A Computational Model for Self-assembling Flexible Tiles

Nataša Jonoska and Gregory L. McColm

Department of Mathematics, University of South Florida Tampa, FL 33620
{jonoska, mccolm}@math.usf.edu

Abstract. We present a theoretical model for self-assembling tiles with flexible branches motivated by DNA branched junction molecules. We encode an instance of a "problem" as a pot of such tiles, and a "solution" as an assembled complete complex without any free sticky ends (called ports), whose number of tiles is within predefined bounds. We develop an algebraic representation of this self-assembly process and use it to prove that this model of self-assembly precisely captures NP-computability when the number of tiles in the minimal complete complexes is bounded by a polynomial.

1 Introduction

Many researchers use weak chemical bonds to design and "grow" self-assembled nanostructures. Under thermodynamic equilibrium, molecules assemble using several types of non-covalent intermolecular interactions (e.g., hydrogen or ionic bonds), evolving into relatively stable structures [23,24]. This paper is motivated mainly by DNA self-assembly, employing Watson-Crick complementarity and hydrogen bonding, as a case study for molecular self-organization due to weak bonding. The interest comes from several major experimental developments that use DNA for constructing nanostructures, for information processing, and as a material for nanodevices.

Nanostructures. An essential part of the chemical engineering of self-assembling structures is the design of the molecular building blocks that will bind into larger, and more complex structures. Although naturally-occurring DNA is a linear molecule (considering its helix axis as a segment of a curve), DNA molecules can be constructed as stable branch points [31,35] and ultimately as more complex structures with lateral fusion of DNA helices such as double DX and triple crossover TX molecules [11,22,32]. DNA and other molecules have been used for self-assembly of two dimensional arrays [36,37], three-dimensional graph-like structures [15,30] and regular polyhedra [6,40], including the octahedron (by DX and PX molecules) [33].

Algorithmic self-assembly. Beginning with the initial successful experiment by Adleman [1] and more recently one from the same group solving an instance

C.S. Calude et al. (Eds.): UC 2005, LNCS 3699, pp. 142–156, 2005.

of SAT with 20 variables [5], computations by biomolecular protocols include, among others, binary addition (simulation of XOR) using triple cross-over molecules (tiles) [25], a 9-bit instance of the "knight problem" using RNA [10], and a small instance of the maximal clique problem using plasmids [13]. Recently, Winfree [29] used algorithmic self-assembly to obtain cellular automata like two-dimensional arrays of the Sierpinski triangle.

Nanodevices. Based on a B-Z transition of the DNA helix, a nano-mechanical device was introduced in [26]. Soon after, "DNA fuel" strands based on Watson-Crick hydrogen bonding were used to produce devices whose activity were controlled by DNA strands [38], [39]. The device introduced in [38] has two distinct positions, each obtained by adding a pair of DNA strands that hybridize with the device so that the molecule is either in the first or in the second position.

Theoretical observations. Despite notable advances in experimental molecular self-assembly, the theoretical understanding of this process is lacking. Algorithmically, it has been observed that DNA tiles can simulate Wang tiles and as such, are capable of simulating the transitions of a Universal Turing machine [36]. However, there is a real need for understanding the limitations, boundaries, and complexity of the process of self-assembly. Only a few theoretical results have been obtained, primarily for DNA assemblies using rigid tiles. The complexity, measured as the number of tiles needed for a unique assembly of $n \times n$ squares, is considered in [28], where it was observed that only $O(\log n)$ molecules are needed for this task. Comparison of such "shape" complexity with Kolmogorov complexity is investigated in [34]. The same model was used to theoretically observe a possible two-dimensional tile self-assembly of a cube [21]. In [2], computing the minimal number of tiles needed for unique self-assembly in a given shape proved to be NP-hard, and $O(\log n)$-approximation of the concentration of the tiles needed for fast assembly in the desired structure is computed. The question of whether a given set of tiles arrange in an infinite ribbon-like shape was proved to be undecidable [3]. In [16] and [17], the tiles are "flexible" in that they have extended bendable branches, the branches being labeled so that the assembly process is guided by complementary Watson-Crick labels; the algorithmic view is that the input is encoded as a collection of tiles to use, and an output (if any) is a sufficiently small complete complex, i.e., a complex of tiles with no free ports. This flexible tile model has been investigated in [18], which presented a heuristic model to predict the distribution of products of the self-assembly. In [27], a step-wise assembly of junction molecules is investigated and the computational complexity of the problem, whether a complete complex is produced by a given pot of a certain form is considered.

In this paper, we take a somewhat reverse course from [27], and ask what are the problems solvable by flexible tile assembly; we find that these are precisely the NP-complete problems. Specifically, we find that all flexible tile DNA computations within a polynomial bound are reducible to the NP-computable integer programming problem of [4] and [20], and conversely, that any NPTIME computation can be simulated by a flexible tile computation. This note is an extended abstract of results appearing in [19].

2 The Theoretical Model

The main building blocks used in our assembly model are motivated by branched junction DNA molecules with junctions ending with single stranded portions (sticky ends) that can bind to their Watson-Crick complementary sequence. It is also assumed that the structure of the junctions (and possibly the branches) is such that the branches are rather flexible and various connections are possible. A schematic view of this process is presented in Figure 1. It has been shown theoretically that several NP-complete problems such as 3SAT and 3-vertex-colorability can be solved by self-assembly of DNA graphs [15,16,17]. The coding of the problems is such that a solution is obtained if and only if the graph can be assembled. Also, experimental confirmation of DNA graph self-assembly has been obtained by flexible tiles in [15,30]. In what follows we develop the theoretical model and prove these initial observations in a general setting.

Let $H \subset \Sigma^*$ be a finite set of words over alphabet Σ that we call *port (bonding) types* and let θ: $H \to H$ be an involution. We call $\theta(\mathbf{h}) \in H$ the *complementary* string to $\mathbf{h}$ such that ports of types $\mathbf{h}$ and $\theta(\mathbf{h})$ bond. For each $\mathbf{h} \in H$ we assume that $\theta(\mathbf{h}) \neq \mathbf{h} = \theta(\theta(\mathbf{h}))$.

Fix H and θ for the rest of this paper. To ease notation we write $\hat{\mathbf{h}}$ for $\theta(\mathbf{h})$.

Definition 1. *A* tile type *over* (H, θ) *is a function* $\mathbf{t}$: $H \to \mathbb{N}$.

A tile of type $\mathbf{t}$ will have $\mathbf{t}(\mathbf{h})$ ports of type $\mathbf{h}$. If $|H| = k$, then a tile type can be written as a k-dimensional vector with non-negative integer entries; alternatively, a tile type can be regarded as a multiset of port bonding types. We call $d = d(\mathbf{t}) = \sum_{\mathbf{h}} \mathbf{t}(\mathbf{h})$ the *degree* of tile type $\mathbf{t}$. In order to get the tiles themselves, we first get a collection of "prototiles." A *prototile corresponding to tile type* $\mathbf{t}$ is a star-like graph with one central vertex of degree $d(\mathbf{t})$ indicating the center of the prototile and $d(\mathbf{t})$ vertices of degree one labeled with ports indicating the branches seeking to bond with complementary ports. A tile is thus a copy of a prototile, so we say that a tile t that is a copy of a prototile of type $\mathbf{t}$ is thus itself of type $\mathbf{t}$, and we write: for each $\mathbf{h} \in H$, $t(\mathbf{h}) = \mathbf{t}(\mathbf{h})$ means

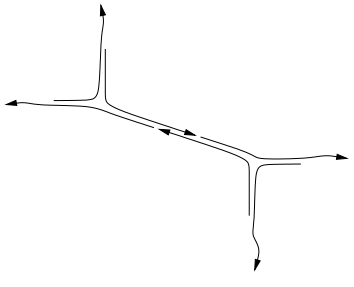

Fig. 1. Watson-Crick bonding of two DNA junction molecules. The double helix structure is not depicted for simplicity. The arrowhead indicates the $3'$ end also ending with a single stranded sticky end sequence.

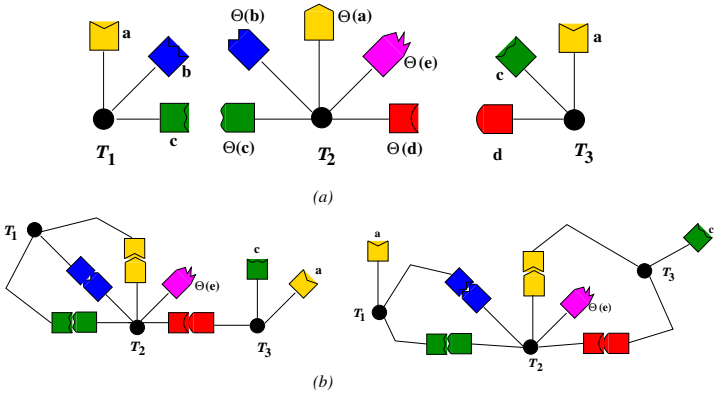

Fig. 2. Complexes. (a) Three tiles, with the central vertex indicated as a black circle, and the one-degree vertices with ports schematically presented with different colors and shapes. The complementary ports have compatible shapes, and same colors. (b) Two different (incomplete) complexes obtained by gluing the three tiles. They are of the same complex type despite their different structures, as they have the same multiset of free port types.

that t has exactly $\mathbf{t}(\mathbf{h})$ ports of type $\mathbf{h}$. Figure 2 (a) shows examples of three tiles with degrees 3, 5 and 3 respectively. The central vertex is represented with a black circle and ports are indicated with different colors and shapes.

We put tiles together to construct complexes. But first, we need to classify the pots that the complexes are assembled in.

Definition 2. *A* pot type *over* (H, θ) *is a set* $\mathbf{P}$ *of prototiles corresponding to tile types over* (H, θ) *such that for any* $\mathbf{h} \in H$ *and* $\mathbf{t} \in \mathbf{P}$*, if* $\mathbf{t}(\mathbf{h}) > 0$ *then there exists* $\mathbf{t}' \in \mathbf{P}$ *such that* $\mathbf{t}'(\hat{\mathbf{h}}) > 0$.

Thus no pot admits tiles with unattachable ports. A *pot* P is a set of tiles from $\mathbf{P}$.

Definition 3. *A* complex *over a pot type* $\mathbf{P}$ *is a pair* $\mathfrak{C} = \langle T, J \rangle$ *where* T *is a set of tiles with tile types in* $\mathbf{P}$ *and* J *is a set of unordered pairs* $e = \{(t, \mathbf{h}), (t', \mathbf{h}')\}$ *satisfying the following two properties:*

- *For each* $e = \{(t, \mathbf{h}), (t', \mathbf{h}')\} \in J$, $t, t' \in T$, $t(\mathbf{h}), t'(\mathbf{h}') > 0$, $\mathbf{h}' = \hat{\mathbf{h}}$ (e indicates the connection between two complementary ports)*, and*
- *the cardinality* $|\{e \,|\, (t, \mathbf{h}) \in e\}| \leq t(\mathbf{h})$ (this prevents the tile from making more connections than it has ports).

The type *of a complex* $\mathfrak{C} = \langle T, J \rangle$ *is the function* $\mathrm{type}(\mathfrak{C})$: $H \to \mathbb{N}$ *defined by*

$$\mathrm{type}(\mathfrak{C})(\mathbf{h}) = \sum_{t \in T} t(\mathbf{h}) - |\{e \,|\, (t, \mathbf{h}) \in e\}|$$

Thus the type of $\mathfrak{C}$ indicates the types of its ports, free (i.e., those not appearing in J). Similarly, as with tile types, the complex type can be viewed as a k-dimensional vector with non-negative integer entries. Note that each tile can be considered as a complex where the set J is empty and the set T is a singleton. Then the type of tile t is equal to the type of the complex it represents. We would like to distinguish tiles from complexes merely to indicate the fact that complexes are assembled from tiles.

We assume that assembly occurs in an extremely dilute solution, so that when two complexes meet, *all* of their complementary free sticky ends join up so that there are no complementary free ports. (This is where the flexibility of the tiles is so critical.) Thus:

Definition 4. *A complex* $\mathfrak{C}$ *over* (H, θ) *is* stable *if, for each* $\mathbf{h} \in H$*, either* $\text{type}(\mathfrak{C})(\mathbf{h}) = 0$ *or* $\text{type}(\mathfrak{C})(\hat{\mathbf{h}}) = 0$.

In this paper, we assume that all complexes are stable unless otherwise indicated.

As an example, consider Figure 2 (b), in which the ports of three tiles are (maximally) connected to produce either of the two non-isomorphic complexes of the same type. The ports are shown with different colors and shapes.

More generally, we may join complexes to obtain bigger complexes: if $\mathfrak{C}_1 = \langle T_1, J_1 \rangle$ and $\mathfrak{C}_2 = \langle T_2, J_2 \rangle$ are two complexes, then we can *glue* them together by connecting up their complementary ports to get a complex $\mathfrak{C} = \langle T, J \rangle$. There may be several non-equivalent ways to do this. Let ΔJ be a set of unordered pairs $\{(t_1, \mathbf{h}_1), (t_2, \mathbf{h}_2)\}$ such that a free port of type $\mathbf{h}_1$ from tile $t_1 \in T_1$ connects to a free port of type $\mathbf{h}_2 = \hat{\mathbf{h}}_1$ from tile $t_2 \in T_2$. We have the following definition:

Definition 5. *We say that* $\mathfrak{C} = \langle T, J \rangle$ *is obtained by* gluing *complexes* $\mathfrak{C}_1 = \langle T_1, J_1 \rangle$ *and* $\mathfrak{C}_2 = \langle T_2, J_2 \rangle$ *if*

$$T = T_1 \cup T_2 \qquad \text{and} \qquad J = J_1 \cup J_2 \cup \Delta J,$$

with the restriction that for each $\mathbf{h} \in H$*, as many ports of type* $\mathbf{h}$ *as possible are joined, i.e., for each* $\mathbf{h}$, $\text{type}(\mathfrak{C})(\mathbf{h}) = |\text{type}(\mathfrak{C}_1)(\mathbf{h}) - \text{type}(\mathfrak{C}_2)(\hat{\mathbf{h}})|$.

A complex is called *complete* if it has no free ports, i.e., if for all bonding port types $\mathbf{h}$, $\text{type}(\mathfrak{C})(\mathbf{h}) = 0$. For a pot P we denote with $\mathcal{C}(P)$ the set of all complete complexes that can be obtained by tiles from P, and for a pot type $\mathbf{P}$, let $\mathcal{C}(\mathbf{P}) = \bigcup_{P \in \mathbf{P}} \mathcal{C}(P)$. Note that if P is finite, so is $\mathcal{C}(P)$. The case when $\mathcal{C}$ is finite is discussed in [19].

3 Example: The 3SAT Problem

The *satisfiability problem* (SAT) asks whether for a given boolean formula φ, there is an assignment of {TRUE, FALSE} (or $\{T, F\}$) to the variables in φ that would assign to φ the value TRUE. (This is the archetypic NP-complete

problem [7].) Writing "+" for "or" and concatenation for "and," a formula φ is in *conjunctive normal form* (CNF) if it can be expressed as $\varphi = c_1 c_2 \cdots c_r$ where each c_i is a clause of the form $(a_1 + \cdots + a_k)$, and each a_i is a *literal*, i.e., either a variable v or a negation of a variable $\bar{v}$. It is in *k-conjunctive normal form* (k-CNF) if each conjunctive clause contains at most k literals. The satisfiability problem for 3-CNF formulas is known as 3SAT [7]. Consider for example:

$$\varphi = (\bar{x} + y + \bar{z})(x + \bar{y} + z)(\bar{x} + \bar{y} + z). \tag{1}$$

This Formula 1 has value T for the assignments $(x, y, z) \in \{(F, F, F), (T, T, T), (T, F, F), (F, T, T), (F, F, T)\}$, and F for any other assignment.

For each 3-CNF formula φ we associate a pot type $\mathbf{P}$ with the following tiles. The alphabet from which the port types are taken is

$$H = \{T, F, \hat{T}, \hat{F}\} \cup \{x, \bar{x}, \hat{x}, \hat{\bar{x}} \,|\, x \text{ is a variable } \} \cup \{c, \hat{c} \,|\, c \text{ is a clause } \}.$$

The symbols with $\hat{}$ indicate the θ complements. The set of port types consists of three letter words which we write as ordered triples for easier reading.

$$\begin{aligned} H = \big\{&(\iota, c, a), \theta(\iota, c, a) \,|\, \iota \in \{T, F\},\ c \text{ a clause},\ a \text{ a literal } \big\} \\ &\cup \big\{(x, \bar{x}, \iota), \theta(x, \bar{x}, \iota) \,|\, x \text{ is a variable},\ \iota \in \{T, F\}\big\}. \end{aligned}$$

The tiles differ in the assignment of the free ports. For each clause c_i there are seven tiles with three free ports of types (ι, c, a), a being a literal of c and ι being a truth assignment. Each tile corresponds to a truth assignment to the variables making the clause it belongs to true: its ports will be of types (ι, c, a), for a having truth assignment ι. (The truth assignment making the clause false gets no tile.) For example, the tile type corresponding to the clause $c_1 = (\bar{x} + y + \bar{z})$ for Formula 1 with assignment $(x \to F, y \to T, z \to T)$ has ports of types $(T, c_1, \bar{x})$, (T, c_1, y), and $(F, c_1, \bar{z})$ indicating $(\bar{x} \to T, y \to T, \bar{z} \to F)$. This is depicted in Figure 3 (a).

For each variable x, if x appears in s clauses, we associate x with two tiles, each with $s + 1$ ports. Each such tile corresponds to one of the two possible truth values of the variable, T or F. Each of s ports is complementary to the

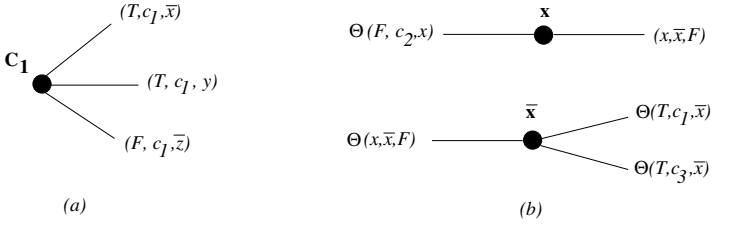

Fig. 3. (a) Tile for clause c_1 for Formula 1 with assignment (F, T, T). (b) Tiles corresponding to variable x and $\bar{x}$ (with value F for x) in 1.

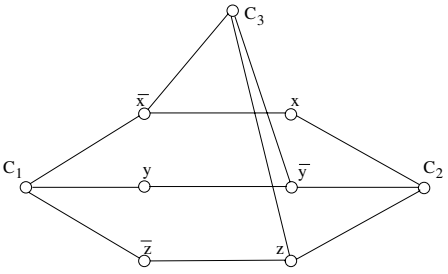

Fig. 4. A complete complex for Formula 1. Connections between complemenary ports are treated as edges in the graph.

corresponding port in one of the corresponding clause tiles (encoding θ(truth value, clause, variable)). The additional $(s+1)$st port is connected to the tile of the negation of the variable. Similarly, such tiles are encoded for the complements of the variables. The tiles corresponding to x and $\bar{x}$ in Formula 1 assigning value F to x are depicted in Figure 3 (b).

Any complete complex has to have at least one tile for each clause, and one for each variable (two if the variable and its negation both appear). A complete complex with exactly one tile for each clause represents a truth assignment satisfying the given 3-CNF formula (for details see [15]). Such a complete complex for the formula (1) is depicted in Figure 4.

On the other hand, if there is no satisfying truth assignment, complete complexes may still assemble. However the condition that for each clause and each variable there is precisely one corresponding tile in the complex will not be satisfied. Moreover, all such complexes contain larger (integer multiple) number of tiles than the number of clauses and variables. See [16] for an explanation of multiple covers.

Hence, a way to tell if the inputted 3-CNF formula is satisfiable is to determine if there are sufficiently small complete complexes in the pot. In the following we specify what "sufficiently small" means; we start with a system of indexes for guiding the construction of complexes.

Definition 6. *Fix a graph* $\mathfrak{G} = \langle V, E\rangle$*. A pot type* $\mathbf{P}$ *is* properly indexed *by* $\mathfrak{G}$ *if there is a complete complex* $\mathfrak{C}$ *that consists of tiles of distinct types, labeled by vertices* $g \in V$*, such that the map* $t \mapsto \mathrm{label}(t)$ *is an isomorphism of* $\mathfrak{C}$ *onto* $\mathfrak{G}$*.*

Thus a complete complex of a properly indexed pot type (indexed by a graph $\mathfrak{G}$) is isomorphic to $\mathfrak{G}$, or of the form of several copies of $\mathfrak{G}$, all tangled together.

Thus in the case of 3SAT the pot type is properly indexed by the graph corresponding to the formula if and only if the formula has a solution. Moreover, the graph that properly indexes the pot is unique up to an isomorphism. The proof of the following proposition follows from basic topological properties of graphs, and we omit the proof.

Proposition 1. *Let* $\mathbf{P}$ *be a pot type and assume that there is precisely one graph* $\mathfrak{G} = \langle V, E \rangle$ *such that* $\mathbf{P}$ *is properly indexed by* $\mathfrak{G}$*. Suppose that* $\mathfrak{G}$ *is connected. Then any complete complex in* $\mathcal{C}(\mathbf{P})$ *has* $n|V|$ *tiles, for some integer* n*.*

We can design properly indexed pots so that a complete complex witnessing, say, the solution of a problem has $|V|$ tiles, while a spurious complete complex has at least $2|V|$ tiles. For example, a 3-CNF formula generates a pot, which provides a solution (i.e., a satisfying truth assignment) in a proper complex; if there is no satisfying truth assignments, there is no proper complex. Thus a formula of k clauses and m variables is satisfied by some truth assignment iff the corresponding pot admits a complete complex of $k + 2m$ tiles; any larger complexes will have at least $2k + 4m$ tiles.

Definition 7. *A pot type* $\mathbf{P}$ *is* weakly satisfiable within bound b *if it admits a complete complex of at most* b *tiles.*

4 Computing NP Problems by Flexible-Tile Assembly

This section shows how NPTIME computations can be accomplished by (flexible tile) assembly within given bounds. First we fix the nomenclature. Consider a finite alphabet Σ, with a symbol $\# \notin \Sigma$ for "blank." We consider problems of recognizing a formal language L over alphabet Σ, i.e., for an alphabet Σ and a language $L \subseteq \Sigma^*$, given input $x \in \Sigma^*$, is $x \in L$?

Definition 8. *A* Deterministic Turing Machine *is a tuple* $(Q, \Sigma, \delta, q_0, \#)$*, such that:*

- *The set* Q *is the set of* states *of the machine;* q_0 *is the* initial state *and* q_F *is the* terminal state*.*
- *The finite set* Σ *is the* alphabet*, which does not contain the "blank" character* $\#$*.*
- *The* transition function δ *is a partial function* $(Q \times (\Sigma \cup \{\#\})) \to (Q \times \Sigma \times \{L, 0, R\})$*. If* $\delta(q, \sigma) = (q', \sigma', \xi)$*, and if the tapehead is reading* $\sigma \in \Sigma \cup \{\#\}$ *while the machine is in state* q*, it replaces* σ *with* σ' *and moves* L*eft if* $\xi = L$ *or* R*ight if* $\xi = R$ *(or for* $\xi = 0$*, does not move), and changes its state to* q'*. Each such transition is a* step *denoted* $(q, \sigma) \to (q', \sigma', \xi)$*. The machine* halts *in state* q*, reading symbol* σ*, if* $\delta(q, \sigma)$ *is undefined.*
- *The machine starts in the initial state* q_0*, with the tapehead at the leftmost square of the input string* $x \in \Sigma^*$*. If it halts, the input string is accepted.*

The machine is a Non-Deterministic Turing Machine (NTM) *if, for each state* q *and character* σ*,* $\delta(q, \sigma)$ *is a nonempty subset of* $Q \times \Sigma \times \{L, 0, R\}$ *such that a transition is a step* $(q, \sigma) \to (q', \sigma', \xi)$ *if* $(q', \sigma', \xi) \in \delta(q, \sigma)$*.*

In the case of an NTM we assume that the machine proceeds by choosing successive actions out of the available options.

A language $L \subseteq \Sigma^*$ is *PTIME computable* if there exists a DTM M and a polynomial p such that for any input $x \in \Sigma^*$, M halts within $p(|x|)$ steps if

and only if $x \in L$. Call a language $L \subseteq \Sigma^*$ *NPTIME computable* if there exists an NTM M and a polynomial p such that for any input $x \in \Sigma^*$, M can make choices in its computation to eventually halt in accepting state within $p(|x|)$ steps if and only if $x \in L$. These machines are called *PTIME acceptors* or *NPTIME acceptors*, respectively, as they "accept" their respective languages. Incidentally, a PTIME *program* or *algorithm* for converting one type of problem into another is a DTM M (with associated polynomial p) such that for any input x, within $p(|x|)$ steps the content of the tape will be a representation of the desired output.

Definition 9. *The* Flexible Tile Assembly Polynomial time *(or FTAP) class of languages is the class of languages $L \subseteq \Sigma^*$ such that there is a polynomial p and a PTIME algorithm converting any $x \in \Sigma^*$ into a pair $(\mathbf{P}_x, b)$ where:*

- *the pot type $\mathbf{P}_x$ is weakly satisfiable within bound b if and only if $x \in L$, and*
- *the bound b satisfies $b \leq p(|x|)$.*

We use tile assembly to construct a complex representing an entire NTM computation.

A *configuration* of a TM is a tuple (q, m, s), where q is the state of the machine, m is the position of the tapehead, and s is the string currently on the tape. Thus a *computation* of a Turing Machine is a sequence of configuration $C_0, C_1, \ldots$, where C_0 is an initial configuration, and for each i, C_{i+1} is obtained from C_i by a transition step. The computation terminates if and only if machine halts.

Here is the first half of the main theorem.

Theorem 1. *If a language $L \subseteq \Sigma^*$ is NPTIME computable, then it is in FTAP.*

Sketch of proof. Fix a polynomial p, and let L be accepted by the NTM M, which accepts strings of length n within $p(n)$ steps – or not at all. We construct a pot type such that each prototile in the pot type represents one tape square at a position m at a time t. For each input string x, if M accepts x it must do so within $p(|x|)$ steps. Set $t_x = p(|x|)$. Then the number of squares on the tape of the machine that is visited by the head is bounded by $n = 2t_x + 1$. Denote $N_{\text{time}} = \{0, 1, \ldots, t_x\}$ and $N_{\text{tape}} = \{-t_x, -t_x + 1, \ldots, 0, 1, \ldots, t_x\}$.

The set of prototiles is a subset of $(\Sigma \cup \{\#\}) \times N_{\text{tape}} \times N_{\text{time}} \times X^3 \times (Q \cup \{0\})^2$ where $X = \{-1, 0, 1\}$. Note that the number of prototiles is bounded by a polynomial in $|x|$. Intuitively, at time t and position m, there is a symbol σ in the square. The prototile corresponding to this square is $(\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$. We define $\xi_0 = 1$ if the tapehead is at this square at time t, and $\xi_0 = 0$ otherwise. If $\xi_0 = 1$, we define ξ_- and ξ_+ to represent the previous and next moves of the tapehead, otherwise they remain 0. The head had moved from the left ($\xi_- = 1$) or the right ($\xi_- = -1$), and the head will move either to the left ($\xi_+ = -1$) or the right ($\xi_+ = 1$). And if the machine entered square m while in a state q at time t, then it will change to a state q' at time $t + 1$. Finally, if the computation halted at time t, $q' = q$ and $\xi_+ = 0$. If the tapehead is not positioned at the square m at time t, then $(\xi_-, \xi_0, \xi_+) = (0, 0, 0)$ and $(q, q') = (0, 0)$.

Given a prototile $\alpha = (\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$ and a Turing Machine M having transition function δ, we say that α is *consistent* with M if:

- $\xi_0 = 0$, implies $\xi_- = \xi_+ = 0$ and $q = q' = 0$,
- $\xi_0 = 1$ if $\delta(q, \sigma) = (q', \sigma', \xi)$ for $\xi \in \{R, 0, L\}$ and some $\sigma' \in \Sigma$.

Our pot type $\mathbf{P} = \mathbf{P}_M$ consists of the maximal set of prototiles consistent with M such that their second and third entries satisfy $m \in N_{\text{tape}}$ and $t \in N_{\text{time}}$.

There are eight codes for ports available, of which any particular tile may have degree from two to six. The degree of a tile $\alpha = (\sigma, m, t, (\xi_-, \xi_0, \xi_+), (q, q'))$ is determined by

$$d(\alpha) = \begin{cases} 2 \text{ if } m \in \{0, n\} \text{ and } t \in \{0, t_x\} & \text{if a corner tile;} \\ 3 \text{ if } m \in \{0, n\} \text{ xor } t \in \{0, t_x\} & \text{if a side tile;} \\ 4 \text{ if } \xi_0 = 0 & \text{if a central tile with no tapehead;} \\ 5 \text{ if } q' = q_F & \text{if the computation halts;} \\ 6 \text{ if } \xi_0 = 1 \text{ and } q' \neq q_F & \text{if the tapehead is present, not halting.} \end{cases}$$

We note that there are other kinds of tiles that encode specific boundary conditions, such as the head reaching a corner during the computation, but we do not include these details. Recall that a tile represents a particular mth square of the Turing machine at a particular time t. The idea is that successive rows of tiles represent successive configurations of the Turing Machine, starting with the initial (northernmost) row, and heading south. Several typical tiles in the computation assembly are depicted in Figure 5. Comparing to the simulation of a TM by rigid tiles (see for ex. [36,34]) one has to be careful to "force" the assembly into a rectangular grid using port codes. Each code indicates the current position of a given tile at the computation process of the Turing Machine (i.e. position m and time step t) as well as when necessary additional information such as the symbol, state and movement of the machine at a particular instance. This is depicted in Figure 5. The complementary ports are indicated by θ. Note that when the complex assembles, the "top" row represents the initial configuration with tiles encoding x on squares $0, \ldots, |x| - 1$. The rest of the tiles in the "top" row carry the blank character #, the head is on square 0 in state q_0.

The tiles assemble into a rectangular grid, with each horizontal row of tiles giving a configuration of the machine, with successive rows southward representing successive configurations, and each vertical column representing a particular square. The diagonal connections represent motions of the head. The tiles are designed so that there are no north, south, east, or west ports off the end of the complex, so the complex will be complete if and only if there are no diagonal ports sticking out the bottom, i.e., if and only if the computation has halted within t_x steps.

So suppose L is NPTIME computable, i.e., there is a polynomial p such that for any x, $x \in L$ if and only if the NTM accepting L halts in a terminal state within $p(|x|)$ steps. If $x \in L$, $t_x = p(|x|)$ gives us a complete complex of at most $b = (t_x + 1)(2t_x + 1) = 2p(|x|)^2 + 3p(|x|) + 1$ tiles, a polynomial bound. If $x \notin L$, there is no complete complex at all. ■

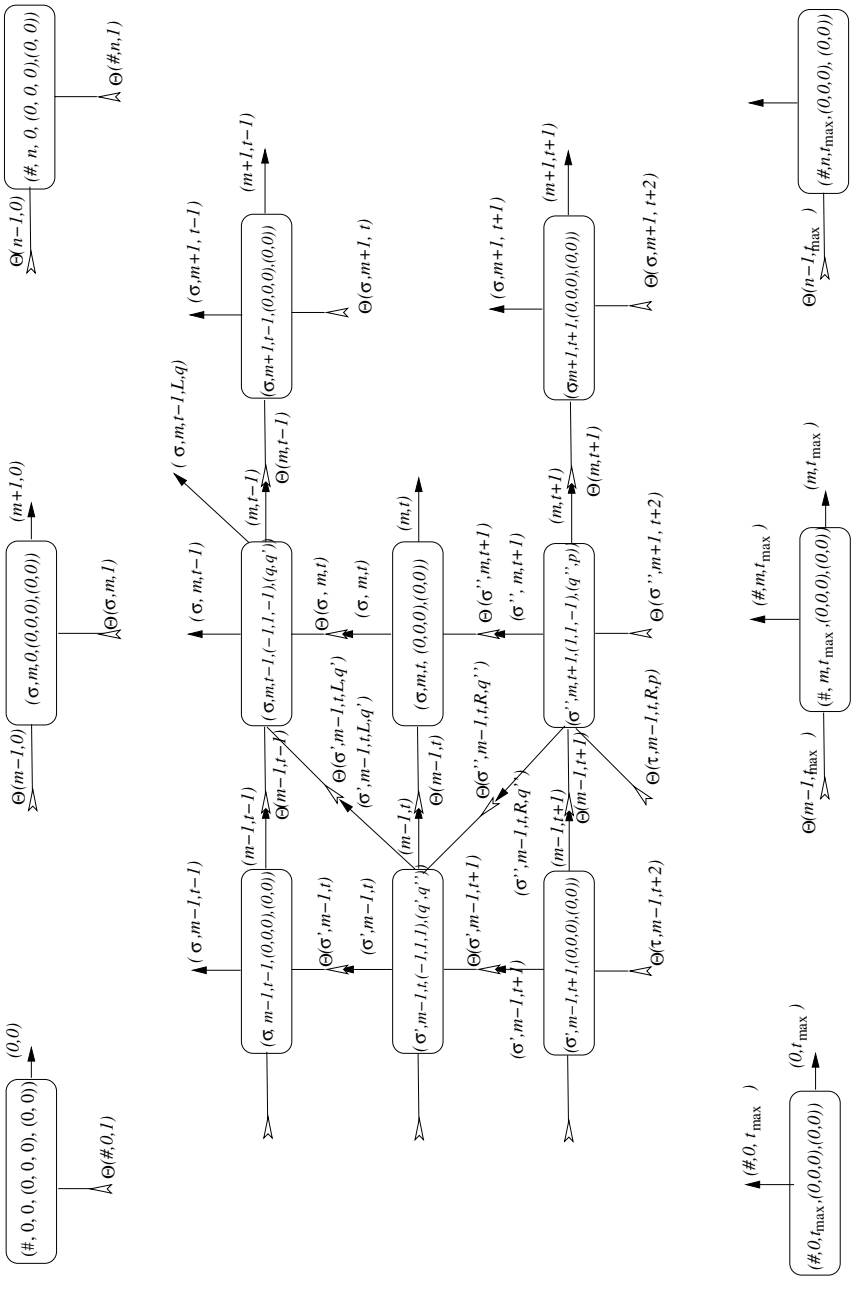

Fig. 5. Portions of assembly of tiles from P_M consistent with the Turing Machine M

5 Flexible-Tile Assembly Is Within NPTIME

We now go the other direction, proving that all weakly satisfiable problems within bounds are NPTIME-computable. We will use the Integer Programming Problem, called MP1 in [12], that was proved to be NP-computable by Borosh & Treybig [4] and NP-hard by Karp [20].

Definition 10. *This is the Integer Programming Problem (***MP1***).*

- INPUT:
  - I1. A set finite $X \subseteq \mathbb{Z}^m \times \mathbb{Z}$ of $(m+1)$-tuples $(\boldsymbol{x}, b)$ of integers.
  - I2. An m-tuple $\boldsymbol{c} \in \mathbb{Z}^m$.
  - I3. An integer B.
- QUESTION: Does there exist an m-tuple $\boldsymbol{y} \in \mathbb{Z}^m$ such that the following is true?
  - Q1. For each $(\boldsymbol{x}, b) \in X$, $\boldsymbol{x} \cdot \boldsymbol{y} \leq b$, where "$\cdot$" is vector dot or inner product.
  - Q2. And $\boldsymbol{c} \cdot \boldsymbol{y} \geq B$.

Theorem 2. *All FTAP computable problems are NPTIME-computable.*

Idea of Proof. We prove that there is a PTIME reduction of pot types to MP1 problems that are solvable if and only if the original pot type was weakly satisfiable within bounds.

Given a pot type $\mathbf{P}$ over (H, θ), we will reduce the problem to whether there is a set of nonnegative integers (tile multiplicities) $\{y_{\mathbf{t}} \in \mathbb{N}\colon \mathbf{t} \in \mathbf{P}\}$, not all zero, giving us the number of each kind of tile in a complete complex. Considering each tile type as a vector with non-negative integer entries, we treat the pot $\mathbf{P}$ as a set of vectors of integers.

We need an input as in Definition 10 that is solvable iff there exists a system of integers $y_{\mathbf{t}}$ such that:

(i) Each $y_{\mathbf{t}}$ has $y_t \geq 0$, i.e., a number of tiles of a particular type.
(ii) For each $\mathbf{h}$, $\sum_{\mathbf{t}\in\mathbf{P}} y_{\mathbf{t}} \cdot \mathbf{t}(\mathbf{h}) = \sum_{\mathbf{t}\in\mathbf{P}} y_{\mathbf{t}} \cdot \mathbf{t}(\hat{\mathbf{h}})$, i.e., $\mathbf{P}$ can generate a complete complex.
(iii) $\sum_{\mathbf{t}\in\mathbf{P}} y_{\mathbf{t}} > 0$, i.e., the complete complex that is generated is non-trivial, contains at least one tile.
(iv) $\sum_{\mathbf{t}\in\mathbf{P}} y_{\mathbf{t}} \leq b$, i.e., the complete complex that is generated contains at most b tiles.

We rearrange the above inequalities to fit those that appear as input in Definition 10. ■

6 Final Remarks

This paper suggests at least two collections of problems that are yet to be considered.

First, while this model provides a method for computing NP problems in theory, it is not entirely clear how well it will work in practice. So far, laboratory experiments have determined that flexible tile computations would perform correctly for a 3-DNF propositional calculus formula of perhaps five clauses; it is not clear how well it would determine if a 5,000-clause formula is satisfiable (and a problem of this size is, in general, beyond our current computational capacities). The actual behavior of this model should be investigated further.

Second, this model suggests a set of logics describing computational complexity classes. The main theorem of this paper is a "representation theorem" similar to the main theorem of [9] (see [14] or [8], which presented a representation of NPTIME somewhat similar the one presented here. The combinatorial properties of this model, and related models, may provide additional tools for investigating some of the more obstinate classes in computational complexity theory.

Acknowledgement. This work has been supported in part by NSF Grants CCF #0432009 and EIA#0086015.

References

1. L. Adleman, *Molecular computation of solutions of combinatorial problems*, Science **266** (1994) 1021-1024.
2. L.M. Adleman, Q. Cheng, A. Goel, M-D. Huang, D. Kempe, P. Moisset de Espanés, P.W.K. Rothemund. *Combinatorial optimization problems in self-assembly*, *STOC'02 Proceedings*, Montreal Quebec, Canada, 2002.
3. L.M. Adleman, J. Kari, L. Kari, D. Reishus, *On the decidability of self-assembly of infinite ribbons*, *Proceedings of FOCS 2002*, IEEE Symposium on Foundations of Computer Science, Washington (2002) 530-537.
4. I. Borosh & L. B. Treybig, *Bounds on the positive integral solutions of linear Diophantine equations*, *Proc. Amer. Math. Soc.* **55** (1976) 299-304.
5. R. S. Braich, N. Chelyapov, C. Johnson, P.W.K. Rothemund, L. Adleman, *Solution of a 20-variable 3-SAT problem on a DNA Computer*, *Science* **296** (2002) 499-502.
6. J.H. Chen, N.C. Seeman, *Synthesis from DNA of a molecule with the connectivity of a cube, Nature* **350** (1991) 631-633.
7. S. Cook, *The complexity of theorem proving procedures*, *Proc. 3rd. ACM Ann. Symp. Theory of Comput.* (1971) 151-158.
8. H.-D. Ebbinghaus, J. Flum, *Finite Model Theory* (Springer, 2nd. ed., 1999).
9. R. Fagin, *Contributions to the Model Theory of Finite Structures*, UC Berkeley Ph.D. Thesis, 1973.
10. D. Faulhammer, A.R. Curkas, R.J. Lipton, L.F. Landweber, *Molecular computation: RNA solution to chess problems*, *PNAS* **97** (2000) 1385-1389.
11. T.J. Fu, N.C. Seeman, *DNA double crossover structures*, *Biochemistry* **32** (1993) 3211-3220.
12. M. R. Garey, D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. H. Freeman, 1979.
13. T. Head et.al, *Computing with DNA by operating on plasmids*, *BioSystems* **57** (2000) 87-93.
14. N. Immerman, *Descriptive Complexity Theory*, (Springer-Verlag, 1999).

15. N. Jonoska, P. Sa-Ardyen, N.C. Seeman, *Computation by self-assembly of DNA graphs*, *Genetic Programming and Evolvable Machines* **4** (2003) 123-137.
16. N. Jonoska, S. Karl, M. Saito, *Three dimensional DNA structures in computing, BioSystems* **52** (1999) 143-153.
17. N. Jonoska, S. Karl, M. Saito, *Creating 3-dimensional graph structures with DNA* in: *DNA based computers III* (H. Rubin, D. Wood, eds.), AMS DIMACS series **vol 48** (1999) 123-136.
18. N. Jonoska, G. McColm, A. Staninska, *Expectation and Variance of Self-Assembled Graph Structures*, *Proceedings of 11th International Meeting on DNA Computing*, London, Ontario, Canada, June 6-9 (2005) 49-58.
19. N. Jonoska, G. McColm, *On a Model of Computation by Self-Assembly*, in preparation.
20. R.M. Karp, *Reducibility among combinatorial problems*, in: *Complexity of Computer Computations* (R. E. Miller, W. E. Thatcher, eds.), Plenum Press (1972) 85-103.
21. M-Y. Kao, V. Ramachandran, *DNA Self-Assembly For Constructing 3D Boxes.* in: *Algorithms and Computation: ISAAC 2001 Proceedings*, Springer LNCS **2223** (2001) 429-440.
22. T.H. LaBean, H. Yan, J. Kopatsch, F. Liu, E. Winfree, J.H. Reif, N.C. Seeman, *J. Am. Chem. Soc.* **122** (2000) 1848-1860.
23. J.M. Lehn, *Sopramolecular Chemistry*, *Science* **260** (1993) 1762-1763.
24. J.M. Lehn, *Toward complex matter: Supramolecular chemistry and self-organization*, *Proceedings of the National Academy of Science USA* **99** No.8 (2002) 4763-4768.
25. C. Mao, T.H. LaBean, J.H. Reif, N.C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules*, *Nature* **407** (2000) 493-496.
26. C. Mao, W. Sun, Z. Shen, N.C. Seeman, *A nanomechanical device based on the B-Z transition of DNA*, *Nature* **397** (1999) 144-146.
27. J. H. Reif, S. Sahu, P. Yin, *Complexity of Graph Self-Assembly in Accretive Systems and Self-Destructive Systems*, *Proceedings of 11th International Meeting on DNA Computing*, London, Ontario, Canada, June 6-9 (2005) 101-112.
28. P.W.K. Rothemund, E. Winfree, *The Program-Size Complexity of Self-Assembled Squares*, *Proceedings of 33rd ACM meeting STOC 2001*, Portland, Oregon, May 21-23 (2001) 459-468.
29. P. Rothemund, N. Papadakis, E. Winfree, *Algorithmic Self-assembly of DNA Sierpinski Triangles*, PLoS Biology **2** (12) e424, 2004, (13 pages)
30. P. Sa-Ardyen, N. Jonoska, N. Seeman, *Self-assembly of graphs represented by DNA helix axis topology J. Am. Chem. Soc.* **126**(21) (2004) 6648-6657.
31. N.C. Seeman, *DNA junctions and lattices*, *J. Theor. Biol.* **99** (1982) 237-247.
32. N.C. Seeman, *DNA nicks and nodes and nanotechnology*, *NanoLetters* **1** (2001) 22-26.
33. W.M. Shihn J.D. Quispe, G.F. Joyce, *A 1.7-kilobase single-stranded DNA folds into a nano-scale octahedron*, *Nature* **427**, Feb. 12 (2004) 618-621.
34. D. Soloveichik, E. Winfree, *Complexity of Self-Assembled Shapes*, preprint at http://arxiv.org/abs/cs.CC/0412096.
35. Y. Wang, J.E. Mueller, B. Kemper, N.C. Seeman, *The assembly and characterization of 5-arm and 6-arm DNA junctions*, *Biochemistry* **30** (1991) 5667-5674.
36. E. Winfree, X. Yang, N.C. Seeman, *Universal computation via self-assembly of DNA: some theory and experiments*, in: *DNA based computers II* (L. Landweber, E. Baum eds.), AMS DIMACS series **44** (1998) 191-214.

37. E. Winfree, F. Liu, L.A. Wenzler, N.C. Seeman, *Design and self-assembly of two-dimensional DNA crystals*, *Nature* **394** (1998) 539-544.
38. H. Yan, X. Zhang, Z. Shen and N.C. Seeman, *A robust DNA mechanical device controlled by hybridization topology*, *Nature* **415** (2002) 62-65.
39. B. Yurke, A.J. Turberfield, A.P. Mills, F.C. Simmel Jr., *A DNA fueled molecular machine made of DNA*, *Nature* **406** (2000) 605-608.
40. Y. Zhang, N.C. Seeman, *The construction of a DNA truncated octahedron*, *J. Am. Chem. Soc.* **160** (1994) 1661-1669.