Constant-Size Tileset for Solving an NP-Complete Problem in Nondeterministic Linear Time

Yuriy Brun

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

Abstract. The tile assembly model, a formal model of crystal growth, is of special interest to computer scientists and mathematicians because it is universal [1]. Therefore, tile assembly model systems can compute all the functions that computers compute. In this paper, I formally define what it means for a system to nondeterministically decide a set, and present a system that solves an NP-complete problem called SubsetSum. Because of the nature of NP-complete problems, this system can be used to solve all NP problems in polynomial time, with high probability. While the proof that the tile assembly model is universal [2] implies the construction of such systems, those systems are in some sense "large" and "slow." The system presented here uses $49 = \Theta(1)$ different tiles and computes in time linear in the input size. I also propose how such systems can be leveraged to program large distributed software systems.

1 Introduction

Self-assembly is a process that is ubiquitous in nature. Systems form on all scales via self-assembly: atoms self-assemble to form molecules, molecules to form complexes, and stars and planets to form galaxies. One manifestation of self-assembly is crystal growth: molecules self-assembling to form crystals. Crystal growth is an interesting area of research for computer scientists because it has been shown that, in theory, under careful control, crystals can compute [2]. The field of DNA computation demonstrated that DNA can be used to compute [3], solving NP-complete problems such as the satisfiability problem [4,5]. This idea of using molecules to compute nondeterministically is the driving motivation behind my work.

Winfree showed that DNA computation is Turing-universal [6]. 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 [7,8,9,10,11]. Researchers have used DNA to assemble crystals with patterns of binary counters [12] and Sierpinski triangles [13], 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 this system?" For nondeterministic computation, the following question is also important: "what is the probability of assembling the crystal that encodes the solution?" Researchers have answered these questions for n-long linear polymers [14] and $n \times n$ squares (minimum tileset of size $\Theta(\frac{\log n}{\log \log n})$ and optimal assembly time of $\Theta(n)$) [15,16,17]. A key issue related to assembling squares is the assembly of small binary counters, which theoretically can have as few as 7 tile types [18].

Other early attempts at nondeterministic computation include a proposal by Lagoudakis et al. to solve the satisfiability problem [19]. They informally define a system that nondeterministically computes whether or not an n-variable boolean formula is satisfiable using $\Theta(n^2)$ distinct tiles. In contrast, all the systems I present in this paper use $\Theta(1)$ distinct tiles.

Barish et al. have demonstrated a DNA implementation of tile systems, one that copies an input and another that counts in binary [12]. Similarly, Rothemund et al. have demonstrated a DNA implementation of a tile system that computes the *xor* function, resulting in a Sierpinski triangle [13]. These systems grow crystals using double-crossover complexes [20] as tiles. The theoretical underpinnings of these systems are closely related to the work presented here because these systems compute functions.

1.1 Tile Assembly Model

The tile assembly model [15,1,2] 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 [21]. The model was fully defined by Rothemund and Winfree [15], and the definitions I use are similar to those. Full formal definitions can be found in [22].

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 [22] and [23], I give formal definitions of what it means for a tile system to compute functions, both deterministically and nondeterministically. Here, I am interested in computing 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. Since for all constants $m \in \mathbb{N}$, the cardinalities of $\mathbb{N}^m$ and $\mathbb{N}$ are the same, it makes sense to talk about deciding subsets of $\mathbb{N}^m$.

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 $\boldsymbol{a} \in \mathbb{N}^m$, there exists a seed configuration S that encodes $\boldsymbol{a}$ and for all final configurations F that $\mathbb{S}$ produces on S, $r \in F(\mathbb{Z}^2)$ iff $\boldsymbol{a} \in \Omega$, 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.

This paper provides the definitions necessary for understanding the below constructions and theorems. More complete versions of the definitions and formal proofs of the theorems presented below can be found in [24]. In the remainder of this paper, I require systems to encode their inputs in binary, and call the set of tiles used to encode the input Γ.

2 Solving SubsetSum

SubsetSum is a well known NP-complete problem. The set *SubsetSum* is a set of pairs: a finite sequence $\boldsymbol{B} = \langle B_1, B_2, \cdots, B_n \rangle \in \mathbb{N}^n$, and a target number $v \in \mathbb{N}$, such that $\langle \boldsymbol{B}, v \rangle \in \mathit{SubsetSum}$ iff $\exists \boldsymbol{c} = \langle c_1, c_2, \cdots, c_n \rangle \in \{0,1\}^n$ such that $\sum_{i=1}^{n} c_i B_i = v$. In other words, the sum of some subset of numbers of $\boldsymbol{B}$ equals exactly v.

In order to explain the system that nondeterministically decides *SubsetSum*, I will first define three smaller systems that perform pieces of the necessary computation. The first system subtracts numbers, and given the right conditions, will subtract a B_i from v. The second system computes the identity function and just copies information (this system will be used when a B_i should not be subtracted from v). The third system nondeterministically guesses whether the next B_i should or should not be subtracted. Finally, I will add a few other tiles that ensure that the computations went as planned and attach an identifier tile if the execution found that $\langle \boldsymbol{B}, v \rangle \in \mathit{SubsetSum}$. The system works by nondeterministically choosing a subset of $\boldsymbol{B}$ to subtract from v and comparing the result to 0.

2.1 Subtraction

In this section, I will describe a system that subtracts positive integers. It is similar to one of the addition systems from [22], contains 16 tiles, and will subtract one bit per row of computation.

Figure 1(a) shows the 16 tiles of T_{-}. The value in the middle of each tile t represents that tile's $v(t)$ value. Intuitively, the system will subtract the i^{th} bit on the i^{th} row. The tiles to the right of the i^{th} location will be blue; the tile in the i^{th} location will be yellow; the next tile, the one in the $(i+1)^{\text{st}}$ location, will be magenta; and the rest of the tiles will be green. The purpose of the yellow and magenta tiles is to compute the diagonal line, marking the i^{th} position on the i^{th} row.

$\mathbb{S}_{-}$ is a system that is capable of subtracting numbers, and it does so in time linear in the input. Full proofs of these statements are available in [24]. In Figure 1(b), the system computes $221 - 214 = 7$ In Figure 1(c), the system attempts to compute $221 - 246$, but because $246 > 221$, the computation fails.

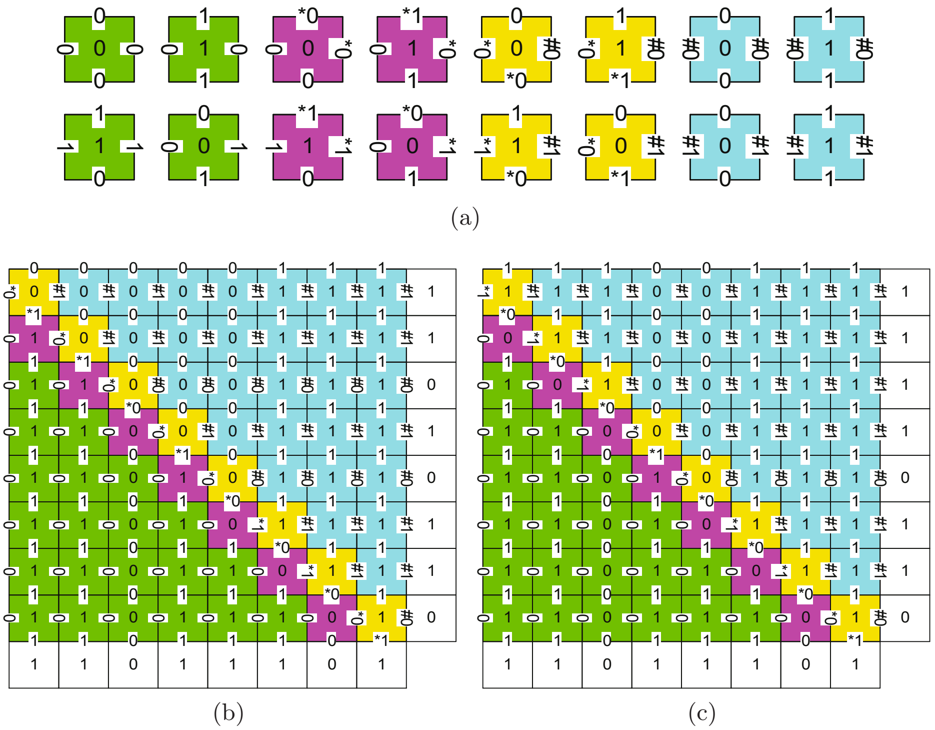

Fig. 1. There are 16 tiles in T_{-} (a). The value in the middle of each tile t represents that tile's $v(t)$ value. In (b), the system subtracts $214 = 11010110_2$ from $221 = 11011101_2$ to get $7 = 111_2$. The inputs are encoded along the bottom row ($221 = 11011101_2$) and right-most column ($214 = 11010110_2$). The output is on the top row ($7 = 00000111_2$). Because $214 \leq 221$, all the west binding domains of the left-most column contain a 0. In (c), the system attempts to subtract $246 = 11110110$ from $221 = 11011101_2$, but because $246 > 221$, the computation fails and indicates its failure with the top- and left-most west binding domain containing a 1.

This system is very similar to an adding system from [22], but not the smallest adding system from [22]. While this system has 16 tiles, it is possible to design a subtracting system with 8 tiles, that is similar to the 8-tile adding system from [22].

2.2 Identity

I now describe a system that ignores the input on the right-most column, and simply copies upwards the input from the bottom row. This is a fairly straightforward system that will not need much explanation. Figure 2(a) shows the 4 tiles in T_{x} and Figure 2(b) shows a sample execution of the $\mathbb{S}_{\text{x}}$ system.

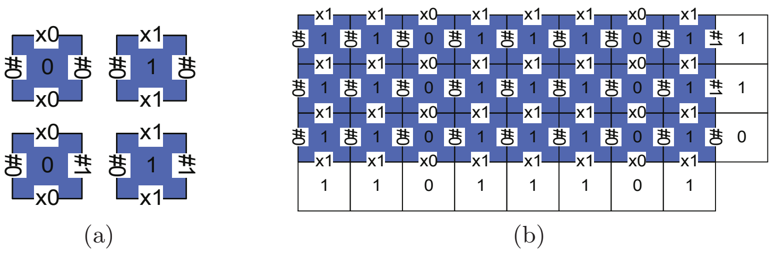

Fig. 2. There are 4 tiles in T_{x} (a); the value in the middle of each tile t represents that tile's $v(t)$ value. In an example of an $\mathbb{S}_{\text{x}}$ execution (b), the system simply copies the input on the bottom row upwards, to the top column.

$\mathbb{S}_{\text{x}}$ is a system that is capable of computing the identify function, and it does so in time linear in the input. Again, full proofs of these statements are available in [24].

2.3 Nondeterministic Guess

In this section, I describe a system that nondeterministically decides whether or not the next B_i should be subtracted from v. It does so by encoding the input for either the $\mathbb{S}_-$ system or the $\mathbb{S}_{\text{x}}$ system.

$\mathbb{S}_?$ is a system that is capable of nondeterministically preparing a valid seed configuration for either $\mathbb{S}_-$ or $\mathbb{S}_{\text{x}}$, and it does so in time linear in the input. Full proofs of these statements are available in [24].

Figure 3 shows two possible executions of $\mathbb{S}_?$. In Figure 3(b), the system attaches tiles with ! east-west binding domains, preparing a valid seed for $\mathbb{S}_-$, and in Figure 3(c), the system attaches tiles with x east-west binding domains, preparing a valid seed for $\mathbb{S}_{\text{x}}$. Only one tile, the orange tile, attaches nondeterministically, determining which tiles attach to its west.

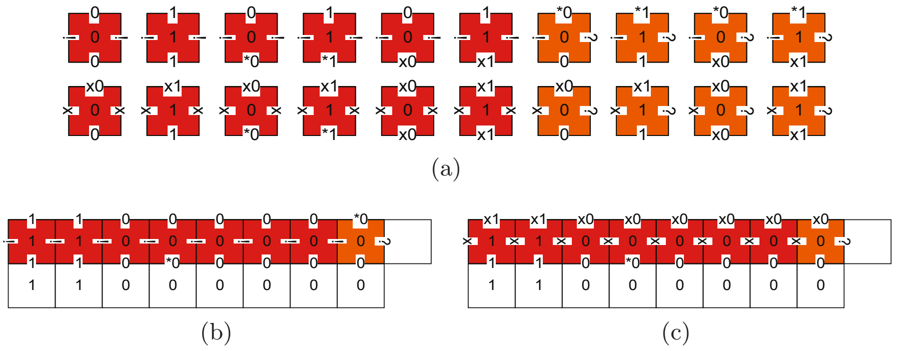

Fig. 3. There are 20 tiles in $T_?$ (a). The value in the middle of each tile t represents that tile's $v(t)$ value. Unlike the red tiles, the orange tiles do not have unique east-south binding domain pairs, and thus will attach nondeterministically. In (b), the system attaches tiles with ! east-west binding domains, preparing a valid seed for $\mathbb{S}_{-}$, and in (c), the system attaches tiles with x east-west binding domains, preparing a valid seed for $\mathbb{S}_{\mathrm{x}}$.

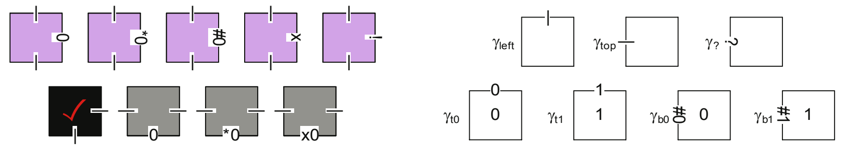

Fig. 4. There are 9 tiles in $T_{\checkmark}$ (a); the black tile with a ✓ in the middle will serve as the identifier tile. There are 7 tiles in Γ_{SS} (b); the value in the middle of each tile t represents that tile's $v(t)$ value and each tile's name is written on its left.

2.4 Deciding *SubsetSum*

I have described three systems that I will now use to design a system to decide *SubsetSum*. Intuitively, I plan to write out the elements of $\boldsymbol{B}$ on a column and v on a row, and the system will nondeterministically choose some of the elements from $\boldsymbol{B}$ to subtract from v. The system will then check to make sure that no subtracted element was larger than the number it was being subtracted from, and whether the result is 0. If the result is 0, then a special identifier tile will attach to signify that $\langle \boldsymbol{B}, v\rangle \in SubsetSum$.

Theorem 1. *Let* $\Sigma_{SS} = \{0, 1, {}^\star0, {}^\star1, \#0, \#1, \mathrm{x}0, \mathrm{x}1, \#0, \#1, ?, !, 0, 1, \mathrm{x}0, \mathrm{x}1, {}^\star0, {}^\star1\}$. *Let* $T_{SS} = T_{-} \cup T_{\mathrm{x}} \cup T_? \cup T_{\checkmark}$, *where* $T_{\checkmark}$ *is defined by Figure 4(a). Let* $g_{SS} = 1$ *and* $\tau_{SS} = 2$. *Let* $\mathbb{S}_{SS} = \langle T_{SS}, g_{SS}, \tau_{SS}\rangle$. *Then* $\mathbb{S}_{SS}$ *nondeterministically decides SubsetSum with the black ✓ tile from* $T_{\checkmark}$ *as the identifier tile.*

I refer the reader to [24] for a full proof of theorem 1.

Figure 5 shows an example execution of $\mathbb{S}_{SS}$. Figure 5(a) encodes a seed configuration with $v = 75 = 1001011_2$ along the bottom row and $\boldsymbol{B} = \langle 11 = 1011_2$,

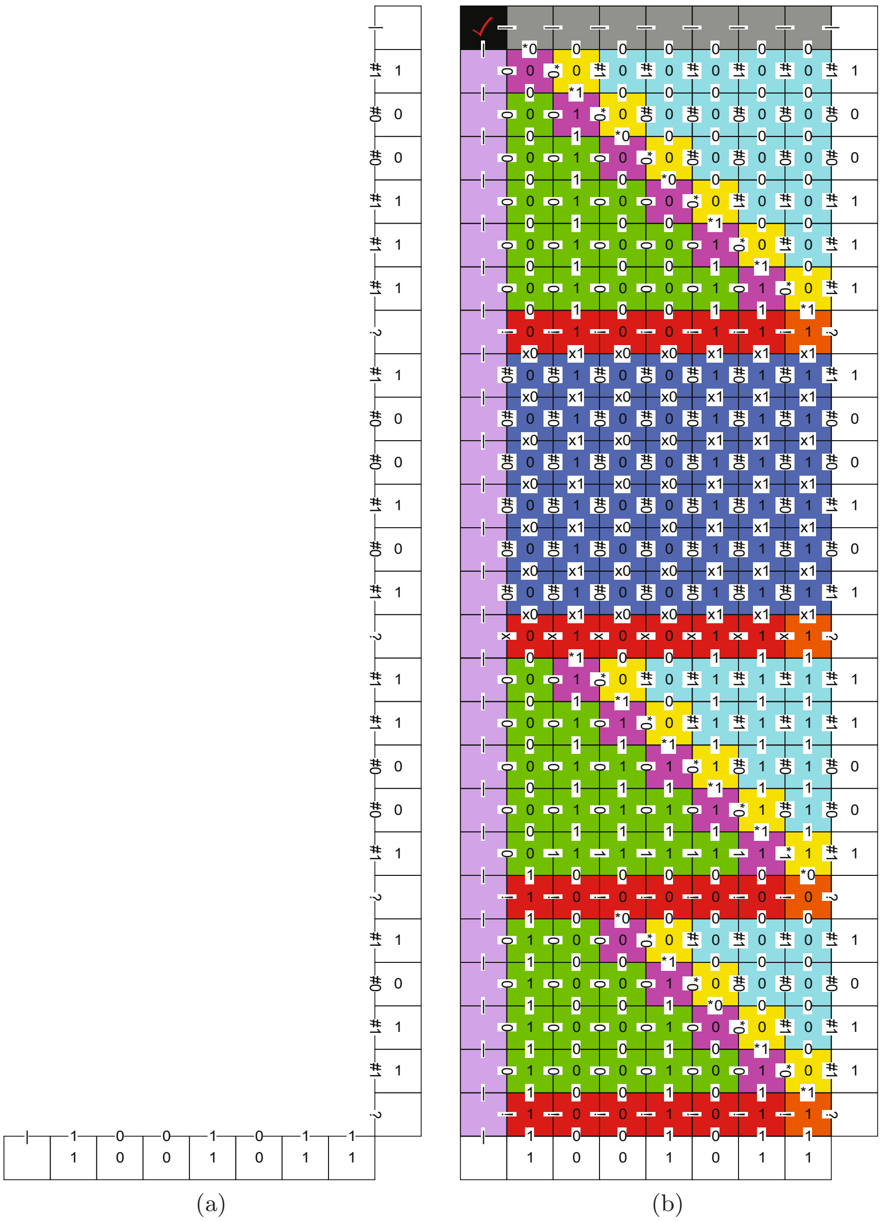

Fig. 5. An example of $\mathbb{S}_{SS}$ solving a *SubsetSum* problem. Here, $v = 75 = 1001011_2$, and $\boldsymbol{B} = \langle 11 = 1011_2,\ 25 = 11001_2,\ 37 = 100101_2,\ 39 = 100111_2\rangle$. The seed configuration encodes v on the bottom row and $\boldsymbol{B}$ on the right-most column (a). The fact that $75 = 11+25+39$ implies that $\langle \boldsymbol{B}, t\rangle \in$ *SubsetSum*, thus at least one final configuration (b) contains the ✓ tile.

$25 = 11001_2$, $37 = 100101_2$, $39 = 100111_2\rangle$ along the right-most column. Note that the seed is encoded using the tiles shown in Figure 4(b). Tiles from T_{SS} attach to the seed configuration, nondeterministically testing all possible values of $\boldsymbol{c} \in \{0,1\}^4$. Figure 5(b) shows one such possible execution, the one that corresponds to $\boldsymbol{c} = \langle 1,1,0,1\rangle$. Because $11 + 25 + 39 = 75$, the ✓ tile attaches in the top left corner.

The assembly time of $\mathbb{S}_{SS}$ is linear in the size of the input (number of bits in $\langle \boldsymbol{B}, v\rangle$), and assuming each tile that may attach to a configuration at a certain position attaches there with a uniform probability distribution, the probability that a single nondeterministic execution of $\mathbb{S}_{SS}$ succeeds in attaching a ✓ tile if $\langle \boldsymbol{B}, v\rangle \in \mathit{SubsetSum}$ is at least $\left(\frac{1}{2}\right)^n$. The proofs of both these statements can be found in [24].

Therefore, a parallel implementation of $\mathbb{S}_{SS}$, such as a DNA implementation like those in [12,13], with 2^n seeds has at least a $1 - \frac{1}{e} \geq 0.5$ chance of correctly deciding whether a $\langle \boldsymbol{B}, v\rangle \in \mathit{SubsetSum}$. An implementation with 100 times as many seeds has at least a $1 - \left(\frac{1}{e}\right)^{100}$ chance.

Note that T_{SS} has 49 computational tile types and uses 7 tile types to encode the input.

3 Software Systems

Fault and adversary tolerance have become not only desirable but required properties of software systems because mission-critical systems are commonly distributed on large networks of insecure nodes. Further, users of such distributed systems may desire their private data to remain private. It is possible for computers on a large network to act as tiles to compute. For example, one can solve NP-complete problems by reducing them to $\mathit{SubsetSum}$ and then using $\mathbb{S}_{SS}$ to solve them, as illustrated in Figure 6. Such a software system can leverage the error-correction work in tile assembly [7,8,9,10,11] to automate fault and adversary tolerance, and distribute computation over network in a way that no small group of nodes nodes the private inputs to the computation [25,26,27].

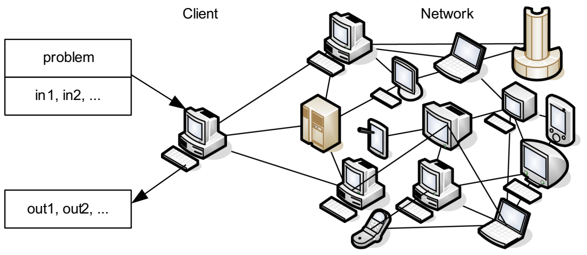

Fig. 6. A schematic of a system implementing a tile-style architecture

4 Contributions

The tile assembly model is a formal model of self-assembly and crystal growth. Here, I defined what it means for a tile system to decide a set and designed and explored a system that decides an NP-complete problem *SubsetSum*. The system computes at temperature two and uses 49 computational tile types and 7 tile types to encode the input. The system computes in time linear in the input size and each nondeterministic assembly has a probability of success of at least $\left(\frac{1}{2}\right)^n$, and that probability can be brought exponentially close to 1 at a linear cost in the number of seeds.

On the way to defining a system that decides *SubsetSum*, I also defined a system that deterministically subtracts numbers. This system uses 16 computational tile types and executes in time linear in the input size. I stated without proof that there exists an 8-tile subtracting system based on the 8-tile adding system from [22].

Finally, I described some preliminary work on using theoretical self-assembly to design complex computational software systems.

References

1. Winfree, E.: Simulations of computing by self-assembly of DNA. Technical Report CS-TR:1998:22, California Insitute of Technology, Pasadena, CA, USA (1998)
2. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Insitute of Technology, Pasadena, CA, USA (June 1998)
3. Adleman, L.: Molecular computation of solutions to combinatorial problems. Science 266, 1021–1024 (1994)
4. Braich, R., Johnson, C.R., Rothemund, P.W.K., Hwang, D., Chelyapov, N., Adleman, L.: Solution of a satisfiability problem on a gel-based DNA computer. In: Proceedings of DNA Computing: 6th International Workshop on DNA-Based Computers (DNA 2000), Leiden, The Netherlands, pp. 27–38 (June 2000)
5. Braich, R., Chelyapov, N., Johnson, C.R., Rothemund, P.W.K., Adleman, L.: Solution of a 20-variable 3-SAT problem on a DNA computer. Science 296(5567), 499–502 (2002)
6. Winfree, E.: On the computational power of DNA annealing and ligation. DNA Based Computers 199–221 (1996)
7. Winfree, E., Bekbolatov, R.: Proofreading tile sets: Error correction for algorithmic self-assembly. In: Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2002), Madison, WI, USA, June 2003, vol. 2943, pp. 126–144. IEEE Computer Society Press, Los Alamitos (2003)
8. Baryshnikov, Y., Coffman, E.G., Seeman, N., Yimwadsana, T.: Self correcting self assembly: Growth models and the hammersley process. In: Carbone, A., Pierce, N.A. (eds.) DNA Computing. LNCS, vol. 3892, Springer, Heidelberg (2006)
9. Chen, H.L., Goel, A.: Error free self-assembly with error prone tiles. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing. LNCS, vol. 3384, Springer, Heidelberg (2005)
10. Reif, J.H., Sahu, S., Yin, P.: Compact error-resilient computational DNA tiling assemblies. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA Computing. LNCS, vol. 3384, Springer, Heidelberg (2005)

11. Winfree, E.: Self-healing tile sets. Nanotechnology: Science and Computation, 55–78 (2006)
12. Barish, R., Rothemund, P.W.K., Winfree, E.: Two computational primitives for algorithmic self-assembly: Copying and counting. Nano Letters 5(12), 2586–2592 (2005)
13. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology 2(12), 424 (2004)
14. Adleman, L., Cheng, Q., Goel, A., Huang, M.-D., Wasserman, H.: Linear self-assemblies: Equilibria, entropy, and convergence rates. In: Proceedings of the 6th International Conference on Difference Equations and Applications (ICDEA 2001), Augsburg, Germany (June 2001)
15. Rothemund, P.W.K., Winfree, E.: The program-size complexity of self-assembled squares. In: Proceedings of the ACM Symposium on Theory of Computing (STOC 2000, Portland, OR, USA, pp. 459–468. ACM Press, New York (2000)
16. Adleman, L., Cheng, Q., Goel, A., Huang, M.-D., Kempe, D., de Espanes, P.M., Rothemund, P.W.K.: Combinatorial optimization problems in self-assembly. In: Proceedings of the ACM Symposium on Theory of Computing (STOC 2002), Montreal, Quebec, Canada, pp. 23–32. ACM Press, New York (2002)
17. Adleman, L., Goel, A., Huang, M.-D., de Espanes, P.M.: Running time and program size for self-assembled squares. In: Proceedings of the ACM Symposium on Theory of Computing (STOC 2002), Montreal, Quebec, Canada, pp. 740–748. ACM Press, New York (2001)
18. de Espanes, P.M.: Computerized exhaustive search for optimal self-assembly counters. In: Proceedings of the 2nd Foundations of Nanoscience: Self-Assembled Architectures and Devices (FNANO 2005), Snowbird, UT, USA, pp. 24–25 (April 2005)
19. Lagoudakis, M.G., LaBean, T.H.: 2D DNA self-assembly for satisfiability. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 54, 141–154 (1999)
20. Fu, T.J., Seeman, N.C.: DNA double-crossover molecules. Biochemistry 32(13), 3211–3220 (1993)
21. Wang, H.: Proving theorems by pattern recognition. II. Bell System Technical Journal 40, 1–42 (1961)
22. Brun, Y.: Arithmetic computation in the tile assembly model: Addition and multiplication. Theoretical Computer Science 378, 17–31 (2007)
23. Brun, Y.: Nondeterministic polynomial time factoring in the tile assembly model. Theoretical Computer Science (2007), doi:10.1016/j.tcs.2007.07.051
24. Brun, Y.: Solving NP-complete problems in the tile assembly model. Theoretical Computer Science (2007), doi:10.1016/j.tcs.2007.07.052
25. Brun, Y., Medvidovic, N.: An architectural style for solving computationally intensive problems on large networks. In: Proceedings of Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2007), Minneapolis, MN, USA (May 2007)
26. Brun, Y., Medvidovic, N.: Fault and adversary tolerance as an emergent property of distributed systems' software architectures. In: Proceedings of the 2nd International Workshop on Engineering Fault Tolerant Systems (EFTS 2007), Dubrovnik, Croatia (September 2007)
27. Brun, Y.: Discreetly distributing computation via self-assembly. Technical Report USC-CSSE-2007-714, Center for Software Engineering, University of Southern California (2007)