Reducing Tileset Size: 3-SAT and Beyond

Yuriy Brun
Department of Computer Science
University of Southern California
Los Angeles, CA 90089
Email: ybrun@usc.edu

Abstract

In self-assembly research, reducing the number of distinct tiles necessary to compute functions can make it feasible to implement tile systems to solve complex problems. Existing methods for solving 3-SAT, a well-known NP-complete problem, in the tile assembly model involve either using $\Theta(n^2)$ distinct tiles to nondeterministically decide whether an n-variable Boolean formula is satisfiable or simulating a cellular automata system simulating a Turing machine, which requires a constant but large number of tiles to deterministically make the decision. Here, I propose a system that solves k-SAT nondeterministically, for all $k \in \mathbb{N}$, in time linear in the input size using only 64 distinct tiles. Further, I propose a mechanism for converting the tilesets of tile systems for many NP-complete and other problems from tilesets whose size depends on the input into $\Theta(1)$-size tilesets.

1 Introduction

Self-assembly is the study of how simple objects come together to form more complex objects. In nature, self-assembly occurs on all scales, from atoms self-assembling into molecules to stars and planets to form galaxies. Crystal growth is just one example of self-assembly, and one that has been studied at length and shown capable of computing mathematical functions [1]. Molecular systems can also compute, in particular DNA systems [2], and solve NP-complete problems [3, 4]. This idea of using molecules to compute nondeterministically is the driving motivation behind my work.

NP-complete problems are integral to our everyday lives and yet we know of no efficient algorithms to solve them. Many problems in the realms of resource allocation, scheduling, and protein folding have been shown to be NP-hard, and the ability to solve these problems quickly is highly desirable. Molecular computing models are a possible powerful route toward feasible algorithms for solving NP-complete problems quickly because of high information density molecules allow.

Winfree showed that DNA computation is Turing-universal [5]. While DNA computation suffers from relatively high error rates, the study of self-assembly shows how to utilize redundancy to design systems with built-in error correction [6, 7, 8, 9, 10]. Researchers have used DNA to assemble crystals with patterns of binary counters [11] and Sierpinski triangles [12], but while those crystals are deterministic, generating nondeterministic crystals may hold the power to solving complex problems quickly.

Two important questions about self-assembling systems that create shapes or compute functions are: "what is a minimal tile set that can accomplish this goal?" and "what is the minimum assembly time for that system?" Here, I study systems that solve NP-complete problems and ask these questions, as well as another that is important to nondeterministic computation: "what is the probability of assembling the crystal that encodes the solution?" Adleman has emphasized studying the number of steps it takes for an assembly to complete (assuming maximum parallelism) and the minimal number of tiles necessary to assemble a shape [13]. He answered these

questions for n-long linear polymers [14]. Previously, I have extended these questions to apply to computing tile systems [15].

I proposed and studied systems that compute the sums and products of two numbers using the tile assembly model [15]. I then showed that systems can be combined to create systems with more complex behavior, and designed a system that nondeterministically factors numbers [16]. I have also shown a system that solves the NP-complete problem *SubsetSum* using a system with 49 different computational tile types [17]. One interesting aspect of solving satisfiability in the tile assembly model using a constant-size tileset is that writing the input itself is difficult because each Boolean formula has some n distinct variables, and one must encode all variables using a constant number of tiles. Previous attempts have used distinct tiles for each variable. The mechanism I design here can be applied to solving other problems that require an encoding of variables in its input, as well as graph problems that require the encoding of vertices and edges.

The work closest to mine on solving satisfiability using tiles is a proposal by Lagoudakis et al. [18]. They informally define two systems that nondeterministically compute whether or not an n-variable boolean formula is satisfiable using $\Theta(n^4)$ and $\Theta(n^2)$ distinct tiles, respectively. The former system encodes each clause-variable pair as a separate tile, and the latter system encodes each pair of literals as a separate tile. In a DNA implementation of even the smaller system, to solve a 50-variable satisfiability problem, one would need on the order of 2500 different DNA complexes, while current DNA self-assembly systems have on the order of 10 different complexes. In contrast, the system I present in this paper for solving an NP-complete problem uses $\Theta(1)$ distinct tiles and assembles in time linear in the input. Section 1.2 will describe the $\Theta(n^2)$-sized tileset system in detail.

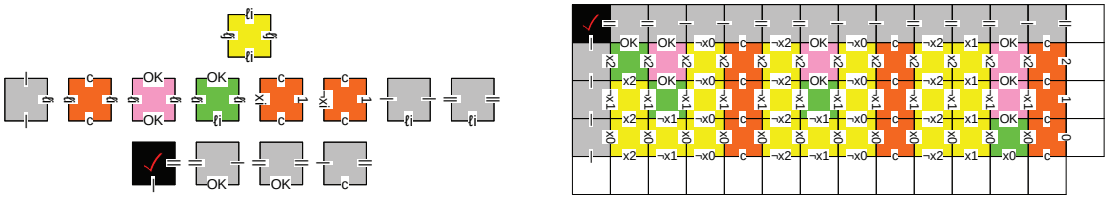
1.1 Tile Assembly Model

The tile assembly model [1, 19] is a formal model of crystal growth. It was designed to model self-assembly of molecules such as DNA. It is an extension of a model proposed by Wang [20]. The model was fully defined by Rothemund and Winfree [19], and the definitions I use are similar to those. Full formal definitions can be found in [17].

Intuitively, the model has *tiles*, or squares, that stick or do not stick together based on various *binding domains* on their four sides. Each tile has a binding domain on its north, east, south, and west side. The four binding domains, elements of a finite alphabet Σ, define the type of the tile. The strength of the binding domains are defined by the *strength function* g. The placement of some tiles on a 2-D grid is called a *configuration*, and a tile may *attach* in empty positions on the grid if the total strength of all the binding domains on that tile that match its neighbors exceeds the current *temperature* (a natural number). Finally, a *tile system* $\mathbb{S}$ is a triple $\langle T, g, \tau \rangle$, where T is a finite set of tiles, g is a strength function, and $\tau \in \mathbb{N}$ is the temperature, where $\mathbb{N} = \mathbb{Z}_{\geq 0}$.

Starting from a *seed configuration* S, tiles may attach to form new configurations. If that process terminates, the resulting configuration is said to be *final*. At some times, it may be possible for more than one tile to attach at a given position, or there may be more than one position where a tile can attach. If for all sequences of tile attachments, all possible final configurations are identical, then $\mathbb{S}$ is said to produce a *unique* final configuration on S. The *assembly time* of the system is the minimal number of steps it takes to build a final configuration, assuming maximum parallelism.

In solving NP-complete problems it is important to compute a particular subset of functions, the characteristic functions of subsets of the natural numbers. A characteristic function of a set has value 1 on arguments that are elements of that set and value 0 on arguments that are not elements of that set. Typically, in computer science, programs and systems that compute such functions are said to decide the set. I adapt the definition of nondeterministically computing functions from [16] to nondeterministically deciding subsets of natural numbers. Since for all



(a) (b)

Figure 1: Solving 3-SAT using $\Theta(n^2)$ distinct tiles. The concepts behind the tiles in T_n^2 (a) indicate that for every $n \in \mathbb{N}$, the set T_n^2 contains $4n^2 + 12n + 4$ tiles: the four bottom-row tiles, including the special ✓ tile, $14n$ middle-row tiles, with ℓi enumerating over all the literals and $0 \leq i < n$, and $4n^2 - 2n$ top row tiles, with ℓi enumerating over all the literals and ℓj enumerating over all the literals such that $\ell i \neq \ell j$. Tiles from T_n^2 attach to a seed to nondeterministically select a truth assignment (b). Here $\phi = (x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$ and the literal selection is $x_0, \neg x_1, x_2$ meaning that $x_0 = x_2 = TRUE$ and $x_1 = FALSE$. Tiles attach to allow the ✓ tile to attach in the northwest corner iff the assignment satisfies the Boolean formula encoded by the seed.

constants $n \in \mathbb{N}$, the cardinalities of $\mathbb{N}^n$ and $\mathbb{N}$ are the same, one can encode an element of $\mathbb{N}^n$ as an element of $\mathbb{N}$. Thus it makes sense to talk about deciding subsets of $\mathbb{N}^n$. Let $\Omega \subseteq \mathbb{N}^m$ be a set. A tile system $\mathbb{S} = \langle T, g, \tau \rangle$ *nondeterministically decides* Ω with identifier tile $r \in T$ iff for all $\vec{a} \in \mathbb{N}^m$, there exists a seed configuration S that encodes $\vec{a}$ and for all final configurations F that $\mathbb{S}$ produces on S, $r \in F(\mathbb{Z}^2)$ iff $\vec{a} \in \Omega$, and there exists at least one final configuration F with r attached. In other words, the *identifier* tile r attaches to one or more of the nondeterministic executions iff the seed encodes an element of Ω. I call the set of tiles used to encode the input Γ.

I have given informal definitions to assist the reader in understanding the system I discuss in this paper and I refer the reader to [17] for more formal definitions.

1.2 Background and Work Related to Solving 3-*SAT*

The Boolean satisfiability (SAT) problem is a well-known NP-complete problem. Let $n \in \mathbb{N}$, then for all $0 \leq i < n$, let x_i be a Boolean variable that can take on values from the set $\{TRUE, FALSE\}$. Let the set of literals be the set of those variables and their negations $(\bigcup_i \{x_i, \neg x_i\})$, where $\neg TRUE = FALSE$, and $\neg FALSE = TRUE$. A clause is a disjunction of literals, e.g., $(x_0 \vee \neg x_1 \vee x_2)$. A Boolean formula, in conjunctive normal form (CNF), is a conjunction of clauses, e.g., $(x_0 \vee \neg x_1 \vee x_2) \wedge (\neg x_3 \vee x_2 \vee \neg x_2)$. If every clause has exactly k literals, the Boolean formula is said to be in kCNF. A Boolean formula is satisfiable iff there exists some assignment of each variable to an element of $\{TRUE, FALSE\}$ such that the formula evaluates to $TRUE$.

The notions of truth assignment and literal selection are in some sense parallel, and I will use them somewhat interchangeably in this paper. Formally, every literal selection corresponds to a truth assignment. Thus I will sometimes use a literal selection, e.g., $x_0, \neg x_1, x_2$, to specify a truth assignment, in this case $x_0 = x_2 = TRUE$ and $x_1 = FALSE$.

The k-SAT problem is, given a kCNF Boolean formula, to determine whether or not it is satisfiable. It is well known that 1-SAT and 2-SAT can be solved in polynomial time, while each k-SAT for $k \geq 3$ is NP-complete. Formally, k-SAT is the set of all Boolean formula in kCNF that are satisfiable. To solve k-SAT means to decide the set k-SAT.

Lagoudakis et al. proposed a tile system for nondeterministically deciding whether a 3CNF Boolean formula is satisfiable [18]. This system uses $\Theta(n^2)$ distinct tiles for a formula with n distinct variables, and can be adapted to decide the satisfiability of k-SAT for arbitrary

$k \in \mathbb{N}$. While Lagoudakis et al. do not formally define what it means for a tile system to solve a problem or compute a function and do not formally argue that their system does in fact solve satisfiability, I believe their system can be made to fit my definitions and proven correct. What I present here is a slight variant of their system, which follows the same basic logic. My approach, described in Section 2 will build on the techniques in this system while reducing the tileset from $\Theta(n^2)$ to $\Theta(1)$, thus understanding (or accepting the correctness of) this system is crucial to understanding my modifications and contributions.

I now describe a family of tile systems that determine whether a Boolean formula with $n \in \mathbb{N}$ distinct variables is satisfiable (because the size of the system depends on the number of variables, there will be a different system for every n, and thus I refer to the systems for all n as a family). I will refer to these systems as $\mathbb{S}_n^2$.

The idea behind encoding the input is to encode the Boolean formula in the 0^{th} row of the seed configuration with a unique tile for each possible literal, prefixing each clause with a special clause tile, and to encode the variables in the 0^{th} column with a unique tile for each variable. There are also three helper tiles used to mark the clauses and the ends of the input. For a given $n \in \mathbb{N}$, the set Γ_n^2 contains the three helper tiles, $2n$ literal tiles, and n variable tiles. Thus $|\Gamma_n^2| = 3n + 3$.

I will use the tiles in Γ_n^2 to encode an n-variable Boolean formula in a specific way. I will place tiles representing the formula's literals in the 0^{th} row such that the literals of each clause are together, place the special clause tile to the east of each clause, place the variable tiles in the 0^{th} column, and place special end tiles in the west-most and north-most positions on that row and column. The clear tiles in Figure 1(b) show a sample seed encoding the 3-variable Boolean formula $(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$ using the tiles from Γ_3^2. Note that while this example tries to follow some logical order, the order of the variables, the clauses, and the literals within each clause is not important.

The main idea of the computation is to have tiles attach nondeterministically in column -1 to select either *TRUE* or *FALSE* for each variable and then to "sweep" those choices westward, checking if a literal in a clause evaluates to *TRUE*. Whenever a literal evaluates to *TRUE*, that information propagates northward, and along the top row, tiles attach to ensure that at least one literal in every clause evaluates to *TRUE*. Iff that is the case, a special ✓ tile attaches in the northwest corner.

The system $\mathbb{S}_n^2$ will use the set of computational tiles T_n^2. Figure 1(a) shows the concepts behind the tiles in T_n^2. The bottom row shows four helper tiles, including the special ✓ tile, that are the same for all n. For each tile in the middle row there will be $\Theta(n)$ tiles in T_n^2, with ℓi enumerating over all the literals and $0 \leq i < n$. Finally, the top row shows the concept behind the tile which will expand to $\Theta(n^2)$ tiles in T_n^2, with ℓi enumerating over all the literals and ℓj enumerating over all the literals such that $\ell i \neq \ell j$. Thus, for a given $n \in \mathbb{N}$, the set T_n^2 contains the four bottom-row tiles, $14n$ middle-row tiles, and $4n^2 - 2n$ top-row tiles. Thus $|T_n^2| = 4n^2 + 12n + 4$. Note that Lagoudakis et al. claim that their systems use $2n^2 + 12n + 10$ distinct tiles, but after careful analysis, I disagree with their calculations and believe their systems actually use significantly more tiles, even more than my system's $4n^2 + 12n + 4$. Most notably, their analysis assumes that there are $2n^2 - n$ tiles identical to my yellow tiles, whereas in reality there are $4n^2 - 2n$ such tiles. The tiles of T_3^2 attach to a seed configuration to nondeterministically select a truth assignment of the variables and check if that assignment satisfies the Boolean formula, as shown in Figure 1(b). If and only if the assignment is satisfiable, the ✓ tile attaches in the northwest corner.

Theorem 1 *For all $n \in \mathbb{N}$, let $\Sigma_n^2 = \{\mathsf{c}, |, ||, i, \mathsf{x}_i, \neg\mathsf{x}_i, \mathrm{OK}\}$, where $0 \leq i < n$. Let T_n^2 be as defined in Figure 1(a). Let $g_n^2 = 1$ and $\tau_n^2 = 2$. Let $\mathbb{S}_n^2 = \langle T_n^2, g_n^2, \tau_n^2 \rangle$. Then $\mathbb{S}_n^2$ nondeterministically decides k-SAT (for all $k \in \mathbb{N}$) with up to n distinct variables per formula with the black ✓ tile from T_n^2 as the identifier tile.*

I refer the reader to [21] for the full proof of this theorem, as well as proofs of the facts that the system's assembly time is linear in the size of the input and that the probability that a single nondeterministic execution of $\mathbb{S}_n^2$ succeeds in attaching a ✓ tile if ϕ is satisfiable is at least $\left(\frac{1}{2}\right)^n$.

2 Reducing Tileset Size

I now describe a nondeterministic tile system $\mathbb{S}_{SAT}$, which will follow a logic similar to that of $\mathbb{S}_n^2$, but will use only a constant number of tiles. The idea of $\mathbb{S}_{SAT}$ is to encode ϕ in the same way $\mathbb{S}_n^2$ did, but instead of using a single tile for each literal, the literals will be encoded by a tile that indicates whether the literal is a negation (I place this tile in the east-most position), and a series of 0 and 1 tiles encoding, in binary, the index of the variable. For example, the literal x_5 would be encoded by a v tile, and by a 1 tile, then a 0 tile, and then a 1 tile (101v) because $5 = 101_2$. The literal $\neg x_4$ would be encoded by a ¬v tile, and by a 1 tile, then a 0 tile, and then a 0 tile (100¬v) because $4 = 100_2$. For consistency, I will use the same number of bits to encode all variables, e.g., if my ϕ has 7 distinct variables, I will need three bits to encode x_7, so I will encode x_1 as 001v. Similarly, I will encode the variables in the 0^{th} column using this binary encoding method. The assemblies in $\mathbb{S}_{SAT}$ will be larger in size than the assemblies in $\mathbb{S}_n^2$ because what used to be encoded by a single tile will now be represented by a $\Theta(\log n) \times \Theta(\log n)$ block of tiles, but the overall logic will remain the same. The key to $\mathbb{S}_{SAT}$ is building the logic of the blocks to correctly match literals without affecting the inherited logic of $\mathbb{S}_n^2$.

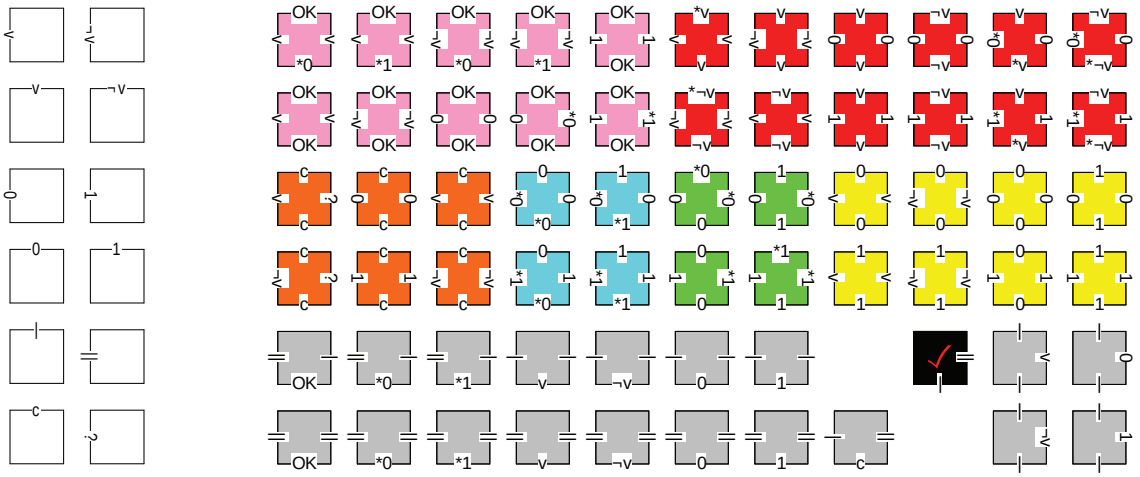
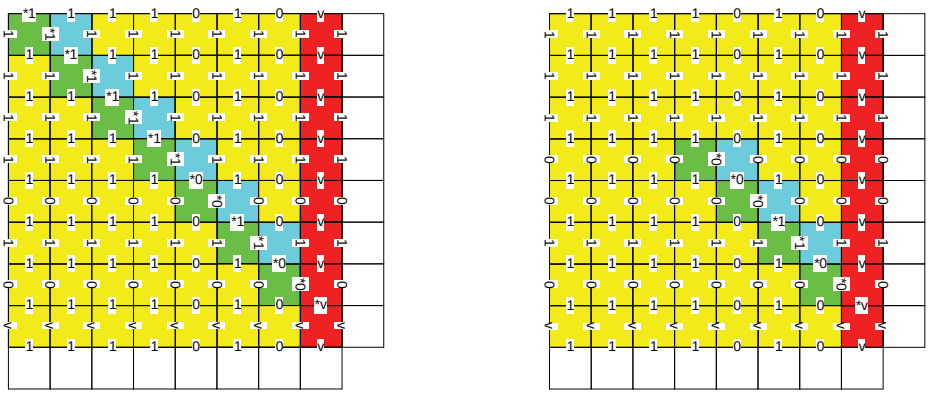


Figure 2: The 12 tiles in Γ_{SAT} (a) and the 64 tiles in T_{SAT} (b).

There are 12 tiles in Γ_{SAT}, no matter how large ϕ is or how many variables it contains. I will use the tiles in Γ_{SAT} to encode an n-variable Boolean formula in a specific way. I will encode the formula's literals in the 0^{th} row, as described above, such that the literals of each clause are together, place the special clause tile to the east of each clause, place the encoded variables in the 0^{th} column, and place special end tiles in the west-most and north-most positions on that row and column. The clear tiles in Figure 4 show a sample seed encoding the 3-variable Boolean formula $(x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$ using the tiles from Γ_{SAT}. Note that while this example tries to follow some logical order, the order of the variables, the clauses, and the literals within each clause is not important.

Just as before, the main idea of the computation is to have tiles attach nondeterministically in column -1 to select either *TRUE* or *FALSE* for each variable and then to "sweep" those choices westward, checking if a literal in a clause evaluates to *TRUE*. The comparison of literals will take place within a $\Theta(\log n) \times \Theta(\log n)$ block of tiles. Whenever a literal evaluates to *TRUE*,



(a) (b)

Figure 3: Tiles comparing two inputs. In (a), the comparison is between 1111010v and 1111010v. Because the two inputs are the same, the northwest tile's north binding domain contains a $\star$, and none of the rest of the exposed binding domains do. In (b), the comparison is between 1111010v and 1110010v. Because the two inputs do not match, no exposed binding domain contains a $\star$.

that information propagates northward, and along the top row, tiles attach to ensure that at least one literal in every clause evaluates to *TRUE*. Iff that is the case, a special ✓ tile attaches in the northwest corner.

The system $\mathbb{S}_{SAT}$ will use the set of computational tiles T_{SAT}. Figure 2(b) shows the 64 tiles in T_{SAT}. The colors of the tiles are coordinated with the colors of the T_n^2 system — the tiles of the same colors perform the same functions.

To reduce the number of tiles from $\Theta(n^2)$ to $\Theta(1)$, I used a technique illustrated in Figure 3. The system illustrated here compares two variable labels. Figure 3(a) shows a comparison of 1111010v and 1111010v. Because the two inputs are the same, the northwest tile's north binding domain contains a $\star$, and none of the rest of the exposed binding domains do. Figure 3(b) shows a comparison of 1111010v and 1110010v. Because the two inputs do not match, no exposed binding domain contains a $\star$.

The tiles of T_{SAT} attach to a seed configuration to nondeterministically select a truth assignment of the variables and check if that assignment satisfies the Boolean formula, as shown in Figure 4.

Theorem 2 *Let* $\Sigma_{SAT} = \{\mathsf{c}, ?, |, ||, \mathsf{v}, {}^\star\mathsf{v}, \neg\mathsf{v}, {}^\star\neg\mathsf{v}, 0, {}^\star0, 1, {}^\star1, \text{OK}\}$. *Let* T_{SAT} *be as defined in Figure 2(b). Let* $g_{SAT} = 1$ *and* $\tau_{SAT} = 2$. *Let* $\mathbb{S}_{SAT} = \langle T_{SAT}, g_{SAT}, \tau_{SAT}\rangle$. *Then* $\mathbb{S}_{SAT}$ *nondeterministically decides k-SAT (for all* $k \in \mathbb{N}$*) with the black ✓ tile from* T_{SAT} *as the identifier tile.*

I refer the reader to [21] for the full proof of this theorem, as well as proofs of the facts that the system's assembly time is linear in the size of the input and that the probability that a single nondeterministic execution of $\mathbb{S}_{SAT}$ succeeds in attaching a ✓ tile if ϕ is satisfiable is at least $\left(\frac{1}{2}\right)^n$.

In summary, $\mathbb{S}_{SAT}$ decides whether a kCNF Boolean formula ϕ on n variables is in k-*SAT*, has $64 = \Theta(1)$ computational tile types, and uses $12 = \Theta(1)$ tile types to encode the input. It computes in time linear in the size of the input, and each assembly has the probability of at least $\left(\frac{1}{2}\right)^n$ of finding the satisfying assignment, if one exists.

2.1 Reducing Other Tile Systems' Tilesets

Satisfiability is just one of many computational problems that require unique identifiers to encode the input. In satisfiability, the n variables of a Boolean formula must be unique. Most graph

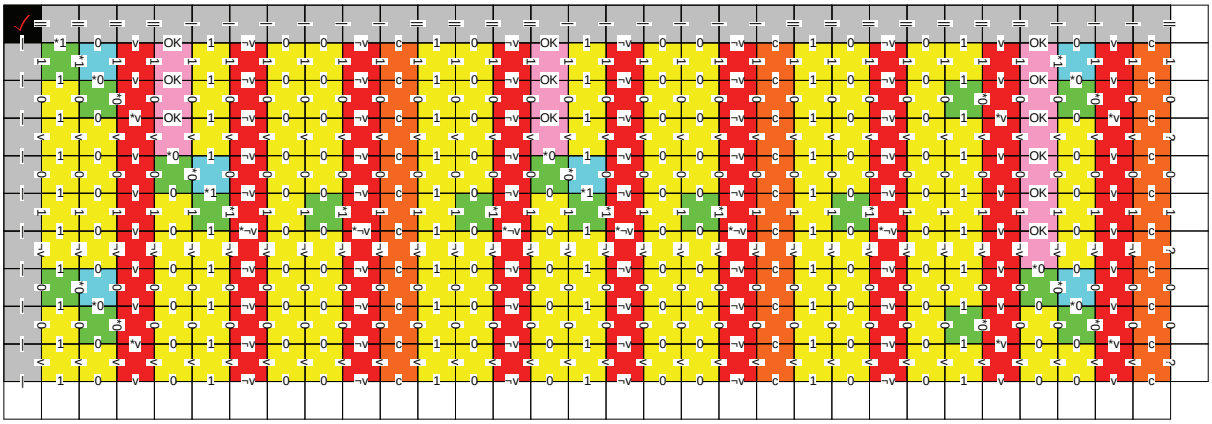


Figure 4: Tiles from T_{SAT} attach to the seed to nondeterministically select a truth assignment. Here $\phi = (x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee \neg x_1 \vee \neg x_0) \wedge (\neg x_2 \vee x_1 \vee x_0)$ and the literal selection is $x_0, \neg x_1, x_2$ meaning that $x_0 = x_2 = \mathit{TRUE}$ and $x_1 = \mathit{FALSE}$. Tiles attach to allow the ✓ tile to attach in the northwest corner iff the assignment satisfies the Boolean formula encoded by the seed.

problems, many of which are known to be NP-complete, require the encoding of vertices and edges, each of which must also be unique. Even though I have previously designed a tile system with a constant-sized tileset to solve an NP-complete problem [17], the smallest previously known tile solution to one of these special problems that require unique identifiers uses $\Theta(n^2)$ tiles [18].

The technique I present here for using a binary encoding to represent the necessary distinct identifiers can be adapted to be used with other problems. The adaptation is not immediately obvious and it is unlikely that a general adaptation exists that would work for all such problems. In the *SAT* system, it is sufficient to build a $\Theta(\log n) \times \Theta(\log n)$ block to compare two literals, and to "display" the result of the comparison in the north binding domain of the northwest tile. Other computational problems may require more complex computations than comparisons, and may require the result in more binding domains, thus larger tile blocks may be necessary for those other problems. The central idea of all such adaptations, however, remains the same: to represent unique identifies using a binary encoding, and to translate computation previously performed within a single tile into relatively small blocks of tiles.

3 Contributions

The main contribution of my work is a technique for reducing the size of the tileset used to solve certain computational problems. Tile system solutions to problems that require such unique identifiers, such as satisfiability and almost all graph problems, have resorted to using $\Theta(n)$ tiles to encode the input, and no fewer than $\Theta(n^2)$ tiles to compute, for inputs of size n [18]. My proposal allows the first constant-size tileset solution that solves such a problem. The mechanism I design for uniquely addressing variables is completely portable to solving other problems.

To illustrate my technique, I have designed a system that solves well-known NP-complete problems *k-SAT*, for all $k \in \mathbb{N}$, in the tile assembly model. The system, $\mathbb{S}_{SAT}$, uses $64 = \Theta(1)$ distinct tiles to decide whether a Boolean formula is satisfiable, computes in time linear in the input size, and each nondeterministic assembly has a probability of success of at least $\left(\frac{1}{2}\right)^n$, where n is the number of distinct variables. Thus a parallel implementation of $\mathbb{S}_{SAT}$, such as a DNA implementation like those in [11, 12], with 2^n seeds has at least a $1 - \frac{1}{e} \geq 0.5$ chance of correctly deciding whether a Boolean formula is satisfiable, and that probability can be brought arbitrarily close to 1 by increasing the number of seeds.

References

[1] Erik Winfree. Simulations of computing by self-assembly of DNA. Technical Report CS-TR:1998:22, California Institute of Technology, Pasadena, CA, USA, 1998.

[2] Leonard Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[3] Ravinderjit Braich, Cliff R. Johnson, Paul W. K. Rothemund, Darryl Hwang, Nickolas Chelyapov, and Leonard Adleman. Solution of a satisfiability problem on a gel-based DNA computer. In *Proceedings of DNA Computing: 6th International Workshop on DNA-Based Computers (DNA00)*, pages 27–38, Leiden, The Netherlands, June 2000.

[4] Ravinderjit Braich, Nickolas Chelyapov, Cliff R. Johnson, Paul W. K. Rothemund, and Leonard Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296(5567):499–502, 2002.

[5] Erik Winfree. On the computational power of DNA annealing and ligation. *DNA Based Computers*, pages 199–221, 1996.

[6] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS02)*, volume 2943, pages 126–144, Madison, WI, USA, June 2003.

[7] Yuliy Baryshnikov, Ed G. Coffman, Nadrian Seeman, and Teddy Yimwadsana. Self correcting self assembly: Growth models and the hammersley process. In *Proceedings of the 11th International Meeting on DNA Computing (DNA05)*, London, Ontario, June 2005.

[8] Ho-Lin Chen and Ashish Goel. Error free self-assembly with error prone tiles. In *Proceedings of the 10th International Meeting on DNA Based Computers (DNA04)*, Milan, Italy, June 2004.

[9] John H. Reif, Sadheer Sahu, and Peng Yin. Compact error-resilient computational DNA tiling assemblies. In *Proceedings of the 10th International Meeting on DNA Based Computers (DNA04)*, Milan, Italy, June 2004.

[10] Erik Winfree. Self-healing tile sets. *Nanotechnology: Science and Computation*, pages 55–78, 2006.

[11] Robert Barish, Paul W. K. Rothemund, and Erik Winfree. Two computational primitives for algorithmic self-assembly: Copying and counting. *Nano Letters*, 5(12):2586–2592, 2005.

[12] Paul W. K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):e424, 2004.

[13] Leonard Adleman. Towards a mathematical theory of self-assembly. Technical Report 00-722, Department of Computer Science, University of Southern California, Los Angeles, CA, 2000.

[14] Leonard Adleman, Qi Cheng, Ahish Goel, Ming-Deh Huang, and Hal Wasserman. Linear self-assemblies: Equilibria, entropy, and convergence rates. In *Proceedings of the 6th International Conference on Difference Equations and Applications (ICDEA01)*, Augsburg, Germany, June 2001.

[15] Yuriy Brun. Arithmetic computation in the tile assembly model: Addition and multiplication. *Theoretical Computer Science*, 378(1):17–31, June 2007.

[16] Yuriy Brun. Nondeterministic polynomial time factoring in the tile assembly model. *Theoretical Computer Science*, 395(1):3–23, 2008.

[17] Yuriy Brun. Solving NP-complete problems in the tile assembly model. *Theoretical Computer Science*, 395(1):31–46, 2008.

[18] Michail G. Lagoudakis and Thomas H. LaBean. 2D DNA self-assembly for satisfiability. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 54:141–154, 1999.

[19] Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC00)*, pages 459–468, Portland, OR, USA, May 2000.

[20] Hao Wang. Proving theorems by pattern recognition. *II. Bell System Technical Journal*, 40:1–42, 1961.

[21] Yuriy Brun. Solving satisfiability in the tile assembly model with a constant-size tileset. Technical Report USC-CSSE-2008-801, Center for Software Engineering, University of Southern California, 2008.