Synthesizing Small and Reliable Tile Sets for Patterned DNA Self-Assembly

Tuomo Lempiäinen, Eugen Czeizler, and Pekka Orponen

Department of Information and Computer Science
Aalto University School of Science
P.O. Box 15400, FI-00076 Aalto, Finland
firstname.lastname@aalto.fi

Abstract. We consider the problem of finding, for a given 2D pattern of coloured tiles, a minimal set of tile types self-assembling to this pattern in the abstract Tile Assembly Model of Winfree (1998). This Patterned self-Assembly Tile set Synthesis (PATS) problem was first introduced by Ma and Lombardi (2008), and subsequently studied by Göös and Orponen (2011), who presented an exhaustive partition-search branch-and-bound algorithm (briefly PS-BB) for it. However, finding the true minimal tile-sets is very time consuming, and the algorithm PS-BB is not well-suited for finding small but not necessarily minimal solutions.
In this paper, we modify the basic partition-search framework by using a heuristic to optimise the order in which the algorithm traverses its search space. We find that by running several parallel instances of the modified algorithm PS-H, the search time for small tile sets can be shortened considerably. Additionally, we suggest a new approach, answer set programmin (ASP), to the PATS problem. We also introduce a method for computing the reliability of a given tile set, i.e. the probability of its error-free self-assembly to the desired target tiling, based on Winfree's analysis of the kinetic Tile Assembly Model (1998). We present empirical data on the reliability of tile sets found by the PS-BB and PS-H algorithms and find that also here the PS-H algorithm constitutes a significant improvement over the PS-BB method.

1 Introduction

Self-assembly of nanostructures templated on synthetic DNA has been proposed by several authors as a potentially ground-breaking technology for the manufacture of next-generation circuits, devices, and materials [2, 8, 15, 16, 23, 24]. Also laboratory techniques for synthesizing the requisite 2D DNA template lattices, many based on Rothemund's [18] DNA origami tiles, have recently been demonstrated by many groups [4, 10, 17, 25].

In order to support the manufacture of aperiodic structures, such as electronic circuit designs, these DNA templates need to be addressable. When the template is construed as a tiling from a family of DNA origami (or other kinds of) tiles, one can view the base tiles as being "coloured" according to their different functionalities, and the completed template implementing a desired colour pattern. Now a given target pattern can be assembled from many different families of base tiles, and it is clearly advantageous to try to minimise the number of tile types needed and/or maximise the probability that they self-assemble to the desired pattern, given some model of tiling errors.

The task of minimising the number of DNA tile types required to implement a given 2D pattern was identified by Ma and Lombardi [13], who formulated it as a combinatorial optimisation problem, the **Patterned self-Assembly Tile set Synthesis** (PATS) problem. Ma and Lombardi proposed two greedy heuristics for solving the task, and subsequently Göös and Orponen [7] presented an exhaustive partition-search branch-and-bound algorithm for it.

While the search algorithm of Göös and Orponen [7], which we denote here as PS-BB, is somewhat successful in finding minimal tile sets for small patterns, the size of the search space grows so rapidly that it seems to hit a complexity barrier at approximately pattern sizes of 7×7 tiles. In practice one would of course not need to find an absolutely minimal tile set for a given pattern, but any reasonably small solution set would suffice. However, when the algorithm PS-BB fails to find a minimal solution, it does not seem to yield very good approximate solutions either.

In the present work, we approach the task of finding small but not necessarily minimal tile sets for a given 2D pattern by tailoring the basic partition-search framework of Göös and Orponen [7] towards this goal. Instead of a systematic branch-and-bound pruning and traversal of the complete search space, we apply a heuristic that attempts to optimise the order of the directions in which the space is explored. The new algorithm, denoted PS-H, is described in more detail below in Section 3.1.

It is well known in the heuristic optimisation community [6, 11] that when the runtime distribution of a randomised search algorithm has a large variance, it is with high probability more efficient to run several independent short runs ("restarts") of the algorithm than a single long run. Correspondingly, we investigate the efficiency of the PS-H search method for a number of parallel executions ranging from 1 to 32, and note that indeed this number has a significant effect on the success rate of the algorithm in finding small tile sets. These results are discussed below in Section 3.3.

In addition to the partition search algorithms, we explore the PATS problem using the artificial intelligence technique of answer set programming (ASP) [9], which has proved highly successful in other hard combinatorial search problems. In Section 3.2 we outline the procedure for transforming the PATS problem, as well as the given target pattern, to an ASP logic program. We present results on the performance of this approach in Section 3.3 and find that, for patterns with a known small minimal solution, the ASP approach indeed works very well in discovering that solution.

Given the inherently stochastic nature of the DNA self-assembly process, it is also of interest to assess the reliability of a given tile set, i.e. the probability of its error-free self-assembly to the desired target tiling. In Section 4.2 we introduce a method for estimating this quantity, based on Winfree's analysis of the kinetic Tile Assembly Model [22]. In Section 4.3 we present empirical data on the reliability of tile sets found by the PS-BB and PS-H algorithms and find that also here the PS-H algorithm with parallel runs constitutes a significant improvement over the PS-BB method.

2 The PATS Problem and the PS-BB Algorithm

In this Section, we first briefly review the abstract Tile Assembly Model (aTAM) from [19, 22], then summarise the PATS problem from [13], and finally outline the PS-BB algorithm from [7].

2.1 The Abstract Tile Assembly Model

Let $\mathcal{D} = \{N, E, S, W\}$ be the set of four functions $\mathbb{Z}^2 \to \mathbb{Z}^2$ corresponding to the cardinal directions (north, east, south, west) so that $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S = N^{-1}$ and $W = E^{-1}$. Let Σ be a set of **glue types** and $s : \Sigma \times \Sigma \to \mathbb{N}$ a **glue strength** function such that $s(\sigma_1, \sigma_2) = 0$ if $\sigma_1 \neq \sigma_2$. A **tile type** $t \in \Sigma^4$ is a quadruple $(\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t))$ and a **(tile) assembly** $\mathcal{A}$ is a partial mapping from $\mathbb{Z}^2$ to Σ^4. A **tile assembly system** (TAS) $\mathscr{T} = (T, \mathcal{S}, s, \tau)$ consists of a finite set T of tile types, an assembly $\mathcal{S}$ called the **seed assembly**, a glue strength function s and a **temperature** $\tau \in \mathbb{Z}^+$ (we use $\tau = 2$).

Now consider a TAS $\mathscr{T} = (T, \mathcal{S}, s, \tau)$. Assembly $\mathcal{A}$ **produces directly** assembly $\mathcal{A}'$, denoted $\mathcal{A} \to_{\mathscr{T}} \mathcal{A}'$ if there exists a site $(x, y) \in \mathbb{Z}^2$ and a tile $t \in T$ such that $\mathcal{A}' = \mathcal{A} \cup \{((x, y), t)\}$, where the union is disjoint, and

$$\sum_D s(\sigma_D(t), \sigma_{D^{-1}}(\mathcal{A}(D(x, y)))\ \geq\ \tau\ ,$$

where D ranges over those directions in $\mathcal{D}$ for which $\mathcal{A}(D(x, y))$ is defined. That is, a new tile can be adjoined to an assembly $\mathcal{A}$ if the new tile shares a common boundary with tiles that bind it into place with total strength at least τ.

Let $\to^*_{\mathscr{T}}$ be the reflexive transitive closure of $\to_{\mathscr{T}}$. A TAS $\mathscr{T}$ **produces** an assembly $\mathcal{A}$ if $\mathcal{S} \to^*_{\mathscr{T}} \mathcal{A}$. Denote by $\operatorname{Prod} \mathscr{T}$ the set of all assemblies produced by $\mathscr{T}$. A TAS $\mathscr{T}$ is **deterministic** if for any assembly $\mathcal{A} \in \operatorname{Prod} \mathscr{T}$ and for every $(x, y) \in \mathbb{Z}^2$ there exists at most one $t \in T$ such that $\mathcal{A}$ can be extended with t at site (x, y). Then the pair $(\operatorname{Prod} \mathscr{T}, \to^*_{\mathscr{T}})$ forms a partially ordered set, which is a lattice if and only if $\mathscr{T}$ is deterministic. The maximal elements in $\operatorname{Prod} \mathscr{T}$, i.e. the assemblies $\mathcal{A}$ for which there do not exist any $\mathcal{A}'$

satisfying $\mathcal{A} \to_{\mathscr{T}} \mathcal{A}'$, are called **terminal assemblies**. Denote by $\operatorname{Term} \mathscr{T}$ the set of terminal assemblies of $\mathscr{T}$. If all **assembly sequences**

$$\mathcal{S} \to_{\mathscr{T}} \mathcal{A}_1 \to_{\mathscr{T}} \mathcal{A}_2 \to_{\mathscr{T}} \cdots$$

terminate and $\operatorname{Term} \mathscr{T} = \{\mathcal{P}\}$ for some assembly $\mathcal{P}$, we say that $\mathscr{T}$ **uniquely produces** $\mathcal{P}$.

2.2 The PATS Problem

Let the dimensions m and n be fixed. A mapping from $[m] \times [n] \subseteq \mathbb{Z}^2$ onto $[k]$ defines a **k-colouring** or a **k-coloured pattern**. To build a given pattern, we start with boundary tiles in place for the west and south borders of the m by n rectangle and keep extending this assembly by tiles with strength-1 glues.

Definition 1 (Pattern self-Assembly Tile set Synthesis (PATS) [13]**).**

Given: *A k-colouring $c : [m] \times [n] \to [k]$.*
Find: *A tile assembly system $\mathscr{T} = (T, \mathcal{S}, s, 2)$ such that*

P1. The tiles in T have bonding strength 1.
P2. The domain of $\mathcal{S}$ is $[0, m] \times \{0\} \cup \{0\} \times [0, n]$ and all the terminal assemblies have the domain $[0, m] \times [0, n]$.
P3. There exists a colouring $d : T \to [k]$ such that for each terminal assembly $\mathcal{A} \in \operatorname{Term} \mathscr{T}$ we have $d(\mathcal{A}(x, y)) = c(x, y)$ for all $(x, y) \in [m] \times [n]$.

The problem of finding minimal solutions (in terms of $|T|$) to the PATS problem has been shown to be NP-hard in [14]. Without loss of generality, we can consider only such TASs $\mathscr{T}$ that have tile sets in which every tile participates in assembling some terminal assembly of $\mathscr{T}$.

In the literature, the seed assembly of a TAS is often taken to be a single seed tile [1, 19] whereas we consider an L-shaped seed assembly. The boundaries can always be self-assembled using $m + n + 1$ different tiles with strength-2 glues, but we wish to make a clear distinction between the complexity of constructing the boundaries and the complexity of the 2D pattern itself.

2.3 The PS-BB Algorithm

The partition-search branch-and-bound (PS-BB) algorithm for the PATS problem proposed in [7], and based partly on ideas from [13], performs an exhaustive search in the lattice of partitions of the ambient rectangle $[m] \times [n]$. For each candidate partition P, the algorithm executes a polynomial-time test (details omitted in the present summary) to see if it is **constructible**, i.e. whether it can be generated by some deterministic tile system $\mathscr{T}$. If so, then the algorithm proceeds to consider coarsenings of P — these correspond to smaller tile systems — if not, then the algorithm backtracks. The search starts with the trivial partition which places each of the $m \cdot n$ sites in different classes, corresponding to an initial tile set that contains a distinct tile type for each of the tile sites in $[m] \times [n]$.

Let us now review some of the basic notions of the PS-BB algorithm in more detail. In the following, a PATS instance is assumed to be given by a fixed k-coloured pattern $c : [m] \times [n] \to [k]$.

The search space. Let X be the set of partitions of the set $[m] \times [n]$. Partition P is **coarser** than partition P' (or P' is a **refinement** of P), denoted $P \sqsubseteq P'$, if

$$\forall p' \in P' : \ \exists p \in P : \ p' \subseteq p \ .$$

Now, $(X, \sqsubseteq)$ is a partially ordered set, and in fact, a lattice. Note that $P \sqsubseteq P'$ implies $|P| \leq |P'|$.

The colouring c induces a partition $P(c) = \{c^{-1}(\{i\}) \mid i \in [k]\}$ of the set $[m] \times [n]$. In addition, since every (deterministic) solution $\mathscr{T} = (T, \mathcal{S}, s, 2)$ of the PATS problem uniquely produces some assembly $\mathcal{A}$, we associate with $\mathscr{T}$ a partition $P(\mathscr{T}) = \{\mathcal{A}^{-1}(\{t\}) \mid t \in \mathcal{A}([m] \times [n])\}$. Here, $|P(\mathscr{T})| = |T|$ in case all tiles in

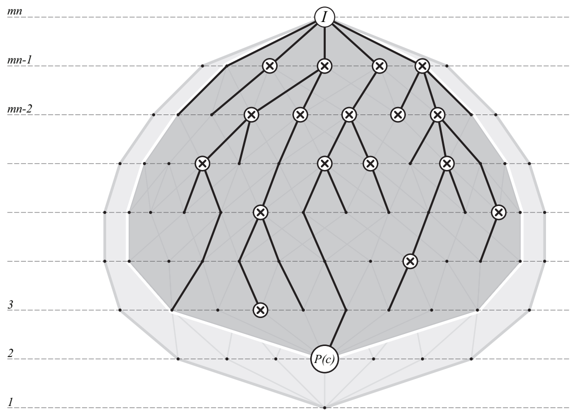


Fig. 1. The search lattice $(X, \sqsubseteq)$. The search starts with the initial partition I of size $|I| = mn$ at (top). The search considers the constructible partitions (crosses) in the upper sublattice of refinements of partition $P(c)$ (bottom).

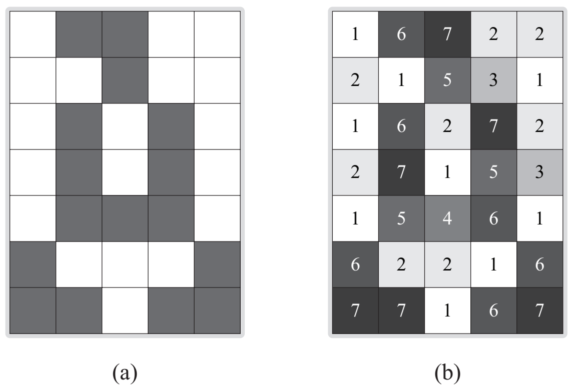


Fig. 2. (a) Partition A. (b) A partition M that is a refinement of A with $|M| = 7$ parts.

T are used in the terminal assembly. With this terminology, the condition *P3* in the definition of the PATS problem is equivalent to requiring that a TAS $\mathscr{T}$ satisfies

$$P(c) \sqsubseteq P(\mathscr{T}) \ .$$

A partition $P \in X$ is **constructible** if $P = P(\mathscr{T})$ for some deterministic TAS $\mathscr{T}$ with properties *P1* and *P2*. Now the PATS problem can be rephrased using partitions as the fundamental search space.

Proposition 1. *A minimal solution to the PATS problem corresponds to a partition $P \in X$ such that P is constructible, $P(c) \sqsubseteq P$ and $|P|$ is minimal.*

Schematically, the PS-BB algorithm performs an exhaustive top-to-bottom search in the lattice $(X, \sqsubseteq)$ as illustrated in Figure 1. The algorithm also involves several bounding heuristics for pruning the branches of the search, but discussion of these is omitted here for lack of space (see [7]).

For example, the 2-coloured pattern in Figure 2(a) defines a 2-part partition A. The 7-part partition M in Figure 2(b) is a refinement of A ($A \sqsubseteq M$) and in fact, M is constructible (see Figure 3(b)) and corresponds to a minimal solution of the PATS problem defined by the pattern A.

Determining constructibility. For lack of space, we shall omit the polynomial-time algorithm for testing whether a given partition P is constructible (see [7]), except for the mention of the following key notion.

Definition 2. *Given a partition P of the set $[m] \times [n]$, a* ***most general tile assignment*** *(MGTA) is a function ("tile map") $f : P \to \Sigma^4$ such that*

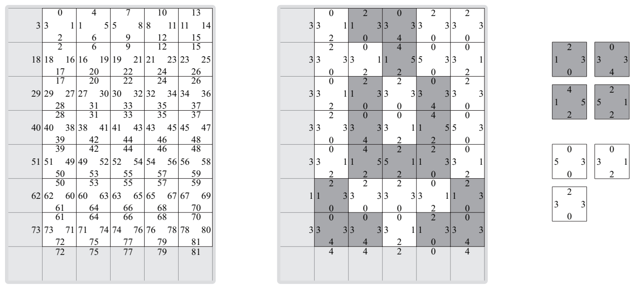
A1. f is consistent: when sites in $[m] \times [n]$ are assigned tile types according to f, any two adjacent sites have matching glues along their common side.
A2. f is minimally constrained: any tile map $g : P \to \Sigma^4$ satisfying A1 satisfies also:[1]

$$f(p_1)_{D_1} = f(p_2)_{D_2} \quad \Longrightarrow \quad g(p_1)_{D_1} = g(p_2)_{D_2},$$

for all partition classes $p_1, p_2 \in P$ and directions $D_1, D_2 \in \mathcal{D}$.

As an illustration, a most general tile assignment $f : I \to \Sigma^4$ for the **initial partition** $I = \{\{a\} \mid a \in [m] \times [n]\}$ is presented in Figure 3(a) and a MGTA for the partition of Figure 2(b) in Figure 3(b).

[1] For brevity we write $f(p)_D$ instead of $\sigma_D(f(p))$.



(a) (b) (c)

Fig. 3. (a) A MGTA for the constructible initial partition I (with a seed assembly in place). (b) Finished assembly for the pattern from Figure 2a. The tile set to construct this assembly is given in (c).

Given a partition $P \in X$ and a tile map $f : P \to \Sigma^4$, tile map $g : P \to \Sigma^4$ is obtained from f by **merging glues** a **and** b if for all $(p, D) \in P \times \mathcal{D}$ we have

$$g(p)_D = \begin{cases} a, & \text{if } f(p)_D = b \\ f(p)_D, & \text{otherwise} \end{cases} .$$

A most general tile assignment for a partition $P \in X$ can be found as follows. One starts with a map $f_0 : P \to \Sigma^4$ that assigns to each tile edge a unique glue type. Next, one considers all pairs of adjacent sites in $[m] \times [n]$ in some order and makes their common sides matching by merging the corresponding glues. This process generates a sequence of tile maps $f_0, f_1, f_2, \ldots, f_N = f$ and terminates after $N \le 2mn$ steps.

Lemma 1. *[7] The above algorithm generates a most general tile assignment.*

3 New Algorithms for Small Tile Sets

3.1 The PS-H Algorithm

Whereas the pruning heuristics of the PS-BB algorithm try to reduce the size of the search space in a "balanced" way, our new PS-H algorithm attempts to "greedily" optimise the order in which the coarsenings of a partition are explored, in the hope of being directly lead to close-to-optimal solutions. Such opportunism may be expected to pay off in case the success probability of the greedy exploration is sufficiently high, and the process is restarted sufficiently often, or equivalently several runs are explored in parallel.

The basic heuristic idea is to try to minimise the effect that a merge operation has on other partition classes than those which are combined. This can be achieved by prefering to merge classes already having as many common glues as possible. In this way one hopes to extend the number of steps the search takes before it runs into a conflict. For example, when merging classes p_1 and p_2 such that $f(p_1)_N = f(p_2)_N$ and $f(p_1)_E = f(p_2)_E$, the glues on the W and S edges of all other classes are unaffected. This way, the search avoids proceeding to a partition which is not constructible after the merge operation is completed. Secondarily, we prefer merging classes which already cover a large number of sites in $[m] \times [n]$. That is, one tries to grow a small number of large classes instead of growing all the classes at an equal rate.

Definition 3. *Given a partition P and a MGTA f for P, the* ***number of common glues*** *between classes $p, q \in P$ is defined by the function $G : P \times P \to \{0, 1, 2, 3, 4\}$,*

$$G(p, q) = \sum_{D \in \mathcal{D}} g(f(p)_D, f(q)_D),$$

where $g(\sigma_1, \sigma_2) = 1$ if $\sigma_1 = \sigma_2$ and 0 otherwise, for $\sigma_1, \sigma_2 \in \Sigma$.

Except for the bounding function, the PS-BB algorithm allows an arbitrary ordering $\{p_i, q_i\}, i = 1, \ldots, N$, for the children (coarsenings) $P[p_i, q_i]$ of a constructible partition P.[2] In the PS-H algorithm, we choose the ordering using the following heuristic. First form the set

$$H := \{\{p, q\} \mid p, q \in P, p \neq q, \exists k \in P(c) : p, q \subseteq k\}$$

of class pairs of same colour, and then repeat the following process until H is empty.

H1. Set $K := H$.
H2. Optimise the number of common glues:

$$K := \{\{p, q\} \in K \mid G(p, q) \geq G(u, v) \text{ for all } \{u, v\} \in K\}.$$

H3. Optimise the size of the larger class:

$$K := \{\{p, q\} \in K \mid \max\{|p|, |q|\} \geq \max\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}.$$

H4. Optimise the size of the smaller class:

$$K := \{\{p, q\} \in K \mid \min\{|p|, |q|\} \geq \min\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}.$$

H5. Pick some pair $\{p, q\} \in K$ at random and visit the partition $P[p, q]$.
H6. Remove $\{p, q\}$ from H:

$$H := H \setminus \{\{p, q\}\}.$$

The PS-H algorithm also omits the pruning process utilised by the PS-BB algorithm. That way, it aims to get to the small solutions quickly by reducing the computational resources used in a single merge operation.

Since step H5 of the heuristic above leaves room for randomisation, the PS-H algorithm performs differently with different random seeds. While some of the randomised runs may lead to small solutions quickly, others may get sidetracked into worthless expanses of the solution space. We make the best of this situation by running several instances of the algorithm in paraller, or equivalently, restarting the search several times with a different random seed. The notation PS-$\mathrm{H_n}$ denotes the heuristic partition search algorithm with n parallel search threads. The solution found by the PS-$\mathrm{H_n}$ algorithm is the smallest solution found by any of the n paraller threads.

3.2 Answer Set Programming

Answer Set Programming (ASP) [9] is a declarative logic programming paradigm for solving difficult combinatorial search problems. In ASP, a problem is described as a logic program, and an answer set solver is then used to compute stable models (answer sets) for the logic program.

ASP can be applied also to the PATS problem. In the following we give a brief desctiption on how to transform the PATS problem to an ASP program using the lparse [21] language. First, we define constant for each position of the grid $[m] \times [n]$, each colour, each available tile type and each available glue type. After that, a number of choice rules is introduced to associate a tile type with each positions of the grid, a glue type with each of the four sides of the tile types and a colour with each of the tile types. We also use choice rules to make the glues of every pair of adjacent tiles to match and to make the tile system deterministic, i.e. to ensure that every tile type has a unique pair of glues on its west and south edges.

We also compile the target pattern to a set of rules that associate every position of the grid with a colour. This description of the pattern is combined with the above-described program and given to an answer set solver, which then outputs a tile type for each position of the grid, given that such a solution exists. The program is run several times using an increasing number of available tile and glue types, until a solution is found.

[2] The notation $P[p, q]$ denotes the partition obtained from P by merging classes $p, q \in P$ into one.

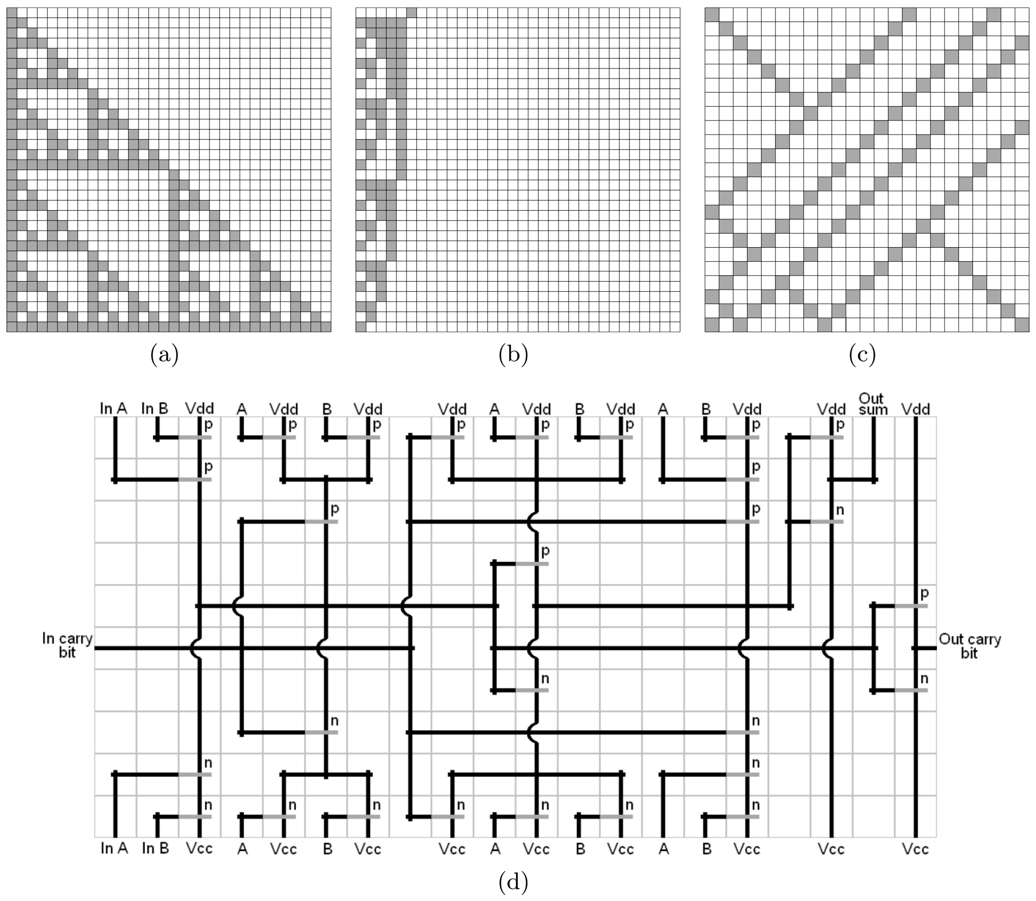


Fig. 4. (a) The 32×32 Sierpinski triangle pattern. (b) The 32×32 binary counter pattern. (c) The 23×23 "tree" pattern. (d) A CMOS full adder design that induces a 15-colour 20×10 pattern.

3.3 Results

The Partition Search Algorithms. Our implementation of the PS-H algorithm is based on the DFS implementation of the PS-BB algorithm used in [7], and we provide results on the PS-$\mathrm{H_n}$ algorithm for $n = 1, 2, 4, 8, 16$ and 32. We consider several different finite 2-coloured input patterns, two of which are classical examples of structured patterns: the discrete Sierpinski triangles of sizes 32×32 (Figure 4(a)) and 64×64, and the binary counter of size 32×32 (Figure 4(b)). Furthermore, we introduce a 2-coloured "tree" pattern of size 23×23 (Figure 4(c)) as well as a 15-coloured pattern of size 20×10 based on a CMOS full adder design (Figure 4(d), [3]). The Sierpinski triangle and binary counter patterns are known to have minimal solution of four tiles, while the minimal solutions for the tree pattern and the full adder pattern are unknown.

Figure 5 presents the evolution of the "current best solution" as a function of time for the (a) 32×32 and (c) 64×64 Sierpinski patterns. To allow fair comparison, Figure 5(b) and 5(d) present the same data with respect to the total processing time taken by all the parallely running instances. The experiments were repeated 21 times and the median of the results is depicted. In 37% of all the runs conducted, the PS-H algorithm is able to find the optimal 4-tile solution for the 32×32 Sierpinski pattern in less than 30 seconds. The similar percentage for the 64×64 Sierpinski pattern is 34% in one hour. Remarkably, the algorithm performs only from 1030 to 1035 and from 4102 to 4107 merge steps before arriving at the optimal solution for the 32×32 and 64×64 patterns, respectively. In other words, the search rarely needs to backtrack. In contrast, the smallest solutions found by the PS-BB algorithm are 42 tiles, reached after $1.4 \cdot 10^6$ merge steps, and 95 tiles, reached after $5.9 \cdot 10^6$ merge steps.

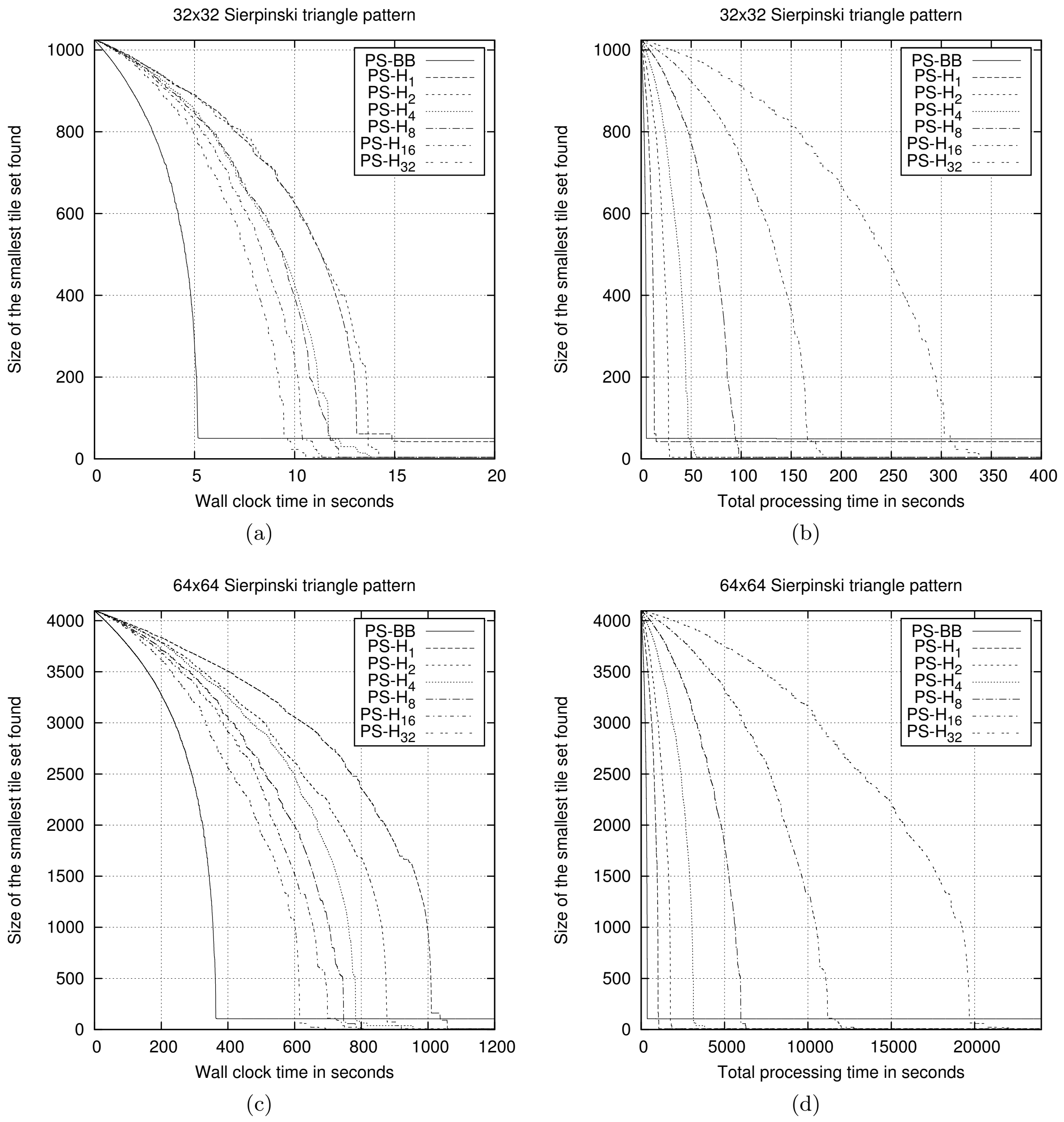


Fig. 5. Evolution of the smallest tile set found as a function of time. The time axis measures (a), (c) wall clock time and (b), (d) wall clock time multiplied by the number of parallel instances.

In Figure 6 we present the corresponding results for the 32×32 binary counter and the 23×23 tree patterns. The size of the smallest solutions found by the PS-H$_{32}$ algorithm were 20 (cf. 307 by PS-BB) and 25 (cf. 192 by PS-BB) tiles, respectively. In the case of the tree pattern, the parallelisation brings significant advantage over a single run. Finally, Figures 7(a)-7(b) show the results for the 20×10 15-colour CMOS full adder pattern. In this case, the improvement over the previous PS-BB algorithm is less clear. The PS-H$_{32}$ algorithm is able to find a solution of 58 tiles, whereas the PS-BB algorithm gives a solution of 69 tiles.

Answer Set Programming. We used the answer set solver SMODELS [21] to run our experiments. We consider two patterns having a minimal solution of four tiles, the Sierpinski triangle and the binary counter.

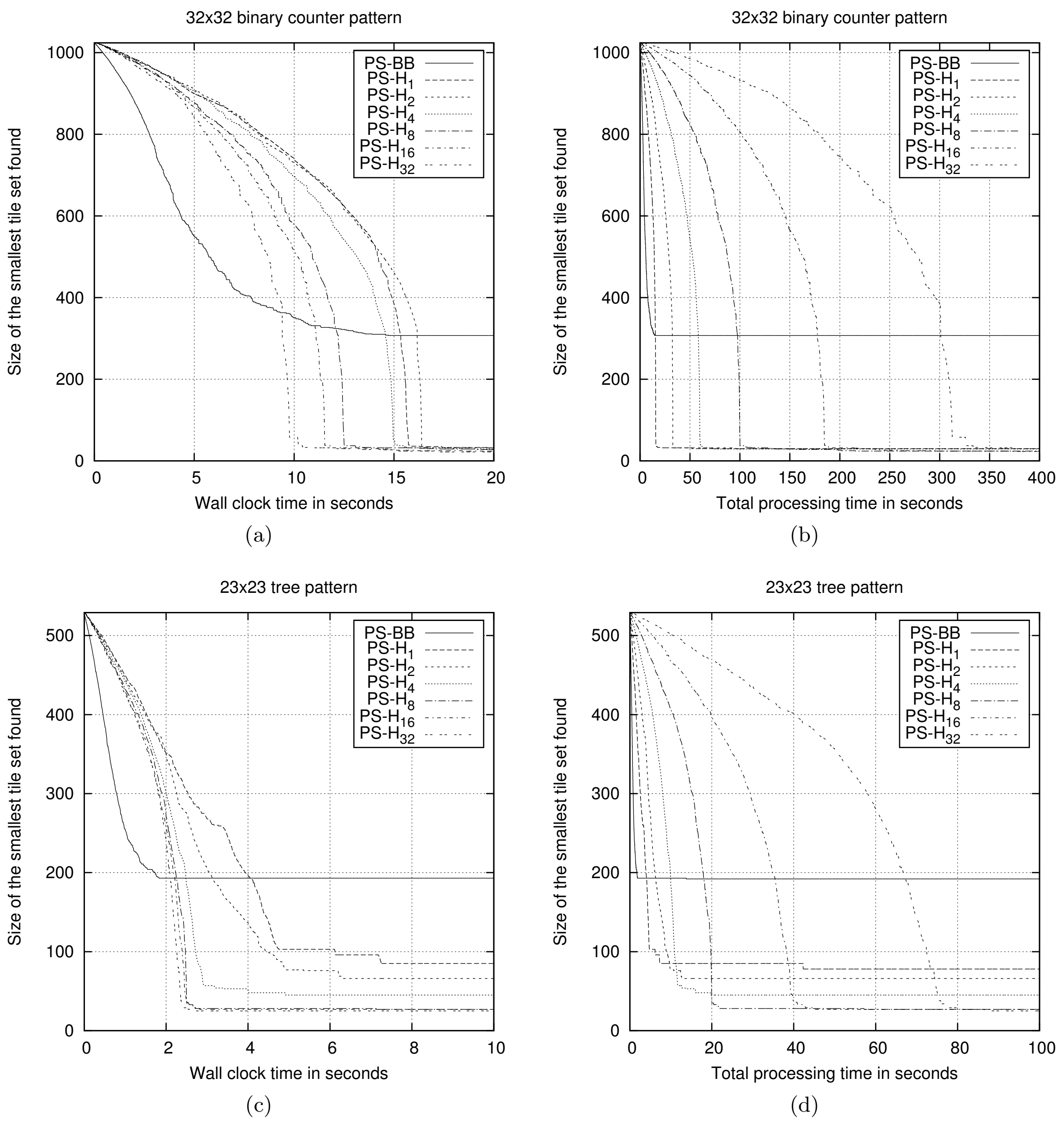


Fig. 6. Evolution of the smallest tile set found as a function of time. The time axis measures (a), (c) wall clock time and (b), (d) wall clock time multiplied by the number of parallel instances.

The program was executed for pattern sizes $1\times 1, 2\times 2, 3\times 3, \ldots, 100\times 100$. The running time of the program is presented in Figure 7(c) for the Sierpinski triangle and in Figure 7(d) for the binary counter. SMODELS was able to find the minimal solution for the 100×100 Sierpinski triangle in little over 5 hours and for the 100×100 binary counter in less than two hours. The running time grows rather consistently with the pattern size, but interestingly, there are a few notable exceptions. The running times for the 49×49 and 55×55 Sierpinski patterns, not shown in the Figure, are close to 40 hours and close to 10 hours, respectively. That is clearly out of line with other proximate pattern sizes. For the binary counter patterns of size 29×29, 35×35 and 60×60 SMODELS was not able to find a solution in less than 48 hours. Thus, the running times for those instances are also missing from the Figure.

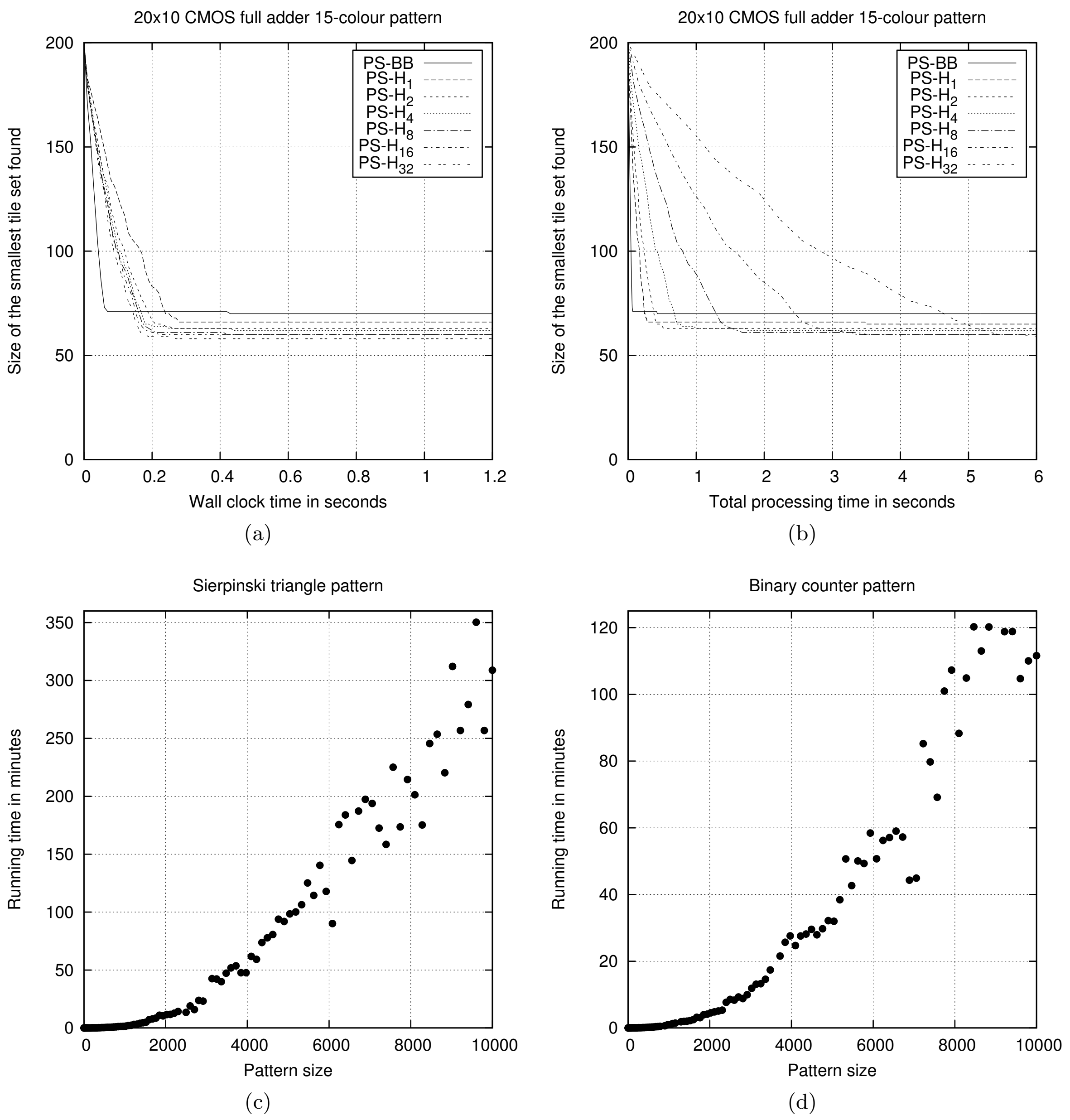


Fig. 7. (a)-(b) Evolution of the smallest tile set found for the 20×10 full adder pattern as a function of time. The time axes measure wall clock time and wall clock time multiplied by the number of parallel instances. (c)-(d) Running time of SMODELS for the Sierpinski triangle and the binary counter patterns.

Based on the above results, the ASP approach performs well when considering patterns with a small minimal solution. However, the running time seems to increase dramatically with patterns having a larger minimal solution. This renders our ASP program less applicable in the real world compared to the partition search algorithms.

4 Reliability of Tile Sets

4.1 The Kinetic Tile Assembly Model

In this Section, we assess the reliability of the tile sets produced by the PS-BB and PS-H algorithms, using the kinetic Tile Assembly Model (kTAM), which has been proposed by Winfree [22] as a kinetic counterpart of the aTAM. Several variants of the kTAM exist, see e.g. [5, 20], however the main elements are similar.

The kTAM simulates two types of reactions, each involving an assembly, i.e. a crystal structure consisting of several merged tiles, and a tile: association of tiles to the assembly, and dissociation.[3] In the first type of reaction, any tile can attach to the assembly at any position (up to the assumption that tile alignment is preserved), even if only a weak bond is formed; the rate of this reaction is proportional to the concentration of free tiles in the solution. In the second type of reaction, any tile can detach from the assembly, with a rate which is exponentially correlated with the total strength of the bonds between the tile and the assembly. Thus, tiles which are connected to the assembly by fewer or weaker bonds, i.e. incorrect "sticky end" matches, are more prone to dissociation than those which are strongly connected by several bonds (well paired sticky end sequences).

In the following, we follow the notations of [22]. For any tile t, the rate constant r_f of the association (forward) reaction of t to an existing assembly is given by

$$r_f = k_f[t] \text{ /sec},$$

where $[t]$ is the concentration in solution of free tiles of type t and k_f is a temperature dependent parameter. In case of DNA double-crossover (DX) tiles, this parameter is given by the formula

$$k_f = A_f e^{-E_f/RT},$$

where $A_f = 5 \cdot 10^8$ M/sec, $E_f = 4000$ cal/mol, $R = 2$ cal/mol/K, and T is the temperature (in K).

In the case of dissociation (reverse) reactions, for a tile t which is connected to the assembly by a total bond strength b, the rate constant $r_{r,b}$ is given by the formula

$$r_{r,b} = k_f e^{\Delta G_b^0/RT},$$

where $\Delta G_b^0/RT$ is the standard free energy needed in order to break the b bonds. In case of DX tiles, as the glues of the tiles are implemented using 5-base long single-stranded DNA molecules, ΔG_b^0 can be estimated using the nearest-neighbor model [12] to

$$\Delta G_b^0 = e^{5b\left(11-\frac{4000}{T}\right)+3} \text{ /sec.}$$

Moreover, b can range with integer values from 0 to 4, corresponding to the cases when the tile is totally erroneously placed in the assembly (no bond connects it to the crystal) and when the tile is fully integrated into the assembly (all its four sticky ends are correctly matched), respectively.

In order to easily represent and scale the system, the free parameters involved in the formulas of the rate constants r_f and $r_{r,b}$ are re-distributed into just two dimensionless parameters, G_{mc} and G_{se}, where the first is dependent on the initial tile concentration, while the second is dependent on the assembly temperature:

$$r_f = \hat{k}_f e^{-G_{mc}} \qquad r_{r,b} = \hat{k}_f e^{-bG_{se}}$$

where, in case of DX tiles, $\hat{k}_f = e^3 k_f$ is adjusted in order to take into consideration possible entropic factors, such as orientation or location of the tiles. The previous parameter re-distribution is made possible as a result of the assumption made in the initial kTAM [22] that all tile types are provided into the solution in similar concentrations, and that the consumption in time of the free monomers is negligible compared to the initial concentration.

[3] Note that interactions between two tiles, such as forming a new assembly, as well as interactions between two assemblies, are not taken into consideration in the initial model [22]. However, they are studied in some of the later developed variants of the kTAM, see e.g. [20].

4.2 Computing the Reliability of a Tile Set

The probability of errors in the assembly process can be made arbitrarily low, at the cost of reduced speed, by choosing appropriate physical conditions [22]. However, we would like to be able to compare the error probability of different tile sets producing the same finite pattern, under the same physical conditions. Given the amount of time the assembly process is allowed to take, we define the **reliability of a tile set** to be the probability that the assembly process of the tile system in question completes without any incorrect tiles being present in the terminal configuration. In this section, we present a method for computing the reliability of a tile set, based on Winfree's analysis of the kTAM and the notion of **kinetic trapping** in [22].

We call the West and South edges of a tile its **input edges**. First, we derive the probability of the correct tile being frozen at a particular site under the condition that the site already has correct tiles on its input edges. Let M^1_{ij} and M^2_{ij} be the number of tile types having one mismatching and two mismatching glue types, respectively, between them and the correct tile type for site $(i,j) \in [m] \times [n]$. Now, for a deterministic tile set T, the total number of tiles is $|T| = 1 + M^1_{ij} + M^2_{ij}$ for all i and j. Given that a site has correct tiles on its input edges, a tile is correct for that site if and only if it has two matches on its input edges.

In what follows, we assume that correct tiles are attached at sites $(i-1,j)$ and $(i,j-1)$. The model for kinetic trapping [22] gives four distinct cases in the situation preceding the site (i,j) being frozen by further growth: (E) An empty site with "off-rate" of $|T|r_f$. (C) The correct tile with off-rate of $r_{r,2}$. (A) A tile with one match, with off-rate of $r_{r,1}$. (I) A tile with no matches, with off-rate of $r_{r,0}$. Additionally, we have two sink states FC and FI, which represent frozen correct and frozen incorrect tiles, respectively. The rate of a site being frozen is equal to the rate of growth $r^* = r_f - r_{r,2}$. Let $p_S(t)$ denote the probability of the site being in state (S) after t seconds for all $S \in \{E, C, A, I, FC, FI\}$. To compute the frozen distribution, we write rate equations for the model of kinetic trapping:

$$M\mathbf{p}(t) := \begin{bmatrix} -|T|r_f & r_{r,2} & r_{r,1} & r_{r,0} & 0 & 0 \\ r_f & -r_{r,2} - r^* & 0 & 0 & 0 & 0 \\ M^1_{ij} r_f & 0 & -r_{r,1} - r^* & 0 & 0 & 0 \\ M^2_{ij} r_f & 0 & 0 & -r_{r,0} - r^* & 0 & 0 \\ 0 & r^* & 0 & 0 & 0 & 0 \\ 0 & 0 & r^* & r^* & 0 & 0 \end{bmatrix} \begin{bmatrix} p_E(t) \\ p_C(t) \\ p_A(t) \\ p_I(t) \\ p_{FC}(t) \\ p_{FI}(t) \end{bmatrix} = \dot{\mathbf{p}}(t),$$

where $\mathbf{p}(0) = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$. To compute the probability of the site being frozen with the correct tile, $p_{FC}(\infty)$, we make use of the steady state of the related flow problem [22]:

$$M\mathbf{p}(\infty) = [1 \quad 0 \quad 0 \quad 0 \quad p_{FC}(\infty) \quad p_{FI}(\infty)]^T = \dot{\mathbf{p}}(\infty),$$

which gives us a system of linear equations. This system has a single solution, namely

$$p_{FC}(\infty) = \frac{\frac{1}{r^* + r_{r,2}}}{\frac{1}{r^* + r_{r,2}} + \frac{M^1_{ij}}{r^* + r_{r,1}} + \frac{M^2_{ij}}{r^* + r_{r,0}}} = \Pr(C(i,j) \,|\, C(i-1,j) \cap C(i,j-1)),$$

where $C(i,j)$ denotes the correct tile being frozen on site (i,j).

The assembly process can be thought of as a sequence of tile addition steps $(a_1, a_2, \ldots, a_N)$ where $a_k = (i_k, j_k)$, $k = 1, 2, \ldots, N$ denotes a tile being frozen on site (i_k, j_k). Due to the fact that the assembly process of the tile systems we consider proceeds uniformly from south-west to north-east, $\{(i_k - 1, j_k), (i_k, j_k - 1)\} \subseteq \{a_1, a_2, \ldots, a_{k-1}\}$ for all $a_k = (i_k, j_k)$. We assume that tiles elsewhere in the configuration do not affect the probability. Now we can compute the probability of a finite-size pattern of size N assembling without any errors, i.e. the reliability of that pattern:

$$\begin{aligned} \Pr(\text{correct pattern}) &= \Pr(C(a_1) \cap C(a_2) \cap \cdots \cap C(a_N)) \\ &= \Pr(C(a_1))\Pr(C(a_2) \,|\, C(a_1)) \cdots \Pr(C(a_N) \,|\, C(a_1) \cap C(a_2) \cap \cdots \cap C(a_{N-1})) \\ &= \prod_{i,j} \Pr(C(i,j) \,|\, C(i-1,j) \cap C(i,j-1)) \ . \end{aligned}$$

We have computed the probability in terms of G_{mc} and G_{se}. Given the desired assembly speed, we want to minimise the error probability by choosing values for G_{mc} and G_{se} appropriately. If the assembly process is allowed to take t seconds, the needed assembly speed for an $m \times n$ pattern is approximately $r^* = \frac{\sqrt{m^2+n^2}}{t}$.

$$\Pr(C(i,j) \,|\, C(i-1,j) \cap C(i,j-1)) = \frac{\frac{1}{r^*+r_{r,2}}}{\frac{1}{r^*+r_{r,2}} + \frac{M_{ij}^1}{r^*+r_{r,1}} + \frac{M_{ij}^2}{r^*+r_{r,0}}} \approx \frac{1}{1 + M_{ij}^1 \frac{r^*+r_{r,2}}{r^*+r_{r,1}}} \ .$$

For small error probability and $2G_{se} > G_{mc} > G_{se}$,

$$\Pr(\neg C(i,j) \,|\, C(i-1,j) \cap C(i,j-1)) \approx M_{ij}^1 \frac{r^* + r_{r,2}}{r^* + r_{r,1}} \approx M_{ij}^1 e^{-(G_{mc}-G_{se})} =: M_{ij}^1 e^{-\triangle G} \ .$$

From

$$r^* = r_f - r_{r,2} = \hat{k}_f (e^{-G_{mc}} - e^{-2G_{se}})$$

we can derive

$$G_{se} = -\frac{1}{2} \log(e^{-G_{mc}} - \frac{r^*}{\hat{k}_f}) \ .$$

Now we can write $\triangle G$ as a function of G_{mc}:

$$\triangle G(G_{mc}) = G_{mc} - G_{se} = G_{mc} + \frac{1}{2} \log(e^{-G_{mc}} - \frac{r^*}{\hat{k}_f}) \ .$$

We find the maximum of $\triangle G$, and thus the minimal error probability, by differentation:

$$G_{mc} = -\log(2\frac{r^*}{\hat{k}_f}) \ .$$

Thus, if the assembly time is t seconds, the maximal reliability is achieved at

$$G_{mc} = -\log(2\frac{\sqrt{m^2+n^2}}{t\hat{k}_f}), \qquad G_{se} = -\frac{1}{2}\log(\frac{\sqrt{m^2+n^2}}{t\hat{k}_f}) \ .$$

4.3 Results

In this Section, we present results on computing the reliability of tile sets using the method we presented in Section 4.2. We assume the assembly process takes place in room temperature (298 K). As a result, we use the value $k_f = A_f e^{-E_f/RT} \approx 6 \cdot 10^5$ /M/sec for the forward reaction rate.

Figure 8(a) shows the reliability of the 4-tile solution to the Sierpinski pattern as a function of pattern size, using five distinct assembly times. As is expected, the longer the assembly time, the better the reliability.

We also applied the method for computing the reliability to the tile sets found by the partition search algorithms. Our results show that the heuristic described in Section 3.1 improves not only the size of the tile sets found, but also the reliability of those tile sets. This can be easily understood by considering the following: The reliability of a tile set is largely determined by the number of tile types that have the same glue as some other tile type on either one of their input edges. Since the heuristic prefers merging class pairs with common glues, it reduces the number of such tile types effectively.

Figures 8(b), 8(c) and 8(d) present the reliability of the tile sets found by the PS-H and PS-BB algorithms for the 32×32 Sierpinski triangle pattern, using assembly times of one hour, one day (24 hours) and one week, respectively. The runs were repeated 100 times; the mean reliability of each tile set size as well as the 10th and 90th percentiles are shown.

As for reliability, we expect a large set of runs of the PS-BB algorithm to produce a somewhat decent sample of all the possible tile sets for a pattern. Based on this, large and small tile sets seem to have a high reliability while medium-size tile sets are clearly more unreliable on average. This observation reduces the problem of finding reliable tile sets back to the problem of finding small tile sets. However, it is important to note that artifacts of the algorithm may have an effect on the exact reliability of the tile sets found.

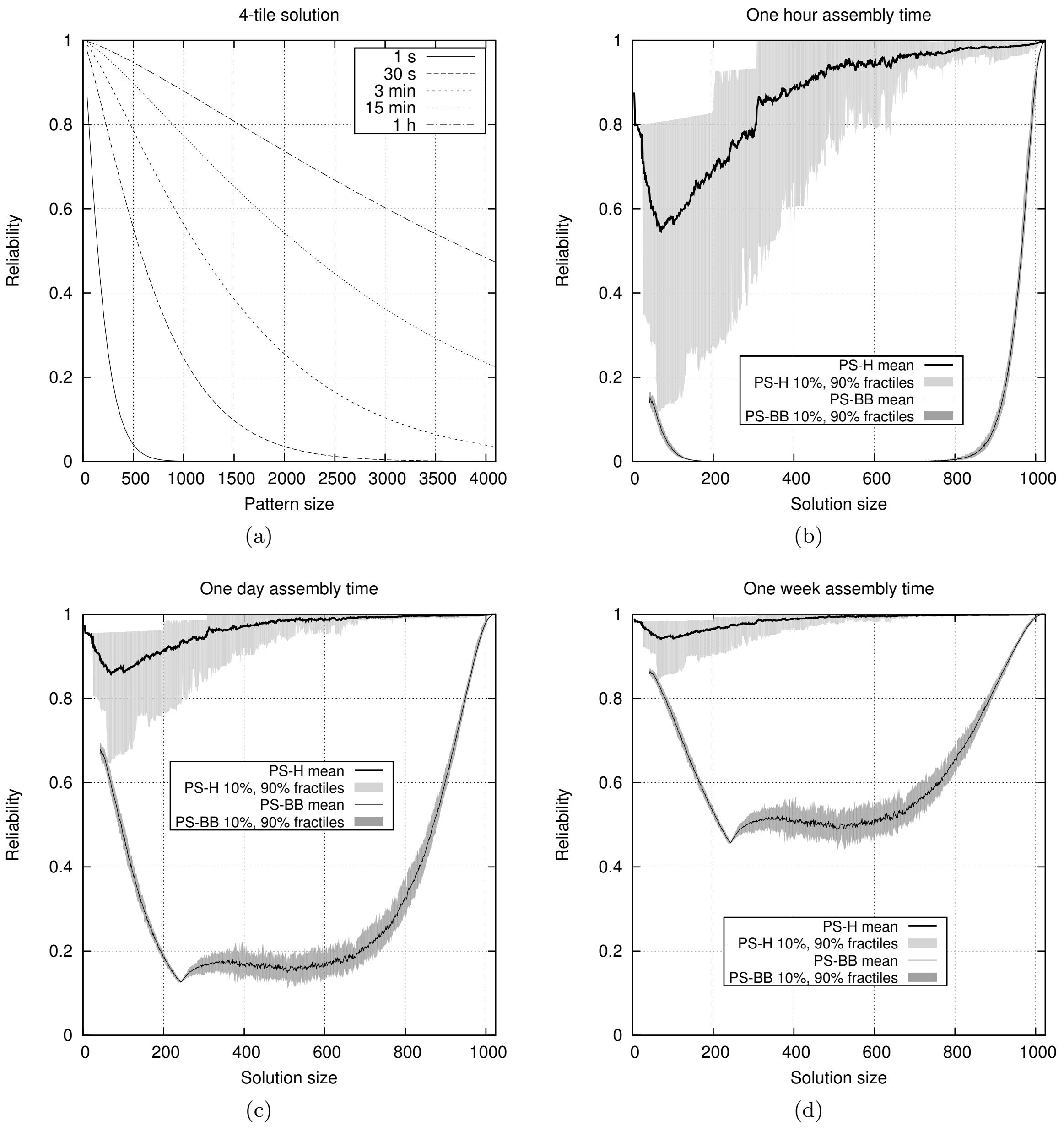


Fig. 8. (a) Reliability of the minimal tile as a function of pattern size for the Sierpinski pattern, using several different assembly times. (b)-(d) Reliability of the solutions for the 32×32 Sierpinski pattern found by the PS-H and PS-BB algorithms, allowing assembly time of one hour, one day and one week.

5 Conclusion

We have presented a new algorithm, PS-H, for addressing the problem of finding small tile sets that have a high probability of self-assembling a given target pattern. Our results show that for most patterns, the new algorithm is able to find significantly smaller solutions in a reasonable amount of time compared to the earlier PS-BB algorithm. Also the reliability of the tile sets produced by the PS-H algorithm clearly exceed that of the tile sets produced by the PS-BB algorithm. We have also suggested the use of a novel approach, ASP, to solve instances of the PATS problem.

Sources

[1] L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund, *Combinatorial optimization problems in self-assembly*, in Proc. 34th Annual ACM Symp. on Theory of Computing, ACM, 2002, pp. 23–32.

[2] E. Braun, Y. Eichen, U. Sivan, and G. Ben-Yoseph, *DNA-templated assembly and electrode attachment of a conducting silver wire*, Nature, 391 (1998), pp. 775–778.

[3] E. Czeizler, T. Lempiäinen, and P. Orponen, *A design framework for carbon nanotube circuits affixed on DNA origami tiles*, in Proc. 8th Annual Conf. on Foundations of Nanoscience: Self-Assembled Architectures and Devices, 2011, pp. 186–187. Poster.

[4] M. Endo, T. Sugita, Y. Katsuda, K. Hidaka, and H. Sugiyama, *Programmed-assembly system using DNA jigsaw pieces*, Chemistry - A European Journal, 16 (2010), pp. 5362–5368.

[5] K. Fujibayashi and S. Murata, *Precise simulation model for DNA tile self-assembly*, IEEE Trans. Nanotechnology, 8 (2009).

[6] C. P. Gomes and B. Selman, *Algorithm portfolios*, Artificial Intelligence, 126 (2001), pp. 43–62.

[7] M. Göös and P. Orponen, *Synthesizing minimal tile sets for patterned DNA self-assembly*, in Proc. 16th Intl. Conf. on DNA Computing and Molecular Programming, vol. 6518 of LNCS, Springer, 2011, pp. 71–82.

[8] K. N. Kim, K. Sarveswaran, L. Mark, and M. Lieberman, *DNA origami as self-assembling circuit boards*, in Proc. 9th Intl. Conf. on Unconventional Computation, vol. 6079 of LNCS, Springer, 2010.

[9] V. Lifschitz, *What is answer set programming?*, in Proceedings of the 23rd national conference on Artificial intelligence - Volume 3, AAAI Press, 2008, pp. 1594–1597.

[10] W. Liu, H. Zhong, R. Wang, and N. C. Seeman, *Crystalline two-dimensional DNA-origami arrays*, Angewandte Chemie International Edition, 50 (2011), pp. 264–267.

[11] M. Luby, A. Sinclair, and D. Zuckerman, *Optimal speedup of Las Vegas algorithms*, Information Processing Letters, 47 (1993), pp. 173–180.

[12] J. S. Lucia, H. T. Allawi, and P. A. Seneviratne, *Improved nearest neighbor parameters for predicting DNA duplex stability*, Biochemistry, 35 (1996).

[13] X. Ma and F. Lombardi, *Synthesis of tile sets for DNA self-assembly*, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 27 (2008), pp. 963–967.

[14] ———, *On the computational complexity of tile set synthesis for DNA self-assembly*, IEEE Trans. Circuits and Systems II: Express Briefs, 56 (2009), pp. 31–35.

[15] H. T. Maune, S. Han, R. D. Barish, M. Bockrath, W. A. G. III, P. W. K. Rothemund, and E. Winfree, *Self-assembly of carbon nanotubes into two-dimensional geometries using DNA origami templates*, Nature Nanotechnology, 5 (2010), pp. 61–66.

[16] S. H. Park, C. Pistol, S. J. Ahn, J. H. Reif, A. R. Lebeck, C. Dwyer, and T. H. LaBean, *Finite-size, fully addressable DNA tile lattices formed by hierarchical assembly procedures*, Angewandte Chemie International Edition, 45 (2006), pp. 735–739.

[17] A. Rajendran, M. Endo, Y. Katsuda, K. Hidaka, and H. Sugiyama, *Programmed two-dimensional self-assembly of multiple DNA origami jigsaw pieces*, ACS Nano, 5 (2011), pp. 665–671.

[18] P. W. K. Rothemund, *Folding DNA to create nanoscale shapes and patterns*, Nature, 440 (2006), pp. 297–302.

[19] P. W. K. Rothemund and E. Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, in Proc. 32nd Annual ACM Symp. on Theory of Computing, ACM, 2000, pp. 459–468.

[20] R. Schulman and E. Winfree, *Programmable control of nucleation for algorithmic self-assembly*, SIAM Journal of Scientific Computing, 39 (2009).

[21] T. Syrjänen and I. Niemelä, *The Smodels system*, in Logic Programming and Nonmotonic Reasoning, T. Eiter, W. Faber, and M. Truszczynski, eds., vol. 2173 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2001, pp. 434–438.

[22] E. Winfree, *Simulations of Computing by Self-Assembly*, Technical Report CSTR 1998.22, California Institute of Technology, 1998.

[23] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, *Design and self-assembly of two-dimensional dna crystals*, Nature, 394 (1998).

[24] H. Yan, S. H. Park, G. Finkelstein, J. H. Reif, and T. H. LaBean, *DNA-templated self-assembly of protein arrays and highly conducive nanowires*, Science, 301 (2003), pp. 1882–1884.

[25] Z. Zhao, H. Yan, and Y. Liu, *A route to scale up DNA origami using DNA tiles as folding staples*, Angewandte Chemie International Edition, 49 (2010), pp. 1414–1417.