a$ to status $b$ if and only if $M$ can transition from status $a$ to $b$ in one step. Finally, we say an initial assembly set $T$ simulates Turing machine $M$ if $T$ walks $G_M$. Further, we say $T$ is a fuel-efficient simulation of $M$ if $T$ constitutes a fuel-efficient walk of $G_M$.

Our main result is as follows:

**Theorem 5.1.** For any Turing machine $M$, there exists an initial assembly set $T$ such that $T$ is a fuel-efficient simulation of $M$. Further, $|T| = O(|\Gamma| \cdot |Q|)$, and each element of $T$ is of $O(1)$ size.

As with our graph walking theorem, it is possible to strengthen this theorem such that $T$ contains only singleton tile assemblies. For clarity of presentation and space, we omit the extended construction that achieves this.

The remainder of this section presents the construction that proves Theorem 5.1.

Figure 8: These tiles represent a symbol and state pair along with their needed utility tiles.

5.1 Template Tile Set

**Symbols And States.** For our construction we add an additional state, the empty state $\emptyset$, to $M$ which will be used to represent a symbol with no state i.e. a location where the head of the Turing Machine currently is not. For every symbol in $\Gamma$ and every state in $Q$ including $\emptyset$ we create the set of tiles shown in Figure 8. This means that we should have exactly $|\Gamma| \cdot (|Q| + 1)$ sets of tiles. The two center tiles labeled with an $A$ in Figure 8 will represent some symbol pair as having the $\emptyset$ state while a stateful state symbol pair has some $q \in Q$. The two leftmost tiles are called state symbol detachment tiles and detach the particular state symbol tile pair they were created with. This will be useful later when trying to transition the state of the Turing Machine. Finally, the last tile on the right is the state symbol attached tile and will allow the assemblies to know that the state symbol tile pair associated with it have both been attached.

Figure 9: This assembly represents a single position on the tape. It is a tape cell which can and should contain some symbol state pair atop it attached to glues $c$ and $d$. The two accompanying tiles are what will allow the tape to extend to the right or left.

**Tape Cell.** A tape cell in our construction is a single piece of tape which contains some state symbol tile pair taken from Figure 8. If the state contained on the tape cell is $\emptyset$ then the tape cell contains only data, i.e. a symbol from $\Gamma$. On the other hand, if the...
state contained on the tape is not $\emptyset$ then the location of the Turing Machine’s head is at that location and it contains some symbol from $\Gamma$. The whole of the tape is then at least one tape cell possibly combined with many more tape cells attached together with the head of the tape in only a single location i.e. with a single cell containing a state unequal to $\emptyset$ at any given time. Should the head of the Turing Machine be placed in multiple locations it will result in undefined behavior. When we later extend the tape we insure that the extended piece of tape gets initialized automatically to the blank state; i.e. the symbol $b$ in combination with the state $\emptyset$. It is important to note that in order to properly transition across the directed graph $G_M$, we must label a tile on the left-most starting tape cell with an $\ell$.

**Transition Function.** A Turing Instruction is the 5-tuple $I = \langle c \in Q, s \in \Gamma, p \in \Gamma, \{L,R\}, n \in Q \rangle$ where $c$ is the current state, $s$ is the scanned symbol, $p$ is the symbol to print, $\{L,R\}$ is the left or right movement of the head, and $n$ is the new state. For each Turing Instruction in $M$ create $|\Gamma|$ transition gadgets. A single transition gadget is shown in Figure \[10\]

For two state symbol pairs we follow the same process as exchanging a single vertex in the Graph Walking Template with a few exceptions. The most obvious exception is that we are now exchanging two symbol state pairs instead of a single vertex pair. In order to facilitate this change we extend the transition gadget to accompany the two exchanges. The next difference is that after exchanging the first pair of tiles the next symbol state pair must be exchanged instead of detaching the whole transition gadget. This is the reason for needing two state symbol detachment tiles instead of just the single one as in the Graph Walking Template. Now, of the two detachment tiles the first is used when detaching the left state symbol pair underneath the transition gadget and the second is used when detaching the right state symbol pair underneath the transition gadget. This along with the changing of a couple glue strengths gives us a transition gadget shown in Figure \[10\] that is able to exchange two symbol state pairs at the same time.

Now, in order to create the transition gadgets which represents a Turing instruction, we must properly set the label of several unlabeled glues. We will temporarily number the unlabeled glues in Figure \[10\] from 1 starting at the left to 8 at the farthest right. It does not matter whether you are moving the head left or right, the stateful pair always becomes stateless and the stateless pair always becomes stateful. Therefore, the only difference between moving the head right or left is that the stateful pair is at the left when moving right or at the right when moving left. We will continue to assume that we are creating the transition gadgets for some $I$ which moves the head to the right. Since we are moving right, glues 3 and 1 will use the proper glue labels for the state

---

Figure 7: The complete assembly set for creating a quaternary oscillator.
symbol pair \((c, s)\). Glues 2 and 4 use the proper glue labels from the state symbol pairs \((\varnothing, p)\). Now, the next part explains the reason why we need \(|\Gamma|\) gadgets. All gadgets now have glues 1 through 4 filled in with labels from our assembly set. Next, fill in glues 5 and 7 for all gadgets with the proper glue labels from every state symbol pair that contains the \(\varnothing\) state. Finally glues 6 and 8 are filled using the glue labels for every state symbol pair that contains state \(a\) such that the symbol represented in 5 and 7 matches those of 6 and 8 within the same transition gadget. Figure 11 contains an overview of the necessary steps that allow the transition gadget to work.

With the gadget just about to detach, we can use the same arguments presented in Section 3.2 to prove that the transition gadget detaches correctly. The tape cell and transition gadget themselves are strongly connected and should not fall apart. Therefore, we start by saying that either both the tape cell and transition gadget end up on the same side of the cut or they end up on different sides of the cut. Using this information and the fact that we want a cut below 1 we can whittle down the possible sides of all the tiles in either situation. As one will be able to see the only possible cut below 1 happens when the tape cell and transition gadget are on opposite sides of the cut and all the fuel tiles are on the same side as the transition gadget.

**Tape Extension.** In order to simulate a Turing Machine and be space efficient we must ensure that our tape does not grow infinitely but extends only when needed. Our construction ensures that our tape grows only when necessary. We do this by using a recurring theme, attaching what we need and then using some negative glues to destabilize the junk assembly.

We will need four tiles, shown in Figure 12 which will serve as a way for the tape extension gadgets to detach themselves once they are done with their work. For simplicity we create a tape extension gadget for every stateful state symbol pair. Since the tape extension gadget must connect to the stateful
state symbol pair at the right end of the tape, we create a tape extension gadget for every stateful state symbol pair by changing only the leftmost $\frac{1}{8}$ glue to match the pair. The tape extension gadget can only attach if the position of the head is at the edge of the tape i.e. the stateful state symbol pair is at the edge. Once the head makes its way to the edge of the tape, the tape extension gadget is able to attach and extend the tape. Then the tape extension gadget places the two tiles shown in Figure 9 which will allow another tape cell assembly to attach. Once the new tape cell assembly has attached the stateless blank or “default” state symbol pair can then be attached to the tape cell. After the attachment of the state symbol pair, tiles cooperatively bind around the tape extension arch until it attaches a tile underneath the tape cell arch. Finally, three tiles attach which cause the detachment of the extension gadget. A detailed overview of this process can be seen in Figure 13. The proof that the tape extension gadget detaches properly is easily verifiable at this point and is left as an exercise to the reader.

5.2 Tape Reduction A Turing machine in theory has an infinite length tape to work with. To simulate this property our construction automatically extends the finite length tape on demand when needed. However, it is reasonable that a Turing machine may grow a large tape, and then subsequently “blank” out much of the tape leaving only a small length portion of tape between the leftmost and rightmost non-blank symbol. In some sense, it is a waste of space to explicitly encode the large number of blank bits that are beyond this region.

In order to maintain space efficiency, which requires that the size of the assembly representing each computational step is asymptotically bounded above by the region of the tape between the leftmost and rightmost non-blank symbol, we have created the tape reduction gadget. This gadget simply clips off the rightmost bit of the tape under certain circumstances such as if the rightmost bit has been blanked. This gadget allows for our Turing machine to achieve space efficiency, which is the requirement that the size of the assembly representing each computational step is asymptotically bounded above by the size of the region of the tape between the leftmost and rightmost non-blank symbol, or the region between the head position and either the right or leftmost non-blank symbol.

One motivation for achieving space-efficiency is the potential implementation of components from active self-assembly models within our passive tile assembly model. In particular, our space efficient construction may be utilized as the internal workings of a larger active component. While the computation required of an active component may require a large tape size, the final output may be taken from a small set of options such as “turn on glue a”, or something similar. With space efficiency, there is hope that our passive implementation will be able to perform the required functionality without growing so large that the geometry of the mechanism interferes with the higher-level active system mechanics.

We do not present the full details of our reduction gadget(s) in this extended abstract.
6 Future Work

The results of this paper raise a number of interesting research questions. The first focuses on the primary drawback of our technique: the use of negative strength glue interactions. Is fuel-efficient computation in passive self-assembly possible with the use of only positive strength glue interactions? Such a system seems difficult to imagine as positive strength systems may only grow and cannot break apart under current standard models. However, there is currently no proof that such a system cannot exist, even in very restricted settings such as the standard positive strength tile assembly model.

Another direction of research is the application of fuel-efficient constructions to complexity questions in tile self-assembly. For example, the existence of a space-efficient tile system with negative strength glues that simulates universal computation implies that some interesting verification problems in tile self-assembly become uncomputable. Does the existence of fuel-efficient systems have similar implications to computability and complexity?

An additional metric that tile assembly research often attempts to minimize is the temperature of the tile system. The temperature is an additional positive integer parameter included in a tile system that dictates how much glue strength is required for two assemblies to stick together. In our paper, if we restrict ourselves to integer glue strengths and include a temperature parameter, our constructions correspond to temperature 8 systems. An interesting question is to determine the smallest integer temperature value at which fuel-efficient computation can take place.

Another interesting general direction is to explore alternate fuel sources for tile assembly. In this paper we have focussed on the natural metric of permanent tile expenditure. An alternate approach is to consider some modification to the model that permits harnessing an external fuel source. One example is a self-assembly system in which the temperature of the system perpetually oscillates up and down. Is it possible to harness such oscillation to obtain a fuel neutral self-assembly system? Do there exist other promising and practically motivated fuel models, such as a model that incorporates a single type of active fuel rod component to fuel the assembly?

An additional promising direction is to explore the extent to which active behavior can be simulated within passive self-assembly models. One recent model of active self-assembly is the signal tile assembly model in which tiles may pass single, fire-once signals across tile faces. Such systems are known to

Figure 13: This figure shows the steps that occur when extending the tape to the right.
be capable of fuel-efficient computation. Can passive self-assembly tile systems with negative glue interactions simulate general signal tile systems?

Finally, while our construction fuel-efficiently simulates a single tape Turing machine, the question of fuel-efficient simulation of more efficient machines, such as a two-tape Turing machine, is still open. In particular, two-tape Turing machines are perhaps the “machine of choice” among Turing machine programmers. In addition, two-tape machines offer a linear factor efficiency improvement over single tape machines. Therefore, it is extremely interesting to know if it is possible to fuel-efficiently simulate a two-tape machine in passive self-assembly, or even in more active models of self-assembly.

Acknowledgements

We would like to thank Damien Woods, Alexandra Keenan, Xingsi Zhong, Jennifer Padilla, Matt Patitz, Scott Summers, and Dave Doty for helpful discussions and comments during the writing of this paper. We would also like to thank the anonymous reviewers for their constructive feedback which has been incorporated into the paper.

References


[16] Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin,


