Search methods for tile sets in patterned DNA self-assembly[☆]

Mika Göös[1], Tuomo Lempiäinen[2], Eugen Czeizler[3,*], Pekka Orponen[3]

Department of Information and Computer Science and
Helsinki Institute for Information Technology HIIT
Aalto University, Finland

Abstract

The Pattern self-Assembly Tile set Synthesis (PATS) problem, which arises in the theory of structured DNA self-assembly, is to determine a set of coloured tiles that, starting from a bordering seed structure, self-assembles to a given rectangular colour pattern. The task of finding minimum-size tile sets is known to be NP-hard. We explore several complete and incomplete search techniques for finding minimal, or at least small, tile sets and also assess the reliability of the solutions obtained according to the kinetic Tile Assembly Model.

Keywords: DNA self-assembly, tilings, Tile Assembly Model, pattern assembly, tile set synthesis, reliable self-assembly

1. Introduction

Algorithmic assembly of nucleic acids (DNA and RNA) has advanced extensively in the past 30 years, from a seminal idea to the current designs and experimental implementations of complex nanostructures and nanodevices with dynamic, programmable evolution and machinelike properties. Recent developments in the field include fundamental constructions such as *in vitro* complex 3D pattern formation and functionalisation [3, 4], robotic designs such as mobile arms, walkers, motors [5, 6], computational primitives [7, 8], and also applications to *in vivo* biosensors [9] and potential drug delivery mechanisms and therapeutics [10].

Self-assembly of nucleic acids can be seen both as a form of structural nanotechnology and as a model of computation. As a computational model, one first encodes the input of a computational problem into an algorithmically designed (DNA) pattern or shape. Then, by making use of both the initial oligomer design and the intrinsic properties of the self-assembly system, one manipulates the structure to produce a new architecture that encodes the desired output.

As a nanotechnology, the goal of algorithmic (DNA/RNA) self-assembly is to design oligomer sequences that in solution would autonomously (or with as little interaction as possible) assemble into complex polymer structures. These may have both static and dynamic properties, may bind other molecules such as gold nanoparticles or various proteins, may act as fully addressable scaffolds, or may be used for further manipulation. Such molecular constructions can be composed of from only a couple of DNA strands to more than 200 and, in some cases, can change their conformation and achieve distinct functionalities.

[☆]Preliminary versions of parts of this work have appeared in the *Proceedings of the 16th* and *17th International Conference on DNA Computing and Molecular Programming* (Hong Kong, China, June 2010 and Pasadena, CA, USA, September 2011, respectively) [1, 2].

[*]Corresponding author.

[1]Current affiliation: Department of Computer Science, University of Toronto. Email address: mika.goos@mail.utoronto.ca.

[2]Current affiliation: Helsinki Institute for Information Technology HIIT and Department of Computer Science, University of Helsinki. Email address: tuomo.lempiainen@helsinki.fi.

[3]Email address: firstname.lastname@aalto.fi.

In recent years there has been a growing interest in integrating these two directions, in order to obtain complex supramolecular constructions with interdependencies between computational functions and conformational switching. Such approaches are envisioned due to a key property of nucleic acid scaffolds, viz. their modularity: multiple functional units can be attached to a common scaffold, thus giving rise to multifunctional devices. Thus, the 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 [11–14]. Also laboratory techniques for synthesising the requisite 2D DNA template lattices, many based on Rothemund's [15] DNA origami tiles, have recently been demonstrated by many groups [16, 17].

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 constructed 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.[4] Now, a given target pattern can be assembled from many different families of base tiles, and to improve the laboratory synthesis it is 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 characteristics 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 [19], who formulated it as a combinatorial optimisation problem, the *Pattern self-Assembly Tile set Synthesis* (PATS) problem, and also proposed two greedy heuristic algorithms for solving the task. The problem was recently proved to be NP-hard [20, 21], and hence finding an absolutely minimum-size tile set for a given pattern most likely requires an exponential amount of time in the worst case. Thus the problem needs to be addressed either with complete methods yielding optimal tile sets for small patterns, or incomplete methods that work also for larger patterns but do not guarantee that the tile sets produced are of minimal size. In this work, we present search algorithms covering both approaches and assess their behaviour experimentally using both randomly generated and benchmark pattern test sets. We attend both to the running time of the respective algorithms, and to the size and assembly reliability of the tile sets produced.

In the following, we first in Section 2 present an overview of the underlying tile assembly model [22, 23] and the PATS problem [19], and then in Section 3 discuss the search space of pattern-consistent tile sets (viewed abstractly as partitions of the ambient rectangular grid). In Section 4 we proceed to describe our exhaustive partition-search branch-and-bound algorithm (PS-BB) to find tile sets of absolutely minimum cardinality. The algorithm makes use of a search tree in the lattice of grid partitions, and an efficient bounding function to prune this search tree.

While the PS-BB algorithm can be used to find certifiably minimal tile sets for small patterns, the size of the search space grows so rapidly that the algorithm hits a complexity barrier at approximately pattern sizes of 7×7 tiles, for random test patterns. Thus, in a second approach, presented in Section 5, we tailor the basic partition-search framework of the PS-BB algorithm towards the goal of finding small, but not necessarily minimal tile sets. Instead of a systematic branch-and-bound pruning and traversal of the complete search space, the modified algorithm PS-H applies heuristics which attempt to optimise the order of the directions in which the space is explored.

It is well known in the heuristic optimisation community [24, 25] 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 algorithm 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.

As a third alternative, presented in Section 6, we formulate the PATS problem as an Answer Set Programming (ASP) task [26], and apply a generic ASP solver to find solutions to it. Here our experimental results indicate that for patterns with a small optimal solution, the ASP approach indeed works well in discovering that solution.

[4]For examples of such tile-based high-level designs for nano-electric circuits cf. Appendix A, which summarises a scheme from Czeizler et al. [18].

Given the inherently stochastic nature of the DNA self-assembly process, it is important also to assess the reliability of a given tile set, i.e. the probability of its error-free self-assembly to the desired target pattern. In Section 7 we introduce a method for estimating this quantity, based on Winfree's analysis of the kinetic Tile Assembly Model [22]. We present experimental data on the reliability of tile sets found by the PS-BB and PS-H algorithms and find that also here the heuristic optimisations introduced in the PS-H approach result in a notable improvement over the basic PS-BB method.

2. Preliminaries

In this section, we first briefly review the abstract Tile Assembly Model (aTAM) as introduced by Winfree and Rothemund [22, 23] and then summarise the PATS problem [19].

2.1. The abstract Tile Assembly Model

The aTAM is a custom-made generalisation of Wang tile systems [27, 28], designed for the study of self-assembly systems. The basic components of the aTAM are non-rotatable unit square tiles, uniquely defined by the sets of four "glues" assigned to their edges. The glues come from a finite alphabet, and each pair of two glues is associated a strength value that determines the stability of a link between two tiles having these glues on the abutting edges. In most cases, it is assumed that the strength of two distinct glues is zero, while a pair of matching glues has strength either 1 or 2.

Let $\mathcal{D} = \{N, E, S, W\}$ be the set of four functions $\mathbb{Z}^2 \to \mathbb{Z}^2$ corresponding to the four cardinal directions:[5] $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S = N^{-1}$ and $W = E^{-1}$. Let Σ be a finite set of *glue types* and $s \colon \Sigma \times \Sigma \to \mathbb{N}$ a *glue strength* function such that, unless otherwise specified, $s(\sigma, \sigma') > 0$ only if $\sigma = \sigma'$. A *tile type* $t \in \Sigma^4$ is a quadruple $(\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t))$ of glue types for each side of the unit square. A *tile system* $T \subseteq \Sigma^4$ is a finite collection of different tile types.

A *(tile) assembly* $\mathcal{A}$ is a partial mapping $\mathcal{A} \colon \mathbb{Z}^2 \to \Sigma^4$ that assigns tiles to locations in the two-dimensional grid. A *tile assembly system* (TAS) $\mathscr{T} = (T, \mathcal{S}, s, \tau)$ consists of a tile system T, a *seed assembly* $\mathcal{S}$, a glue strength function s and a *temperature* $\tau \in \mathbb{Z}^+$ (we use $\tau = 2$). The seed structure $\mathcal{S}$ can be either an individual tile or a connected, finite assembly. Given an existing (connected) assembly $\mathcal{A}$, such as the seed structure $\mathcal{S}$, a tile from T can adjoin the assembly if the total strength of the binding, given by the sum of all strength function values among the glues placed on the boundary between the tile and the assembly, reaches or surpasses the temperature threshold τ. Note that tiles of the seed assembly $\mathcal{S}$ do not need to be in the tile system T, but that $\mathcal{S}$ can be extended only by tiles from T.

Formally, we say that 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.

In Figure 1 we present a TAS with seven tile types and temperature $\tau = 2$ which, starting from the seed tile, assembles a continuously growing structure that corresponds to a binary counter pattern (see Figure 9(b)). Out of the seven tile types in Figure 1(a), one can distinguish the tile **s** used as a seed, two tile types which assemble the boundary of the structure, and four rule-tile types (two of which are distinguished by **x** and **y**), which fill the area in between the L-shaped boundary. Considering the partial assembly presented in Figure 1(b), a tile of type **y** can adjoin the assembly at position $(4, 3)$ since

$$s(\sigma_S(\mathbf{y}), \sigma_{S^{-1}}(\mathcal{A}(S(4, 3)))) + s(\sigma_W(\mathbf{y}), \sigma_{W^{-1}}(\mathcal{A}(W(4, 3)))) = 1 + 1 \geq \tau,$$

while a tile of type **x** cannot adjoin the assembly at the same position (i.e. $(4, 3)$) since

$$s(\sigma_S(\mathbf{x}), \sigma_{S^{-1}}(\mathcal{A}(S(4, 3)))) + s(\sigma_W(\mathbf{x}), \sigma_{W^{-1}}(\mathcal{A}(W(4, 3)))) = 0 + 1 < \tau.$$

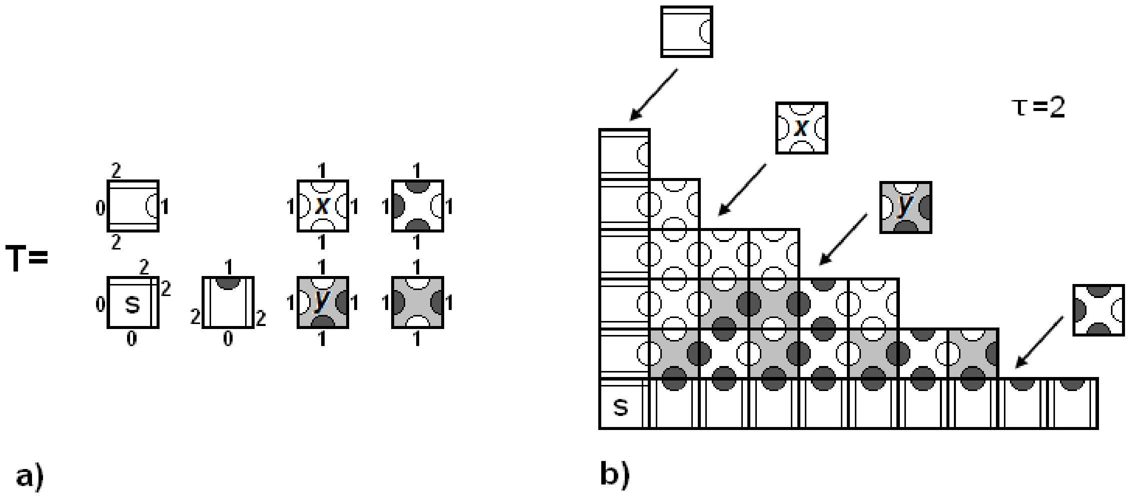


Figure 1: (a) The binary counter tile set [23]. The different glues are graphically differentiated, while their associated strengths are marked numerically. The colour of the tiles is an indicator of which tiles represent a black spot and which tiles represent a white spot in the pattern. (b) The assembly of the binary counter pattern for a TAS using the tile set T, a seed structure consisting of the single tile **s** and the temperature threshold $\tau = 2$.

Let $\to^*_{\mathscr{T}}$ be the reflexive transitive closure of $\to_{\mathscr{T}}$. A TAS $\mathscr{T}$ *produces* an assembly $\mathcal{A}$ if $\mathcal{A}$ is an extension of the seed assembly $\mathcal{S}$, that is, $\mathcal{S} \to^*_{\mathscr{T}} \mathcal{A}$. 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 does 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 $\mathscr{T}$. In case of finite assemblies, an equivalent definition of determinism is that 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 $\mathcal{P}$. In this case 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) [19])**.**

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

Finding minimal solutions (in terms of $|T|$) to the PATS problem was claimed to be NP-hard by Ma and

[5]In many cases, we use the elements of $\mathcal{D}$ just as direction labels and do not interpret them as functions. However, in these cases too, we identify $S = N^{-1}$ and $W = E^{-1}$.

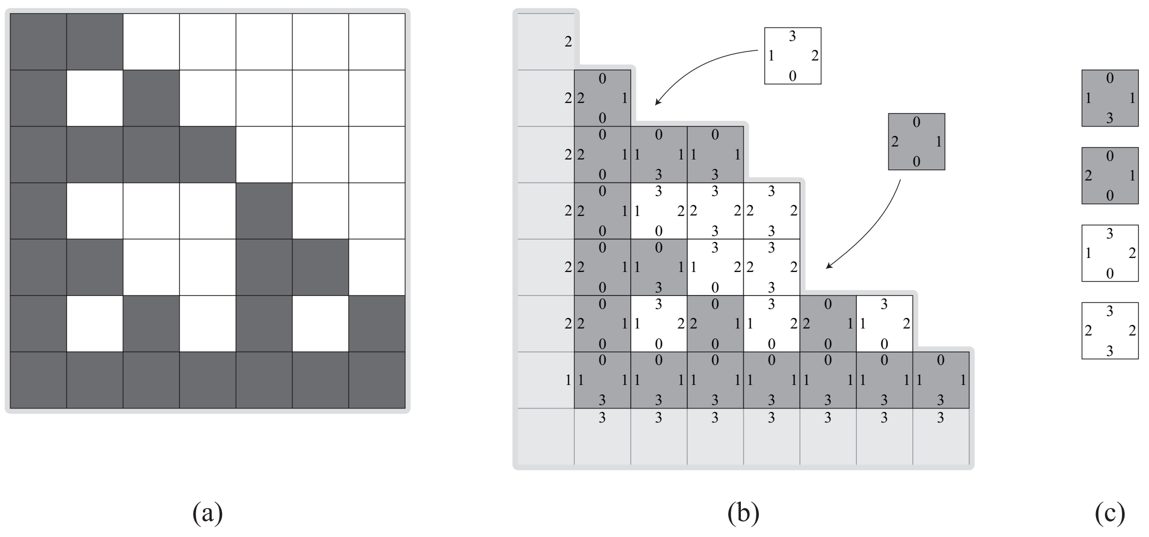


Figure 2: (a) A finite subset of the discrete Sierpinski triangle pattern. This 2-colouring of the set $[7] \times [7]$ defines an instance of the PATS problem. (b) Assembling the Sierpinski triangle pattern (see e.g. Winfree [22]) with a TAS that has an appropriate seed assembly and a (coloured) tile set shown in (c).

Lombardi [19] and proved to be so by Czeizler and Popa [20].[6] Without loss of generality, we consider only TASs $\mathscr{T}$ in which every tile type participates in some terminal assembly of $\mathscr{T}$.

As an illustration, using a 4-tile TAS from Winfree [22], we construct a 7×7 Sierpinski triangle pattern in Figure 2. We use natural numbers as glue labels in our figures.

In the literature, the seed assembly of a TAS is often taken to be a single seed tile [23] 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. Moreover, in some experimental designs for DNA tile assembly systems, such as that by Fujibayashi et al. [29], the implementation of seed structures by the DNA origami technique [15] allows the creation of such complete boundary conditions in a natural way.

Due to constraint *P1* the self-assembly process proceeds in a uniform manner directed from south-west to north-east. This paves the way for a simple characterisation of deterministic TASs in the context of the PATS problem.

Proposition 1. *Solutions $\mathscr{T} = (T, \mathcal{S}, s, 2)$ of the PATS problem are deterministic precisely when for each pair of glue types $(\sigma_1, \sigma_2) \in \Sigma^2$ there is at most one tile type $t \in T$ such that $\sigma_S(t) = \sigma_1$ and $\sigma_W(t) = \sigma_2$.*

A simple observation reduces the work needed in finding minimal solutions of the PATS problem.

Lemma 2. *The minimal solutions of the PATS problem are deterministic TASs.*

Proof. For the sake of contradiction, suppose that $\mathscr{N} = (T, \mathcal{S}, s, 2)$ is a minimal solution to a PATS problem instance and that $\mathscr{N}$ is not deterministic. By the above proposition, let tiles $t_1, t_2 \in T$ be such that $\sigma_S(t_1) = \sigma_S(t_2)$ and $\sigma_W(t_1) = \sigma_W(t_2)$. Consider the simplified TAS $\mathscr{N}' = (T \smallsetminus \{t_2\}, \mathcal{S}, s, 2)$. We show that this, too, is a solution to the PATS problem, which violates the minimality of $|T|$.

Suppose $\mathcal{A} \in \operatorname{Term} \mathscr{N}'$. If $\mathcal{A} \notin \operatorname{Term} \mathscr{N}$, then some $t \in T$ can be used to extend $\mathcal{A}$ in $\mathscr{N}$. If $t \in T \smallsetminus \{t_2\}$, then t could be used to extend $\mathcal{A}$ in $\mathscr{N}'$, so we must have $t = t_2$. But since new tiles are always attached by binding to south and west sides of the tile, $\mathcal{A}$ could then be extended by t_1 in $\mathscr{N}'$. Thus, we conclude that

[6]An improvement of the result to use only a constant number of tile colours is due to Seki [21].

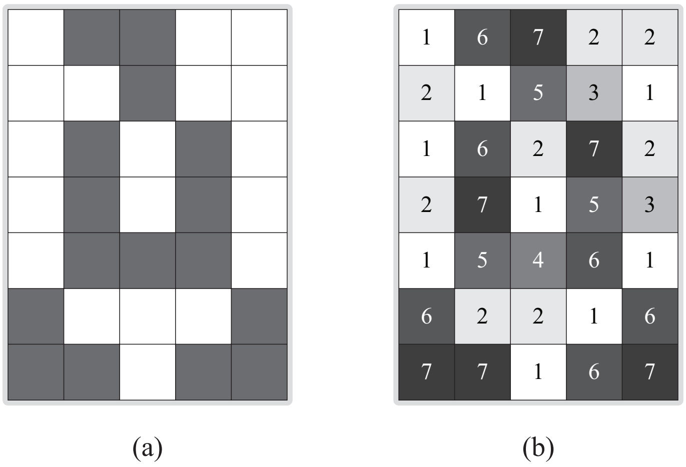


Figure 3: (a) Partition A. (b) A partition M that is a refinement of A with $|M| = 7$ partition classes.

$\mathcal{A} \in \operatorname{Term} \mathscr{N}$ and furthermore $\operatorname{Term} \mathscr{N}' \subseteq \operatorname{Term} \mathscr{N}$. This demonstrates that $\mathscr{N}'$ has property *P2*. The properties *P1* and *P3* can be readily seen to hold for $\mathscr{N}'$ as well. In terms of $|T|$ we have found a more optimal solution—and a contradiction. □

We consider only deterministic TASs in the sequel.

3. The search space of consistent tile sets

Let X be the family 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

$$\forall p' \in P' \colon \;\; \exists p \in P \colon \;\; p' \subseteq p.$$

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

A colouring $c \colon [m] \times [n] \to [k]$ 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 $P(\mathscr{T})$ of $[m] \times [n]$, $P(\mathscr{T}) = \{\mathcal{A}^{-1}(t) \mid t \in \mathcal{A}([m] \times [n])\}$. Here, $|P(\mathscr{T})| = |T|$ in case all tiles in T are used in the terminal assembly. Now condition *P3* in the definition of PATS 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 $\mathscr{T}$ satisfying properties *P1* and *P2*. Hence the PATS problem can be rephrased using the family of partitions as the fundamental search space.

Proposition 3. *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.*

For example, the 2-coloured pattern in Figure 3(a) defines a 2-class partition A. The 7-class partition M in Figure 3(b) is a refinement of A ($A \sqsubseteq M$) and in fact, M is constructible (see Figure 4(b)) and corresponds to a minimal solution of the PATS problem instance defined by the pattern A.

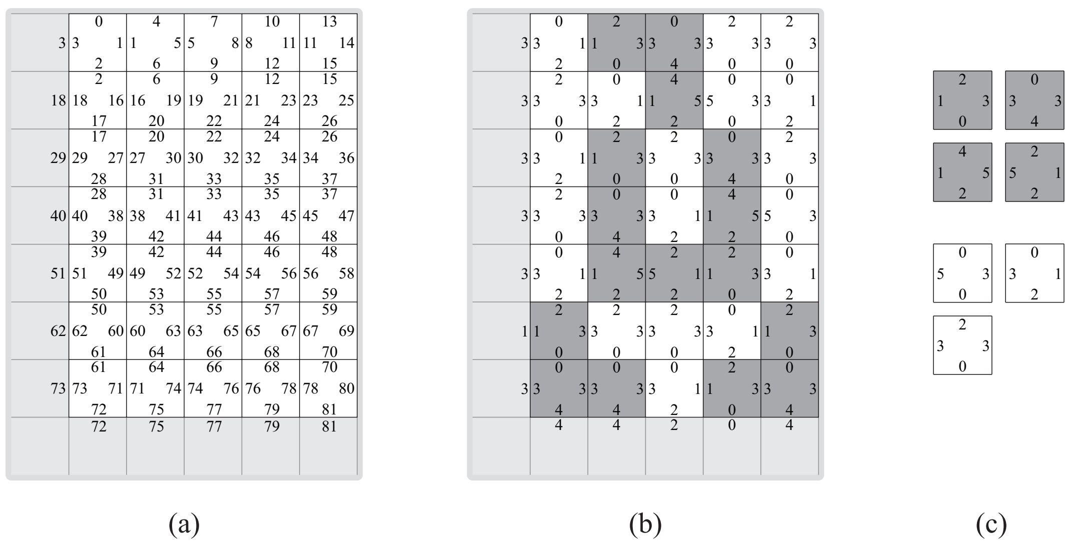


Figure 4: (a) A MGTA for the constructible initial partition I (with a seed assembly in place). (b) Finished assembly for the pattern from Figure 3(a). The tile set to construct this assembly is given in (c).

3.1. Determining constructibility

In this section, we give an algorithm for deciding the constructibility of a given partition in polynomial time. To do this, we use the concept of *most general* (or least constraining) *tile assignments*. For simplicity, we assume the set of glue labels Σ to be infinite.

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

A1. When every position in $[m] \times [n]$ is assigned a tile type according to f, any two adjacent positions agree on the glue type of the side between them.

A2. For all assignments $g\colon P \to \Sigma^4$ satisfying A1 we have[7]

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

for all $(p_1, D_1), (p_2, D_2) \in P \times \mathcal{D}$.

To demonstrate this concept, we present a most general tile assignment $f\colon I \to \Sigma^4$ for the *initial partition* $I = \{\{a\} \mid a \in [m] \times [n]\}$ in Figure 4(a) and an MGTA for the partition of Figure 3(b) in Figure 4(b).

Given a partition $P \in X$ and a function $f\colon P \to \Sigma^4$, we say that $g\colon P \to \Sigma^4$ 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. We start with a function $f_0\colon P \to \Sigma^4$ that assigns to each tile edge a unique glue type, or in other words, a function f_0 such that the mapping $(p, D) \mapsto f_0(p)_D$ is injective. Next, we go through all pairs of adjacent positions in $[m] \times [n]$ in some order and require their matching sides to have the same glue type by merging the corresponding glues. This process generates a sequence of functions $f_0, f_1, f_2, \ldots, f_N = f$ and terminates after $N \leq 2mn$ steps.

[7]To shorten the notation, we write $f(p)_D$ instead of $\sigma_D(f(p))$.

Lemma 4. *The above algorithm generates a most general tile assignment.*

Proof. By the end, we are left with a function f that satisfies property *A1* by construction. To see why property *A2* is satisfied, we again use the language of partitions.

Any tile assignment on P gives rise to a set of equivalence classes (or a partition) on $P \times \mathcal{D}$: class-direction pairs that are assigned the same glue type reside in the same equivalence class. The initial assignment f_0 gives each class-direction pair a unique glue type, and thus corresponds to the initial partition $J = \{\{a\} \mid a \in P \times \mathcal{D}\}$. In the algorithm, any glue merging operation corresponds to the combining of two equivalence classes.

The algorithm goes through a list of pairs $\{\{a_i, b_i\}\}_{i=0}^{N-1}$ of elements from $P \times \mathcal{D}$ that are required to have the same glue type. In this way, the list records necessary conditions for property *A1* to hold. This is to say that every tile assignment satisfying *A1* has to correspond to a partition of $P \times \mathcal{D}$ that is coarser than each of the partitions in $\mathcal{L} = \{J[a_i, b_i]\}_{i=0}^{N-1}$, where $J[a, b]$ is the partition obtained from the initial partition by combining classes a and b. Since the set $(P \times \mathcal{D}, \sqsubseteq)$ is a lattice, there exists a unique greatest lower bound $\inf \mathcal{L}$ of the partitions in $\mathcal{L}$. This is exactly the partition that the algorithm calculates in the form of the assignment f. As a greatest lower bound, $\inf \mathcal{L}$ is finer than any partition corresponding to an assignment satisfying *A1*, but this is precisely the requirement for condition *A2*. □

The above analysis also gives the following.

Corollary 5. *For a given partition of* $[m] \times [n]$*, MGTAs are unique up to relabelling of the glue types.*

Thus, for each partition $P \in X$, we take *the MGTA for P* to be some canonical representative from the class of MGTAs for P.

For efficiency purposes, it is worth mentioning that MGTAs can be generated iteratively: A partition $P \in X$ can be obtained by repeatedly combining partition classes starting from the initial partition I:

$$I = P_1 \sqsupseteq P_2 \sqsupseteq \cdots \sqsupseteq P_N = P.$$

As a base case, an MGTA for I can be computed by the above algorithm. An MGTA for each P_{i+1} can be computed from an MGTA for the previous partition P_i by just a small modification: Let an MGTA $f_i \colon P_i \to \Sigma^4$ be given for P_i and suppose P_{i+1} is obtained from P_i by combining classes $p_1, p_2 \in P_i$. Now, an MGTA f_{i+1} for P_{i+1} can be obtained from f_i by *merging tiles* $f_i(p_1)$ and $f_i(p_2)$, that is, merging the glue types on the four corresponding sides.

We now give the conditions for a partition to be constructible in terms of MGTAs.

Lemma 6. *A partition $P \in X$ is constructible iff the MGTA $f \colon P \to \Sigma^4$ for P is injective and the tile set $f(P)$ is deterministic in the sense of Proposition 1.*

Proof. "⇒": Let $P \in X$ be constructible and let the MGTA $f \colon P \to \Sigma^4$ for P be given. Let $\mathscr{T}$ be a deterministic TAS such that $P(\mathscr{T}) = P$. The uniquely produced assembly of $\mathscr{T}$ induces a tile assignment $g \colon P \to \Sigma^4$ that satisfies property *A1*. Now using property *A2* for the MGTA f we see that any violation of the injectivity of f or any violation of the determinism of the tile set $f(P)$ would imply such violations for g. But since g corresponds to a constructible partition, no violations can occur for g and thus none for f.

"⇐": Let $f \colon P \to \Sigma^4$ be an injective MGTA with deterministic tile set $f(P)$. Because $f(P)$ is deterministic, we can choose glue types for a seed assembly $\mathcal{S}$ so that the westernmost and southernmost tiles fall into place according to f in the self-assembly process. The TAS $\mathscr{T} = (f(P), \mathcal{S}, s, 2)$, with appropriate glue strengths s, then uniquely produces a terminal assembly that agrees with f on $[m] \times [n]$. This gives $P(\mathscr{T}) \sqsubseteq P$, but since f is injective, $|P| = |f(P)| = |P(\mathscr{T})|$ and so $P(\mathscr{T}) = P$. □

In order to understand the result of Lemma 6 better, let us consider the 2-coloured pattern in Figure 5(a), and associate to it the 2-class partition generated by the colours of the pattern. We can use the result of the previous lemma to show that this partition in not constructible. Indeed, if we consider the procedure for generating an MGTA for this partition, e.g. Figure 5(b) and (c), we obtain that the two tiles of the MGTA (one coloured white, the other black) must have the same glues on all corresponding positions. Hence the MGTA is not injective, nor deterministic in the sense of Proposition 1.

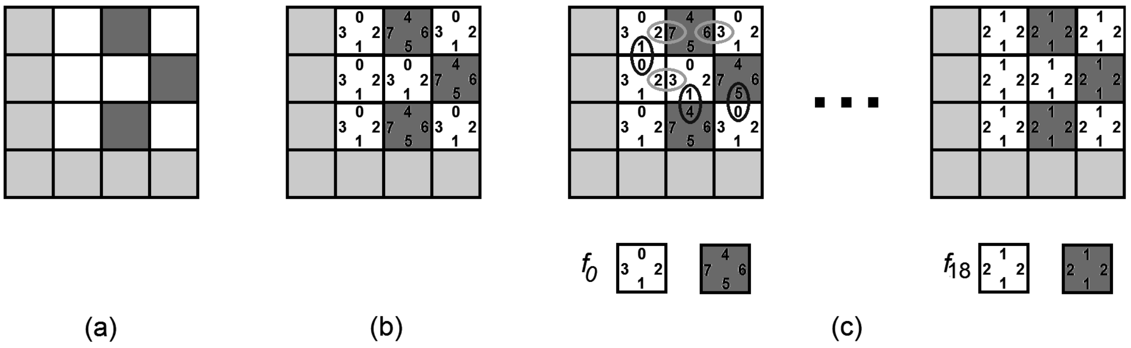


Figure 5: (a) A 2-coloured pattern. (b) The 2-class partition generated by the two colours and the initialisation of the procedure for finding an MGTA for this partition. (c) As a result of the MGTA generation procedure we obtain two tiles which have the same glues on all corresponding edges.

4. Complete search for minimal tile sets

We now extend the techniques of Ma and Lombardi [19] to obtain an exhaustive branch-and-bound search method to find minimal solutions to the PATS problem. We call this approach the *partition-search branch-and-bound* (PS-BB) algorithm. The idea of Ma and Lombardi [19] (following experimental work of Park et al. [30]) is to start with an *initial tile set* that consists of mn different tiles, one for each of the grid positions in $[m] \times [n]$. Their algorithm then proceeds to merge tile types in order to minimise $|T|$. We formalise this search process as an exhaustive search in the set of all partitions of the set $[m] \times [n]$. In the following, we let a PATS instance be given by a fixed k-coloured pattern $c\colon [m] \times [n] \to [k]$.

The PS-BB algorithm performs an exhaustive exploration of the lattice $(X, \sqsubseteq)$, searching for constructible partitions (see Figure 6). We start with the initial partition I that is always constructible. In the search, we maintain and incrementally update MGTAs for every partition we visit. First, we describe simple branching rules to obtain a rooted directed acyclic graph search structure and later give rules to prune this DAG to a node-disjoint search tree.

The root of the DAG is taken to be the initial partition I. For each partition $P \in X$ we next define the set $C(P) \subseteq X$ of children of P. Our algorithm always proceeds by combining classes of the partition currently being visited, so for each $P' \in C(P)$ we will have $P' \sqsubseteq P$. Say we visit a partition $P \in X$. We have two possibilities:

C1. *P is constructible:*

1. If P is not a refinement of the target pattern $P(c)$, that is if $P(c) \not\sqsubseteq P$, we can drop this branch of the search, since no possible descendant $P' \sqsubseteq P$ can be a refinement of $P(c)$ either.
2. In case $P(c) \sqsubseteq P$, we can use the MGTA for P to give a concrete solution to the PATS problem instance defined by the colouring c. To continue the search and to find further improved solutions we consider each pair of classes $\{p_1, p_2\} \subseteq P$ in turn and recursively visit the partition $P[p_1, p_2]$ where the two classes are combined. In fact, by the above analysis, it is sufficient to consider only pairs of the same colour. So, in this case,

$$C(P) = \{P[p_1, p_2] \mid p_1, p_2 \in P,\ p_1 \neq p_2,\ \exists k \in P(c)\colon p_1, p_2 \subseteq k\}.$$

C2. *P is not constructible:* In this case the MGTA f for P gives $f(p_1)_S = f(p_2)_S$ and $f(p_1)_W = f(p_2)_W$ for some classes $p_1 \neq p_2$. We continue the search from partition $P[p_1, p_2]$.

To guarantee that our algorithm finds the optimal solution in the case C2 above, we need the following.

Lemma 7. *Let $P \in X$ be a non-constructible partition, f the MGTA for P and $p_1, p_2 \in P$, $p_1 \neq p_2$, classes such that $f(p_1)_S = f(p_2)_S$ and $f(p_1)_W = f(p_2)_W$. For all constructible $C \sqsubseteq P$ we have $C \sqsubseteq P[p_1, p_2]$.*

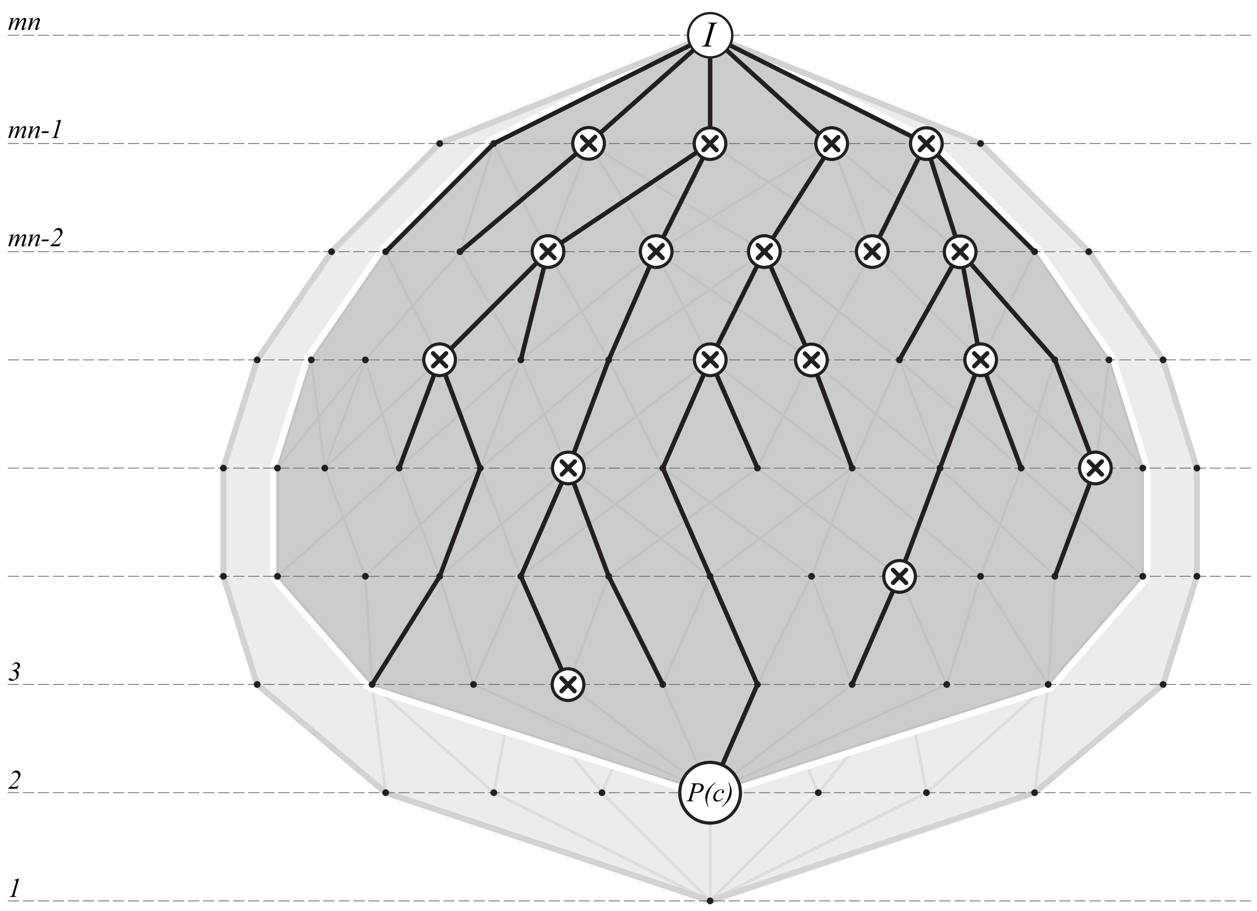


Figure 6: The search tree in the lattice $(X, \sqsubseteq)$. We start with the initial partition I of size $|I| = mn$. The partition $P(c)$ defines the PATS problem instance: We search for constructible partitions (drawn as crosses) in the sublattice (shaded with darker grey) consisting of those partitions that are refinements of $P(c)$. The search tree branches only at the constructible partitions and the tree branches are vertex-disjoint.

Proof. Let P, f, p_1 and p_2 be as in the statement of the lemma. Let $C \sqsubseteq P$ be a constructible partition and $g\colon C \to \Sigma^4$ the MGTA for C. Since C is coarser than P we can obtain from g a tile assignment $g'\colon P \to \Sigma^4$ such that $g'(p) = g(q)$, where for every $p \in P$, $q \in C$ is the unique class for which $p \subseteq q$. The assignment g' has property *A1* and so using *A2* for the MGTA f we get that

$$f(p_1)_S = f(p_2)_S \quad \& \quad f(p_1)_W = f(p_2)_W \qquad \Longrightarrow \qquad g'(p_1)_S = g'(p_2)_S \quad \& \quad g'(p_1)_W = g'(p_2)_W.$$

Now, since C is constructible, the identities $g(q_1)_S = g(q_2)_S$ and $g(q_1)_W = g(q_2)_W$ can not hold for any two different classes $q_1, q_2 \in C$. Looking at the definition of g', we conclude that $p_1 \subseteq q$ and $p_2 \subseteq q$ for some $q \in C$. This demonstrates $C \sqsubseteq P[p_1, p_2]$. □

4.1. Pruning the DAG to a search tree

Computational resources should be saved by not visiting any partition twice. To keep the branches in our search structure node-disjoint, we maintain a list of graphs that store restrictions on the choices the search can make.

For each partition $P \sqsupseteq P(c)$ we associate a family of undirected graphs $\{G_k^P\}_{k \in P(c)}$, one for each colour class of the pattern $P(c)$. Every class in P is represented by a vertex in the graph corresponding to the colour of the class. More formally, the vertex set $V(G_k^P)$ of the graph G_k^P is taken to be those classes $p \in P$

for which $p \subseteq k$. (So now, $\bigcup_{k \in P(c)} V(G_k^P) = P$.) An edge $\{p_1, p_2\} \in E(G_k^P)$ indicates that the classes p_1 and p_2 are not allowed ever to be combined in the search branch in question. When we start our search with the initial partition I, the edge sets are initially empty, $E(G_k^I) = \varnothing$. At each partition P, the graphs $\{G_k^P\}_{k \in P(c)}$ have been determined inductively and the graphs for those children $P' \in C(P)$ that we visit are defined as follows.

D1. *If P is constructible:* We choose some ordering $\{p_i, q_i\}$, $i = 1, \ldots, N$ of similarly coloured pairs of classes. Define $l_i \in P(c)$, $1 \leq i \leq N$ to be the colour of the pair $\{p_i, q_i\}$, so that $p_i, q_i \subseteq l_i$. Now, we visit a partition $P[p_i, q_i]$ if and only if $\{p_i, q_i\} \notin E(G_{l_i}^P)$. If we decide to visit a child partition $P' = P[p_j, q_j]$, we define the edge sets $\{E(G_k^{P'})\}_{k \in P(c)}$ as follows:

1. We start with the graphs $\{G_k^P\}_{k \in P(c)}$ and add the edges $\{p_i, q_i\}$ for all $1 \leq i < j$ to their corresponding graphs. Call the resulting graphs $\{G_k^\star\}_{k \in P(c)}$.
2. Finally, as we combine the classes p_j and q_j to obtain the partition $P[p_j, q_j]$, we merge the vertices p_j and q_j in the graph $G_{l_j}^\star$ (after merging, the neighbourhood of the new vertex $p_j \cup q_j$ is the union of the neighbourhoods for p_j and q_j in $G_{l_j}^\star$). The graphs $\{G_k^{P'}\}_{k \in P(c)}$ follow as a result.

D2. *If P is not constructible:* Here, the MGTA for P suggests a single child partition $P' = P[p_1, p_2]$ for some $p_1, p_2 \subseteq l \in P(c)$. If $\{p_1, p_2\} \in E(G_l^P)$, we terminate this branch of the search. Otherwise, we define the graphs $\{G_k^{P'}\}_{k \in P(c)}$ to be the graphs $\{G_k^P\}_{k \in P(c)}$, except that in $G_l^{P'}$ the vertices p_1 and p_2 are merged.

One can see that the outcome of this pruning process is a search tree that has node-disjoint branches and one in which every possible constructible partition is still guaranteed to be found. Figure 6 presents a sketch of the search tree.

Note that we are not usually interested in finding every constructible partition $P \in X$, but only in finding a minimal one (in terms of $|P|$). Next, we give an efficient method to lower-bound the partition sizes of a given search branch.

4.2. The bounding function

Given a root $P \in X$ of some subtree of the search tree, we ask: What is the smallest partition that can be found from this subtree? The nodes in the subtree rooted at P comprise those partitions $P' \sqsubseteq P$ that can be obtained from P by merging pairs of classes that are not forbidden by the graphs $\{G_k^P\}_{k \in P(c)}$. This merging process halts precisely when all the graphs $\{G_k^{P'}\}_{k \in P(c)}$ have been reduced into cliques. As is well known (and easy to see), the size of the smallest clique that a graph G can be turned into by merging non-adjacent vertices is given by the *chromatic number*[8] $\chi(G)$ of the graph G. This immediately gives the following.

Proposition 8. *For every $P' \sqsubseteq P$ in the subtree rooted at P and constrained by $\{G_k^P\}_{k \in P(c)}$, we have*

$$\sum_{k \in P(c)} \chi(G_k^P) \;\leq\; |P'|.$$

Determining the chromatic number of an arbitrary graph is an NP-hard problem. Fortunately, we can restrict our graphs to be of a special form: graphs that consist only of a clique and some isolated vertices. For these graphs, the chromatic numbers are given by the sizes of the cliques.

To see how to maintain graphs in this form, consider as a base case the initial partition I. Here, $E(G_k^I) = \varnothing$ for all $k \in P(c)$, so G_k^I is of our special form—it has a clique of size 1. For a general partition P, we go through the branching rules D1–D2.

[8]The chromatic number of a graph G is the smallest number of colours $\chi(G)$ needed to colour the vertices of G so that no two adjacent vertices share the same colour.

D1: *P is constructible:* Since we are allowed to choose an arbitrary ordering $\{p_i, q_i\}$, $i = 1, \ldots, N$, for the children $P[p_i, q_i]$, we design an ordering that preserves the special form of the graphs. For a graph G of our special form, let $K(G) \subseteq V(G)$ consist of those vertices that are part of the clique in G. In the algorithm, we first set $H_k = G_k^P$ for all $k \in P(c)$ and repeat the following process until every graph H_k is a complete clique.

1. Pick some colour $k \in P(c)$ and an isolated vertex $v \in V(H_k) \smallsetminus K(H_k)$.
2. Process the pairs $\{v, u\}$ for all $u \in K(H_k)$ in some order. By the end, update H_k to include all the edges $\{v, u\}$ that were just processed (the size of the clique in H_k increases by one).

A moment's inspection reveals that when the graphs G_k^P are of our special form, so are all of the derived graphs passed on to the children of P.

D2: *P is not constructible:* If the algorithm decides to continue the search from a partition $P' = P[p_1, p_2]$, for some $p_1, p_2 \subseteq l \in P(c)$, we have $\{p_1, p_2\} \notin E(G_l^P)$. This means that either $p_1, p_2 \in V(G_l^P) \smallsetminus K(G_l^P)$, in which case we are merging two isolated vertices, or one of p_1 and p_2 is part of the clique $K(G_l^P)$, in which case we merge an isolated vertex to the clique. In both cases, we maintain the special form in the graphs $\{G_k^{P'}\}_{k \in P(c)}$.

4.3. Traversing the search tree

When running a branch-and-bound algorithm we maintain a "current best solution" discovered so far as a global variable. This solution gives an upper bound for the minimal value of the tile set size and can be used to prune such search branches that are guaranteed (by the bounding function) to only yield solutions worse than the current best. There are two general strategies to traverse a branch-and-bound search tree: *Depth-First Search* and *Best-First Search* [31]. Our description of the search tree for the lattice X is general enough to allow either of these strategies to be used in the actual implementation of the algorithm. In the following section we give performance data on our DFS implementation of the PS-BB algorithm.

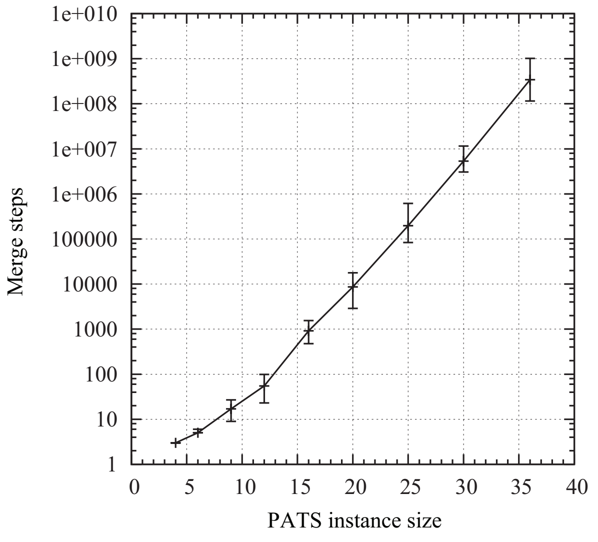
4.4. Results

The running time of the PS-BB algorithm is proportional—up to a polynomial factor—to the number of partitions the algorithm visits. Hence, we measure the running time in terms of the number of merge operations performed in the search. Figure 7(a) presents the number of such merge operations in order to find a minimal solution for random 2-coloured instances of the PATS problem. The algorithm was executed for instance sizes $2 \times 2, 2 \times 3, 3 \times 3, \ldots, 5 \times 6$ and 6×6; the 20th and 80th percentiles are shown alongside the median of 21 separate runs for each instance size. For the limiting case 6×6, the algorithm spent on the order of two hours of (median) computing time on a 2.61 GHz AMD processor.

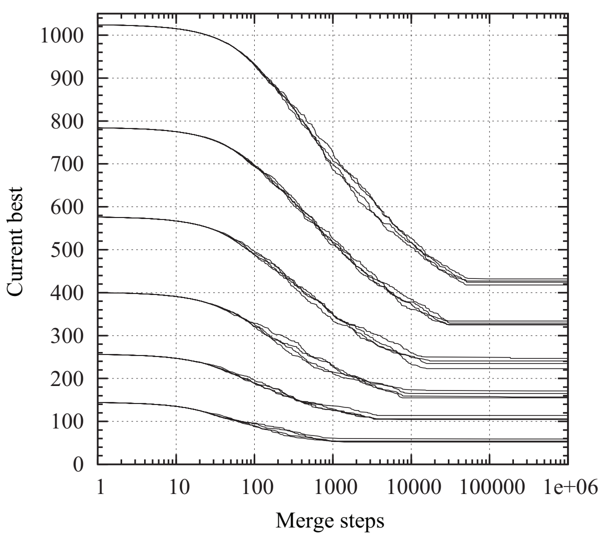
Even though branch-and-bound search is an exact method, it can be used to find approximate solutions by running it for a suitable length of time. Figure 7(b) illustrates how the best solution found up to a point develops as increasingly many steps of the algorithm are run. The figure provides data on random 2-coloured instances of sizes $12 \times 12, 16 \times 16, 20 \times 20, \ldots, 32 \times 32$. Because we begin our search from the initial partition, the best solution at the first step is precisely equal to the instance size. For each size, several different patterns were used. The algorithm was cut off after 10^6 steps. By this time, an approximate reduction of 58% in the size of the tile set was achieved (cf. a reduction of 43.5% in Ma and Lombardi [19]).

Next, we consider two well known examples of structured patterns: the discrete Sierpinski triangle and the binary counter (see Figures 9(a) and 9(b) for 32×32 instances of both patterns). A tile set of size 4 is optimal for both of these patterns, see e.g. Winfree [22] or Rothemund and Winfree [23]. First, for the Sierpinski triangle pattern, we get a tile set reduction of well over 90% (cf. 45% in Ma and Lombardi [19]) in Figure 8(a). We used the same cutoff threshold and instance sizes as in Figure 7(b).

Our description of the PS-BB algorithm leaves some room for randomisation in deciding which search branch the DFS search is to explore next. This randomisation does not seem to affect the search dramatically when considering the Sierpinski triangle pattern—the separate single runs in Figure 8(a) are representative of an average randomised run. By contrast, for the binary counter pattern, randomised runs for a single



(a)



(b)

Figure 7: (a) Running time of the PS-BB algorithm (as measured by the number of merge operations) to solve random 2-coloured near-square-shaped instances of the PATS problem. (b) Evolution of the tile set size of the "current best solution" for the PATS problem for random 2-coloured instances of sizes from 12×12 up to 32×32.

instance size do make a difference. Figure 8(b) depicts several separate runs for instance size 32×32. Here, each run brings about a reduction in solution size that oscillates between a reduction achieved on a random 2-coloured instance (Figure 7(b)) and a reduction achieved on the Sierpinski instance (Figure 8(a)). This suggests that, as is characteristic of DFS traversal, restarting the algorithm with different random seeds may help with large instances that have small optimal solutions. We explore this opportunity for efficiency improvement further in connection to the algorithm PS-H presented in the next section.

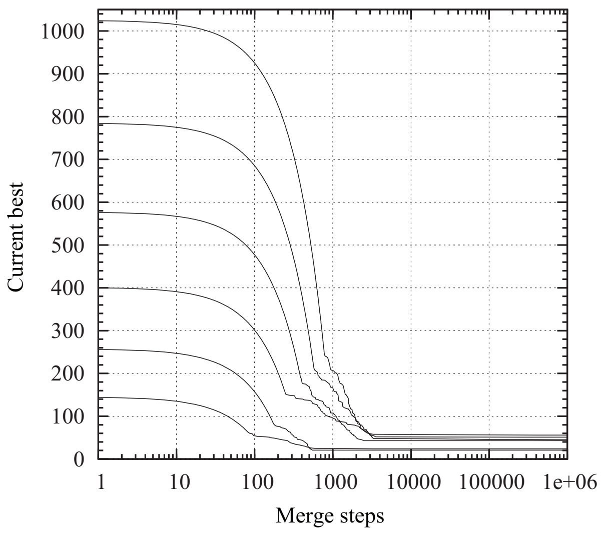
5. Heuristically guided search for small tile sets

5.1. The PS-H algorithm scheme

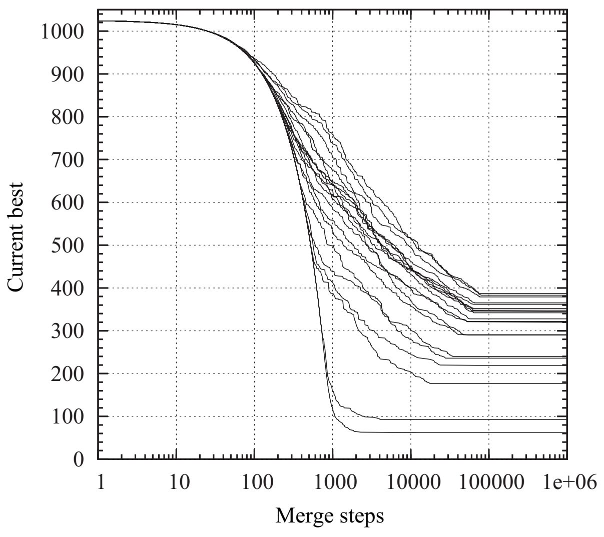
The PS-BB algorithm utilises effective pruning methods to reduce the search space. Even though it offers significant reduction in the size of tile sets compared to earlier approaches, it is in most cases still too slow for patterns of practical size. Often it is not important to find a provably minimal solution, but to find a reasonably small solution in a reasonable amount of time. To address this objective, we present in the following a modification of the basic PS-BB algorithm with a number of search-guiding heuristics. We call this approach the *partition-search with heuristics* (PS-H) algorithm scheme.

Whereas the pruning methods of the PS-BB algorithm try to reduce the size of the search space in a "balanced" way, the PS-H algorithm attempts to "greedily" optimise the order in which the coarsenings of a partition are explored, in the hope of being directly led 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 partition classes other than those which are combined. This can be achieved by preferring 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.



(a)



(b)

Figure 8: Evolution of the "current best solution" of the PS-BB algorithm for (a) the Sierpinski triangle pattern and for (b) the binary counter pattern. The lines represent (a) single runs for instance sizes from 12×12 up to 32×32 and (b) several runs for instance size 32×32. Randomisation in the DFS has a clear effect on the performance of the algorithm in the case of the binary counter pattern, but not in the case of the Sierpinski triangle pattern.

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.

We define the concept of the *number of common glues* formally as follows.

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 $G \colon P \times P \to \{0, 1, 2, 3, 4\}$,*

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

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

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. In the PS-H algorithm, we choose the ordering using the following heuristics. First form the set

$$H \coloneqq \{\{p, q\} \mid p, q \in P, \;\; p \neq q, \;\; \exists r \in P(c) \colon \;\; p, q \subseteq r\}$$

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

H1. Set $K \coloneqq H$.

H2. Maximise the number of common glues:

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

H3. Maximise the size of the larger class:

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

H4. Maximise the size of the smaller class:

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

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

H6. Remove $\{p,q\}$ from H:

$$H \coloneqq H \setminus \{\{p,q\}\}.$$

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 heuristics 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 executions of the algorithm in parallel, or equivalently, restarting the search several times with a different random seed. The notation PS-H$_n$ denotes the heuristic partition search algorithm with n parallel search threads. The solution found by the PS-H$_n$ algorithm is the smallest solution found by any of the n parallel threads.

5.2. Results

In this section, we present results on the performance of the PS-H$_n$ algorithm for $n = 1, 2, 4, 8, 16, 32$ and compare it to the previous PS-BB algorithm. We consider several different finite 2-coloured input patterns, two of which were analysed also previously using the PS-BB algorithm: the discrete Sierpinski triangles of sizes 32×32 (Figure 9(a)) and 64×64, and the binary counter of size 32×32 (Figure 9(b)). Furthermore, we introduce a 2-coloured "tree" pattern of size 23×23 (Figure 9(c)) as well as a 15-coloured pattern of size 20×10 based on a CMOS full adder design (Figure 9(d)).[9] While the Sierpinski triangle and binary counter patterns are known to have minimal solutions of 4 tiles, the minimal solutions for the tree pattern and the full adder pattern are unknown. The experiments were conducted on a high performance computing cluster equipped with 2.6 GHz AMD Opteron 2435 processors and Scientific Linux 6 operating system.

Figure 10 presents the evolution of the "current best solution" as a function of time for the (a) 32×32 and (c) 64×64 Sierpinski triangle patterns. To allow fair comparison, Figures 10(b) and 10(d) present the same data with respect to the total processing time taken by all the executions that run in parallel. The experiments were repeated 21 times and the median of the results is depicted. In 37% of all the individual runs[10] conducted, the PS-H algorithm was able to find the optimal 4-tile solution for the 32×32 Sierpinski triangle pattern in less than 30 seconds. A similar percentage for the 64×64 Sierpinski triangle 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 have 42 tiles, reached after $1.4 \cdot 10^6$ merge steps, and 95 tiles, reached after $5.9 \cdot 10^6$ merge steps.

In Figure 11 we present the corresponding results for the 32×32 binary counter and 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 12(a)–12(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.

[9] For an explanation of the notation used in Figure 9(d), see Appendix A.

[10] In total there were $1 \cdot 21 + 2 \cdot 21 + 4 \cdot 21 + \cdots + 32 \cdot 21 = 1323$ runs for each input pattern.

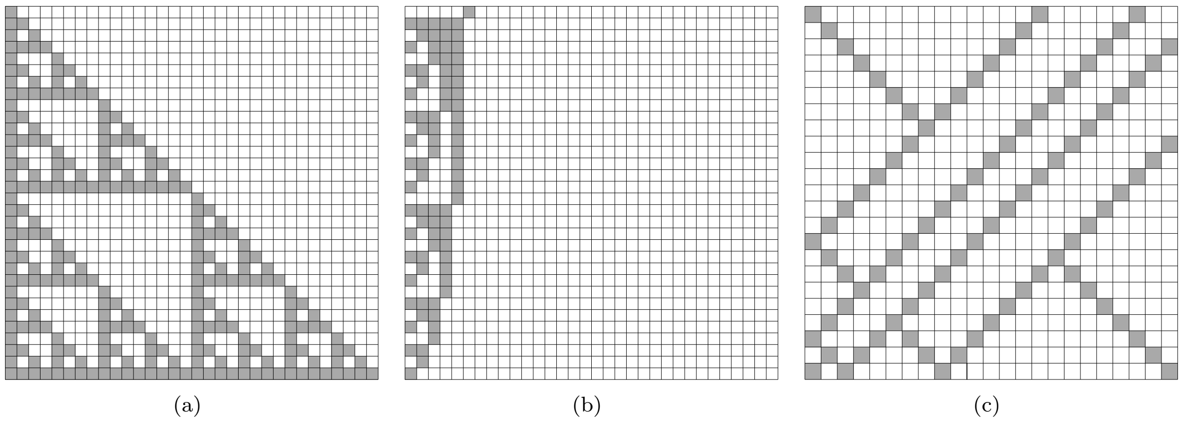
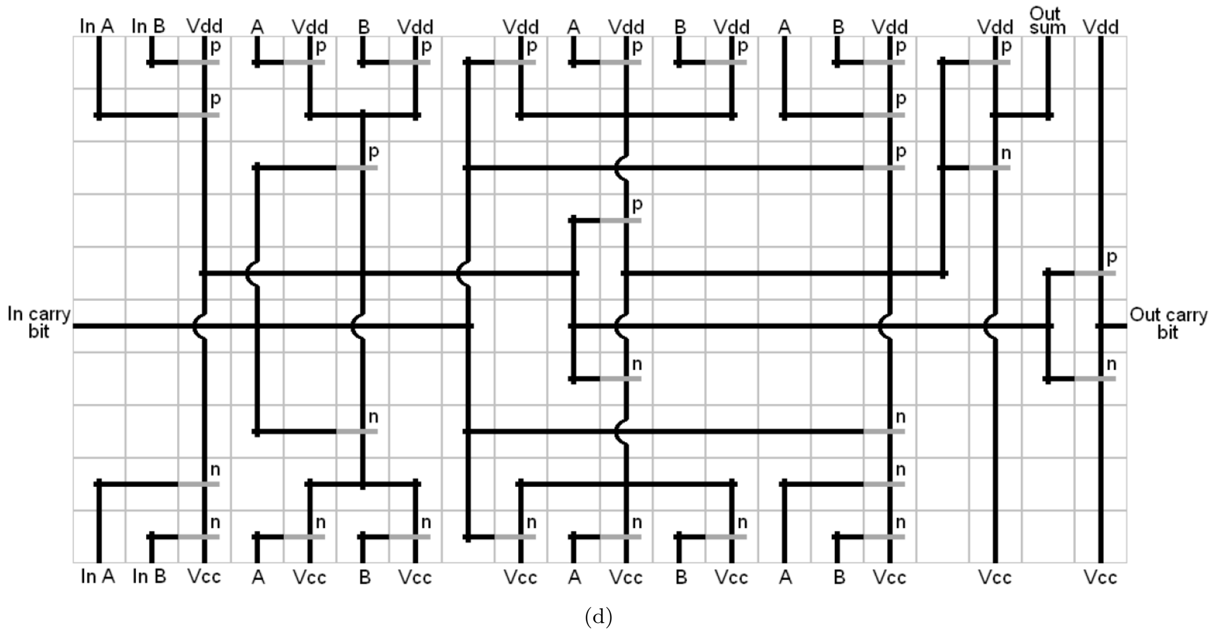



Figure 9: (a) The 32 × 32 Sierpinski triangle pattern. (b) The 32 × 32 binary counter pattern. (c) The 23 × 23 "tree" pattern. (d) A CMOS full adder design that induces a 15-colour 20 × 10 pattern.

6. Answer set programming for minimal tile sets

6.1. An ASP model for PATS

Answer Set Programming (ASP) [26] 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) of the logic program. The ASP paradigm can be applied also to the PATS problem. In the following we give a brief description on how to transform the PATS problem to an ASP program using a modelling language that is accepted by ASP grounders such as LPARSE [32] or GRINGO [33].

First, we define a 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 are introduced to associate a tile type with each position 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. Next, we use basic rules to make the glues of every pair of adjacent tiles match and

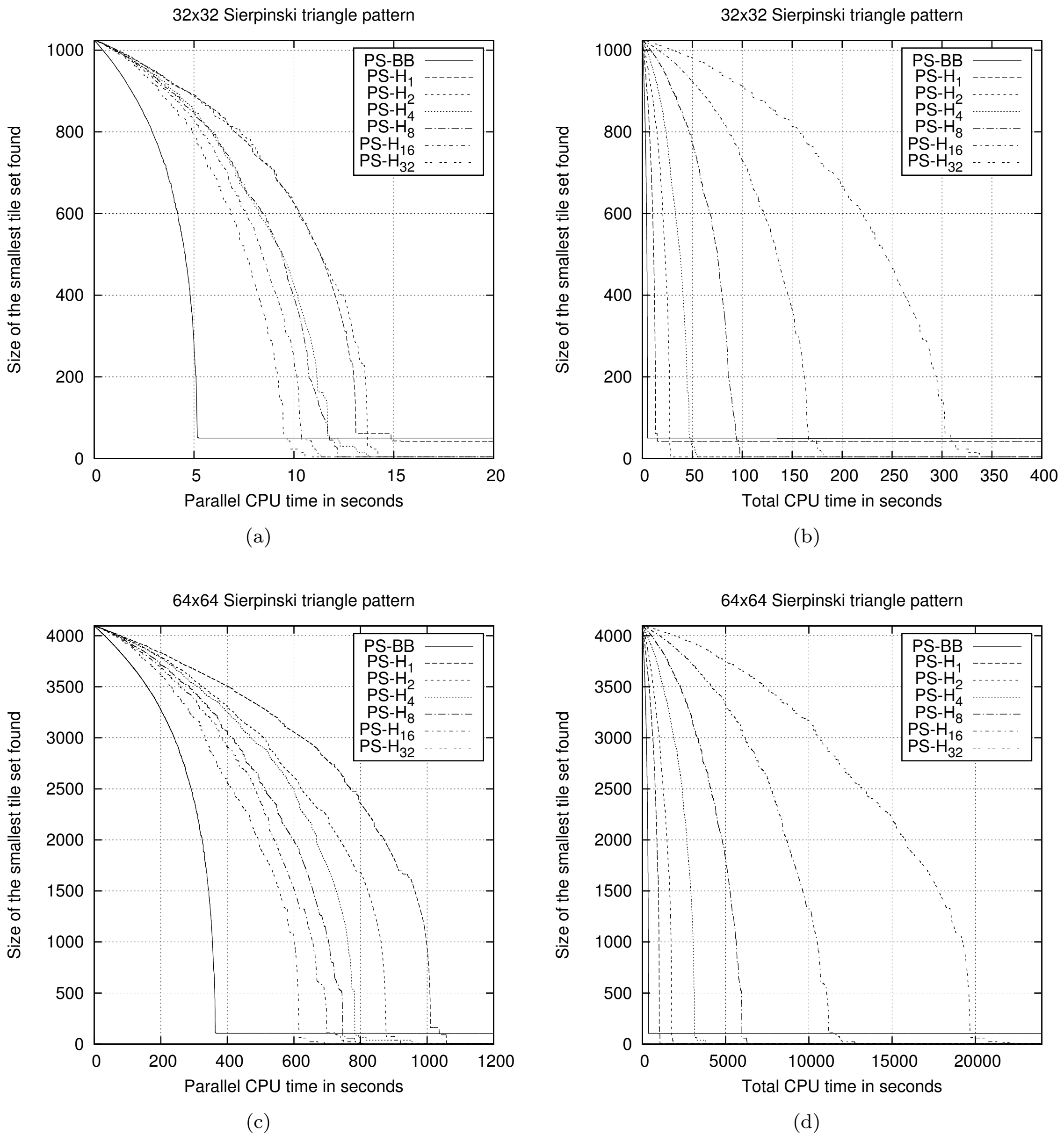
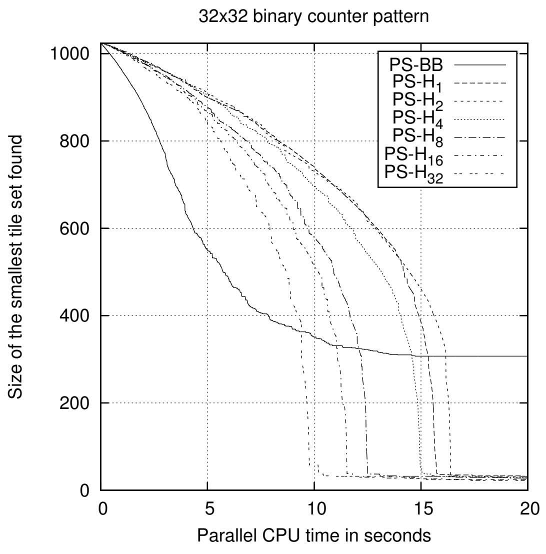


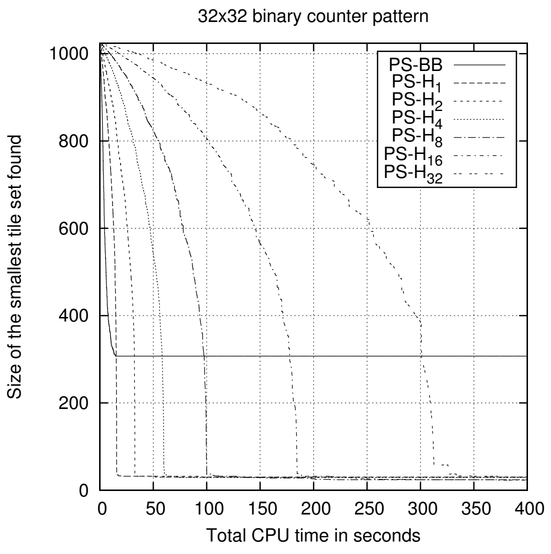
Figure 10: Evolution of the smallest tile set found for the 32×32 and 64×64 Sierpinski triangle patterns as a function of time. The time axes measure (a), (c) CPU time and (b), (d) CPU time multiplied by the number of parallel executions.

to make the tile system deterministic, i.e. to ensure that every tile type has a unique pair of glues on its W and S edges. Finally, we compile the target pattern to a set of rules that associate every position of the grid with the desired colour.

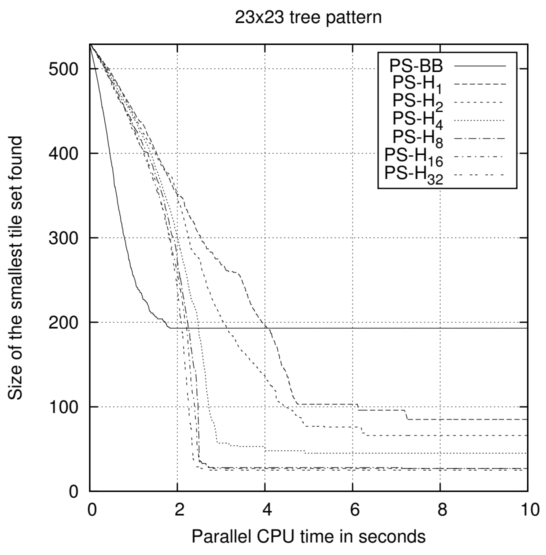
The above-described program is given to a grounder, which computes an equivalent variable-free program. The variable-free program is forwarded to an answer set solver, which then outputs a tile type for each position of the grid, given that such a solution exists. We run the programs repeatedly and increment the number of available tile and glue types, until a solution is found.



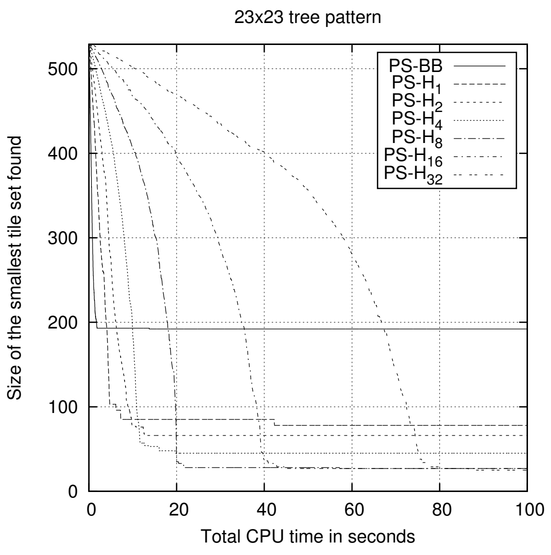
(a)



(b)



(c)

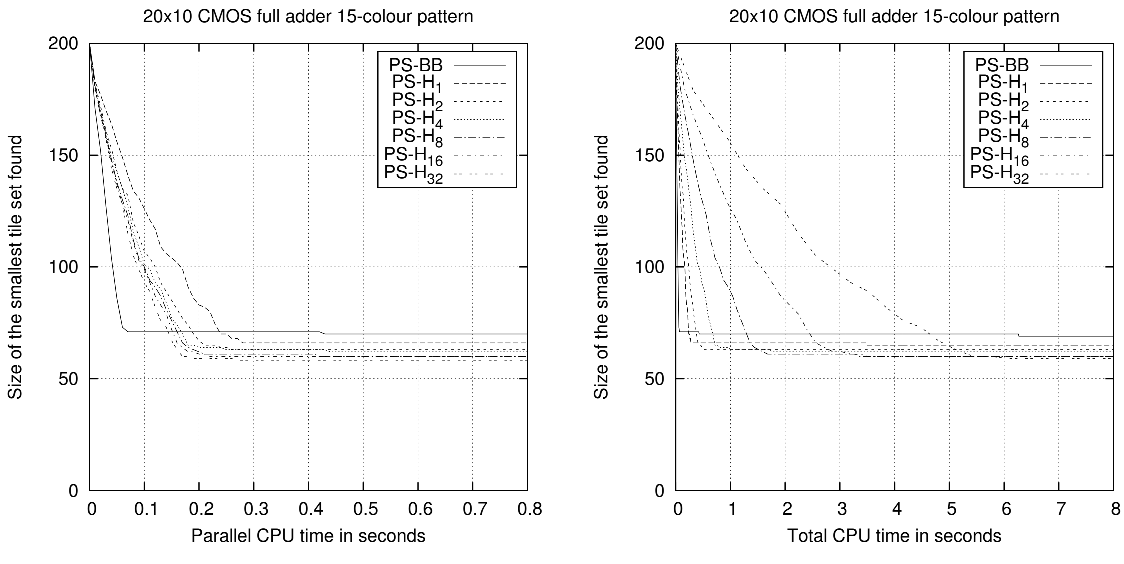


(d)

Figure 11: Evolution of the smallest tile set found for the 32×32 binary counter and 23×23 tree patterns as a function of time. The time axes measure (a), (c) CPU time and (b), (d) CPU time multiplied by the number of parallel executions.

6.2. Results

We used grounder GRINGO 3.0.5 [33] and answer set solver CLASP 2.1.3 [34] with default settings to run our experiments. A traditional solver, SMODELS [35], was also considered, but CLASP proved to be significantly faster in solving instances of the PATS problem. We consider two patterns having a minimal solution of 4 tiles: the Sierpinski triangle and binary counter patterns. The programs were executed for patterns of sizes $8 \times 8, 16 \times 16, \ldots, 256 \times 256$. We repeated the experiments 21 times with different random seeds and the median running time is presented in Figure 13(a) for the Sierpinski triangle pattern and in Figure 13(b) for the binary counter pattern. The results include the running time of both the grounder and



(a) (b)

Figure 12: Evolution of the smallest tile set found for the 20×10 full adder pattern as a function of time. The time axes measure (a) CPU time and (b) CPU time multiplied by the number of parallel executions.

the solver as well as all the incremental steps needed until a solution is found. We were able to find the minimal solution for both the 256×256 Sierpinski triangle pattern and the 256×256 binary counter pattern in approximately 31 minutes of (median) running time. The results were obtained on the same computing cluster as the results in Section 5.2.

Based on the above results, the ASP approach performs very well when considering patterns with a small optimal solution. However, the running time seems to increase dramatically with patterns that have a larger optimal solution. Indeed, we were not able to find solutions for the 23×23 tree pattern or the 20×10 CMOS full adder pattern using the ASP approach.

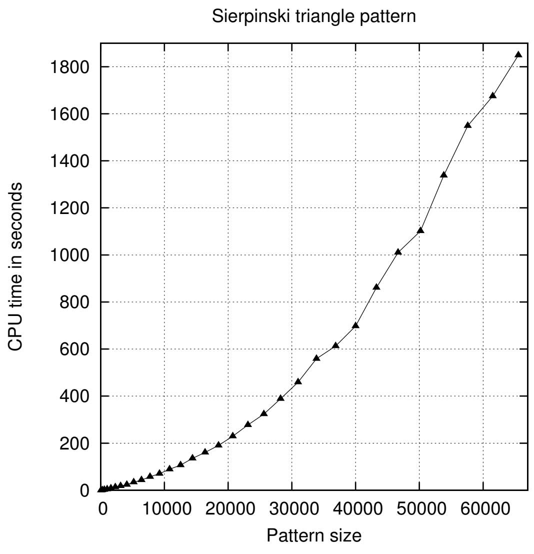
7. The reliability of tile sets

7.1. The kinetic Tile Assembly Model

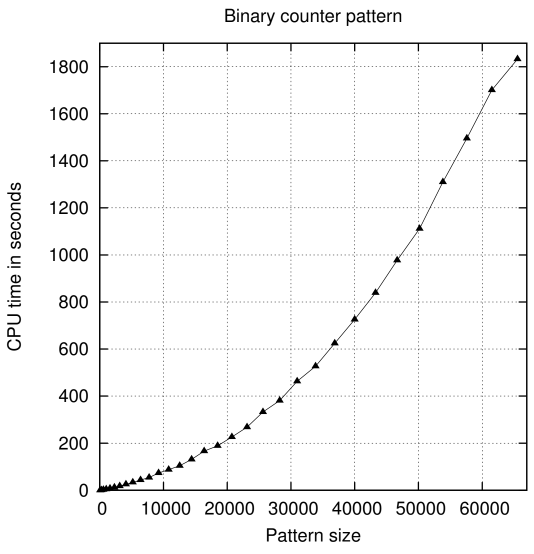
In the following, we utilise the kinetic Tile Assembly Model (kTAM) to assess the reliability of various tile sets generated by the PS-BB and PS-H algorithms. The kTAM was introduced by Winfree [22] as a kinetic counterpart of the aTAM. Several variants of the kTAM exist [36, 37]. However, the main elements 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 (forward reaction) and *dissociation* (reverse reaction), see e.g. Figure 14.[11] 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 r_f 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 rate $r_{r,b}$, $b \in \{0, \ldots, 4\}$, 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).

[11]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. Schulman and Winfree [37].



(a)



(b)

Figure 13: Running time of GRINGO and CLASP for the minimal solutions of the (a) Sierpinski triangle and (b) binary counter patterns as a function of pattern size.

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

$$r_f = k_f[t], \quad \text{(in /sec)}$$

where $[t]$ is the concentration in solution of free tiles of type t and k_f is a temperature dependent parameter. In the 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 reaction), for a tile which is connected to the assembly by a total bond strength b, the rate constant $r_{r,b}$ is given by the formula

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

where ΔG_b^o is the standard free energy needed to break b bonds. In the case of DX tiles, as the glues of the tiles are implemented using 5-base long single-stranded DNA molecules, ΔG_b^o can be estimated using the nearest-neighbour model [38] as

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

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 and the second is dependent on the assembly temperature:

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

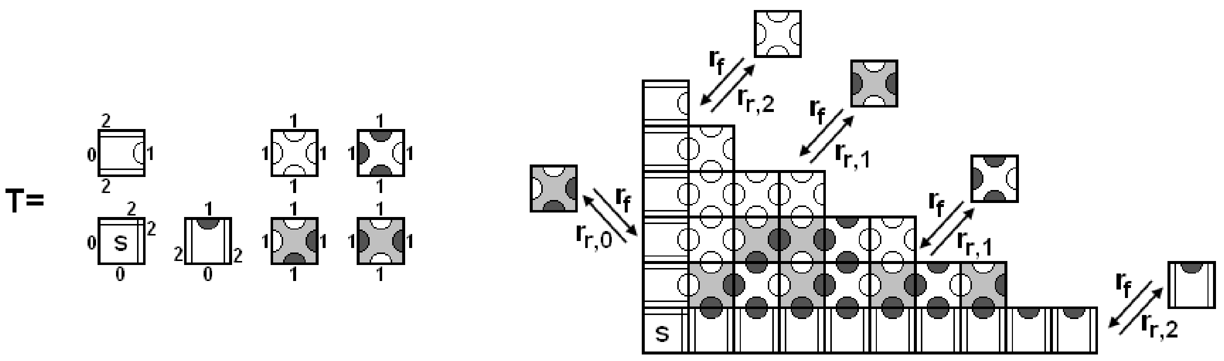


Figure 14: Possible association and dissociation reactions in the kinetic Tile Assembly Model. The rate of all the association reactions is identical; the rates of the dissociation reactions depend on the total strength of the bonds connecting a tile to the assembly.

where, in the 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 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.

7.2. Computing the reliability of a tile set

By choosing appropriate physical conditions, the probability of errors in the assembly process can be made arbitrarily low, at the cost of reducing the assembly rate [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 the following, we present a method for computing the reliability of a tile set, based on Winfree's analysis of the kTAM [22], and the notion of *kinetic trapping* introduced within.

We call the W and S 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 $M^1_{i,j}$ and $M^2_{i,j}$ be the number of tile types having one mismatching and two mismatching input glues, 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 $|T| = 1 + M^1_{i,j} + M^2_{i,j}$ for any $(i,j) \in [m] \times [n]$. Given that a site has the 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. To each of these cases we can associate an "off-rate" for the system to exit its current state: (E) An empty site, with off-rate $|T|r_f$. (C) The correct tile, with off-rate $r_{r,2}$. (A) A tile with one match, with off-rate $r_{r,1}$. (I) A tile with no matches, with off-rate $r_{r,0}$. 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 $r^* = r_f - r_{r,2}$. Figure 15 describes the dynamics of the system. Let $p_S(t)$ denote the probability of the site being in state S after t seconds for all $S \in \{\mathrm{E}, \mathrm{C}, \mathrm{A}, \mathrm{I}, \mathrm{FC}, \mathrm{FI}\}$. To compute the frozen distribution, we write the rate equations for the model of kinetic trapping from Figure 15 as follows:[12]

[12]The notation $\dot{\mathbf{p}}(x)$ is used to denote the derivative of $\mathbf{p}$ with respect to time.

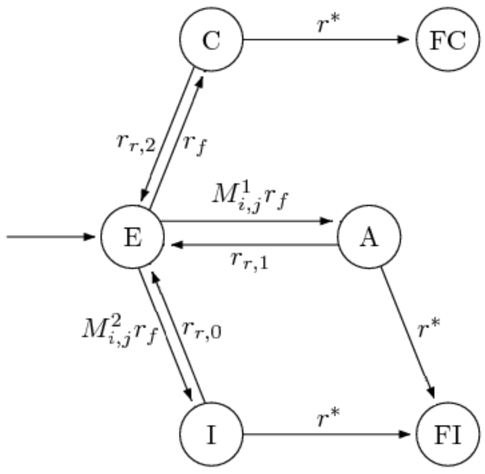


Figure 15: The dynamics of the kinetic trapping model.

$$M\mathbf{p}(t) := \begin{bmatrix} -|T|r_f & r_{r,2} & r_{r,1} & r_{r,0} & 0 & 0 \\ r_f & -r_{r,2} - r^* & 0 & 0 & 0 & 0 \\ M^1_{i,j} r_f & 0 & -r_{r,1} - r^* & 0 & 0 & 0 \\ M^2_{i,j} r_f & 0 & 0 & -r_{r,0} - r^* & 0 & 0 \\ 0 & r^* & 0 & 0 & 0 & 0 \\ 0 & 0 & r^* & r^* & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{\text{E}}(t) \\ p_{\text{C}}(t) \\ p_{\text{A}}(t) \\ p_{\text{I}}(t) \\ p_{\text{FC}}(t) \\ p_{\text{FI}}(t) \end{bmatrix} = \dot{\mathbf{p}}(t),$$

where $\mathbf{p}(0) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$. To compute the steady-state probability of the site being frozen with the correct tile, i.e. $p_{\text{FC}}(\infty)$, we make use of the steady state of the related flow problem [22]:[13]

$$M\mathbf{p}(\infty) = \begin{bmatrix} 1 & 0 & 0 & 0 & p_{\text{FC}}(\infty) & p_{\text{FI}}(\infty) \end{bmatrix}^T = \dot{\mathbf{p}}(\infty),$$

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

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

where $C_{i,j}$ denotes the event of the correct tile being frozen at 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 at site (i_k, j_k). Due to the fact that the assembly process of the tile systems considered here proceeds uniformly from south-west to north-east, we have that $\{(i_k - 1, j_k), (i_k, j_k - 1)\} \subseteq \{a_1, a_2, \ldots, a_{k-1}\}$ 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:

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

[13]By the definition of the kinetic trapping model [22], it is assumed that a unit amount of material is supplied into state E of the system at any time point.

We have computed the probability in terms of G_{mc} and G_{se}. Given the desired assembly rate, 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 rate for an $m \times n$ pattern is approximately $r^* = \frac{\sqrt{m^2+n^2}}{t}$. In order to simplify the computations, we use the approximation

$$\Pr(C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) = \frac{\frac{1}{r^*+r_{r,2}}}{\frac{1}{r^*+r_{r,2}} + \frac{M^1_{i,j}}{r^*+r_{r,1}} + \frac{M^2_{i,j}}{r^*+r_{r,0}}} \approx \frac{1}{1 + M^1_{i,j}\frac{r^*+r_{r,2}}{r^*+r_{r,1}}}.$$

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

$$\Pr(\neg C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) \approx M^1_{i,j}\frac{r^* + r_{r,2}}{r^* + r_{r,1}} \approx M^1_{i,j} e^{-(G_{mc}-G_{se})} =: M^1_{i,j} e^{-\triangle G}.$$

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 differentiation:

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

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

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

7.3. Results

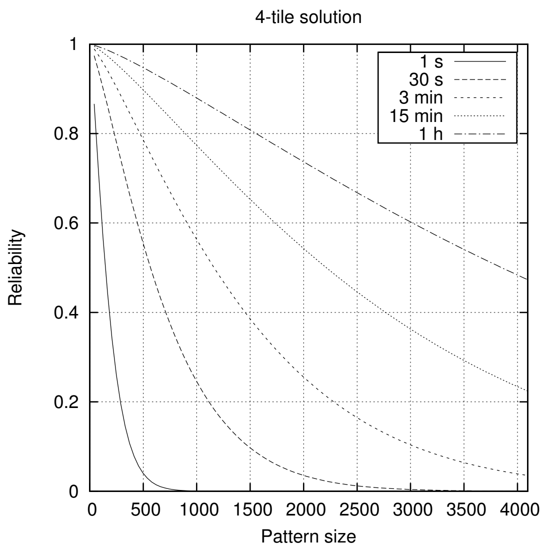
In this section, we present results on computing the reliability of tile sets using the method given above. We assume that the assembly process takes place in room temperature (298 K). As a result, we use the value $k_f = A_f e^{-E_f/RT} \approx 6 \cdot 10^5$ /M/sec for the forward reaction rate.

Figure 16(a) shows the reliability of the 4-tile solution to the Sierpinski triangle pattern as a function of pattern size, using five distinct assembly times. As is to be expected, the longer the assembly time, the better the reliability.

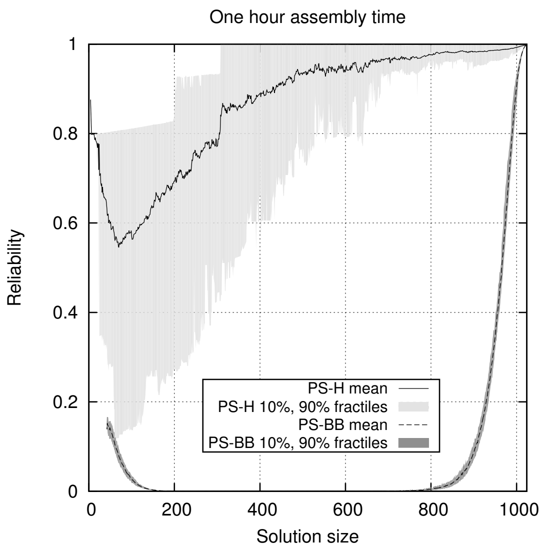
We also applied the method for computing the reliability to tile sets found by the partition-search algorithms. Our results show that the heuristics used in the PS-H algorithm improve not only the size of the tile sets found, but also the reliability of those tile sets. This can be easily understood 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 PS-H algorithm prefers merging class pairs with common glues, it reduces the number of such tile types effectively.

Figures 16(b)–16(d) present the reliability of tile sets found by the PS-H and PS-BB algorithms for the 32×32 Sierpinski triangle pattern, with assembly times of one hour, one day (24 hours) and one week. The runs were repeated 100 times; the mean reliability of each tile set size as well as the 10th and 90th percentiles are shown.

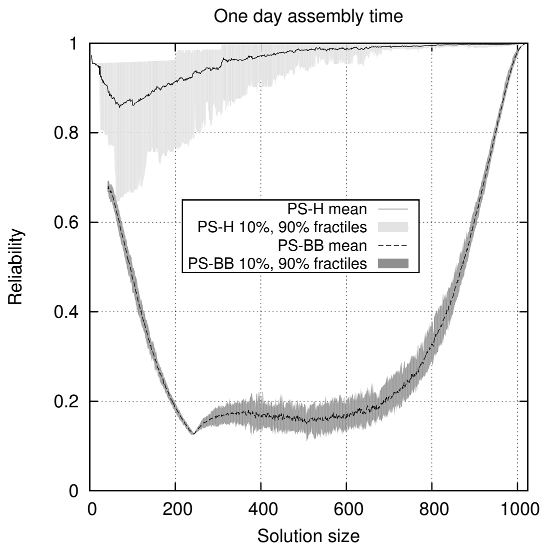
As for reliability, we expect a large set of runs of the PS-BB algorithm to produce a somewhat decent sample of all the possible tile sets for a pattern. Based on this, large and small tile sets seem to have a high reliability while medium-size tile sets are clearly less reliable on average. This observation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



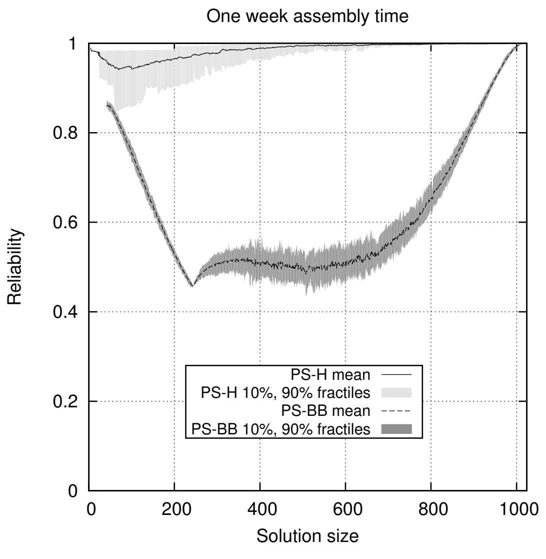
(a)



(b)



(c)



(d)

Figure 16: (a) The reliability of the minimal tile set as a function of pattern size for the Sierpinski triangle pattern, using several different assembly times. (b)–(d) The reliability of solutions for the 32×32 Sierpinski triangle pattern found by the PS-H and PS-BB algorithms, allowing assembly time of one hour, one day and one week.

8. Conclusions

We have investigated several algorithmic approaches towards an efficient solution to the PATS problem, i.e. the task of finding minimal tile sets which would self-assemble into a given k-coloured pattern starting from a bordering seed structure.

Our first algorithm is an exhaustive branch-and-bound method (PS-BB) which makes use of a search tree in the lattice of grid partitions. Given enough time, the algorithm finds a provably minimal tile set for any pattern. Numerical experiments indicate that the PS-BB algorithm is able to find minimal tile sets for randomly generated binary patterns of sizes up to 6×6 tiles. However, for larger patterns, the search space becomes too large for a complete exploration, even with the efficient pruning methods applied by the algorithm.

In a second approach, we addressed the relaxed objective of generating small but not necessarily minimal tile sets. Here our PS-H algorithm applies heuristic rules for optimising the order in which the search space of pattern-consistent tile sets is explored. Experimental results show that for most patterns, the PS-H algorithm is indeed able to find significantly smaller solutions than the PS-BB algorithm, in a reasonable amount of time.

In a third direction, we also considered solving the PATS problem using logic programming techniques, specifically the Answer Set Programming (ASP) method. For patterns having small optimal solutions, our chosen ASP solver is mostly very successful in discovering these solutions; however the running time of the solver seems to increase rapidly with the size of the minimum solution.

On a supporting topic, we used the kinetic Tile Assembly Model to assess the reliability of various tile sets generated by the PS-BB and PS-H algorithms, i.e. their probability of assembling the desired target pattern in an error-free manner. In comparison to the PS-BB approach, we find that the heuristics used in the PS-H algorithm improve also the reliability of tile sets found. In addition, we observed that large and small tile sets seem to have a high reliability, while medium-size tile sets are clearly less reliable on average.

One research question still open is the NP-hardness of the PATS problem restricted to 2-colour patterns. As for new solving methods, further work could include developing polynomial-time approximation algorithms. The declarative approach could possibly be applied to instances with larger optimal solutions by developing a more efficient ASP or boolean satisfiability encoding.

Acknowledgements

We thank the anonymous reviewers for their helpful feedback. The demanding numerical computations were performed on the Triton computing cluster provided by the Aalto University Science-IT programme.

References

[1] M. Göös, P. Orponen, Synthesizing minimal tile sets for patterned DNA self-assembly, in: Proc. 16th International Conference on DNA Computing and Molecular Programming (DNA 2010), volume 6518 of *LNCS*, Springer, Berlin, Germany, 2011, pp. 71–82. doi:10.1007/978-3-642-18305-8_7.

[2] T. Lempiäinen, E. Czeizler, P. Orponen, Synthesizing small and reliable tile sets for patterned DNA self-assembly, in: Proc. 17th International Conference on DNA Computing and Molecular Programming (DNA 2011), volume 6937 of *LNCS*, Springer, Berlin, Germany, 2011, pp. 145–159. doi:10.1007/978-3-642-23638-9_13.

[3] S. M. Douglas, H. Dietz, T. Liedl, B. Högberg, F. Graf, W. M. Shih, Self-assembly of DNA into nanoscale three-dimensional shapes, Nature 459 (2009) 414–418.

[4] A. Kuzyk, K. T. Laitinen, P. Törmä, DNA origami as a nanoscale template for protein assembly, Nanotechnology 20 (2009) 235305:1–235305:5.

[5] K. Lund, A. J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. N. Stojanovic, N. G. Walter, E. Winfree, H. Yan, Molecular robots guided by prescriptive landscapes, Nature 465 (2010) 206–210.

[6] Z. Zhang, E. M. Olsen, M. Kryger, N. V. Voigt, T. Tørring, E. Gültekin, M. Nielsen, R. MohammadZadegan, E. S. Andersen, M. M. Nielsen, J. Kjems, V. Birkedal, K. V. Gothelf, A DNA tile actuator with eleven discrete states, Angewandte Chemie International Edition 50 (2011) 3983–3987.

[7] L. Qian, E. Winfree, Scaling up digital circuit computation with DNA strand displacement cascades, Science 332 (2011) 1196–1201.

[8] L. Qian, E. Winfree, J. Bruck, Neural network computation with DNA strand displacement cascades, Nature 475 (2011) 368–372.

[9] J. Liu, Z. Cao, Y. Lu, Functional nucleic acid sensors, Chemical Reviews 109 (2009) 1948–1998.

[10] J. Li, H. Pei, B. Zhu, L. Liang, M. Wei, Y. He, N. Chen, D. Li, Q. Huang, C. Fan, Self-assembled multivalent DNA nanostructures for noninvasive intracellular delivery of immunostimulatory CpG oligonucleotides, ACS Nano 5 (2011) 8783–8789.

[11] K. N. Kim, K. Sarveswaran, L. Mark, M. Lieberman, DNA origami as self-assembling circuit boards, in: Proc. 9th International Conference on Unconventional Computation (UC 2010), volume 6079 of *LNCS*, Springer, Berlin, Germany, 2010, pp. 56–68. doi:10.1007/978-3-642-13523-1_9.

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

[13] E. Winfree, F. Liu, L. A. Wenzler, N. C. Seeman, Design and self-assembly of two-dimensional DNA crystals, Nature 394 (1998) 539–544.

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

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

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

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

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

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

[20] E. Czeizler, A. Popa, Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly, in: Proc. 18th International Conference on DNA Computing and Molecular Programming (DNA 2012), volume 7433 of *LNCS*, Springer, Berlin, Germany, 2012, pp. 58–72. doi:10.1007/978-3-642-32208-2_5.

[21] S. Seki, Combinatorial optimization in pattern assembly, in: Proc. 12th International Conference on Unconventional Computation and Natural Computation (UCNC 2013), volume 7956 of *LNCS*, Springer, Berlin, Germany, 2013, pp. 220–231. doi:10.1007/978-3-642-39074-6_21, see also arXiv:1301.3771.

[22] E. Winfree, Simulations of Computing by Self-Assembly, Technical Report CaltechCSTR:1998.22, California Institute of Technology, 1998. URL: http://resolver.caltech.edu/CaltechCSTR:1998.22.

[23] P. W. K. Rothemund, E. Winfree, The program-size complexity of self-assembled squares, in: Proc. 32nd Annual ACM Symposium on Theory of Computing (STOC 2000), ACM, New York, NY, USA, 2000, pp. 459–468. doi:10.1145/335305.335358.

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

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

[26] V. Lifschitz, What is answer set programming?, in: Proc. 23rd AAAI Conference on Artificial Intelligence (AAAI 2008), AAAI Press, Menlo Park, California, USA, 2008, pp. 1594–1597.

[27] H. Wang, Proving theorems by pattern recognition – II, Bell System Technical Journal 40 (1961) 1–41.

[28] B. Grünbaum, G. C. Shephard, Tilings and Patterns, W. H. Freeman and Company, New York, NY, USA, 1986.

[29] K. Fujibayashi, R. Hariadi, S. H. Park, E. Winfree, S. Murata, Toward reliable algorithmic self-assembly of DNA tiles: A fixed-width cellular automaton pattern, Nano Letters 8 (2008) 1791–1797.

[30] S. H. Park, H. Yan, J. H. Reif, T. H. LaBean, G. Finkelstein, Electronic nanostructures templated on self-assembled DNA scaffolds, Nanotechnology 15 (2004) S525–S527.

[31] J. Clausen, M. Perregaard, On the best search strategy in parallel branch-and-bound: Best-First Search versus Lazy Depth-First Search, Annals of Operations Research 90 (1999) 1–17.

[32] T. Syrjänen, Implementation of Local Grounding for Logic Programs With Stable Model Semantics, Technical Report B18, Helsinki University of Technology, Digital Systems Laboratory, 1998. URL: http://www.tcs.hut.fi/Publications/info/bibdb.HUT-TCS-B18.shtml.

[33] M. Gebser, R. Kaminski, A. König, T. Schaub, Advances in *gringo* series 3, in: Proc. 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011), volume 6645 of *LNCS*, Springer, Berlin, Germany, 2011, pp. 345–351. doi:10.1007/978-3-642-20895-9_39.

[34] M. Gebser, B. Kaufmann, A. Neumann, T. Schaub, Conflict-driven answer set solving, in: Proc. 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), AAAI Press, Menlo Park, California, USA, 2007, pp. 386–392.

[35] I. Niemelä, P. Simons, Smodels – an implementation of the stable model and well-founded semantics for normal logic programs, in: Proc. 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997), volume 1265 of *LNCS*, Springer, Berlin, Germany, 1997, pp. 420–429. doi:10.1007/3-540-63255-7_32.

[36] K. Fujibayashi, S. Murata, Precise simulation model for DNA tile self-assembly, IEEE Transactions on Nanotechnology 8 (2009) 361–368.

[37] R. Schulman, E. Winfree, Programmable control of nucleation for algorithmic self-assembly, SIAM Journal of Computing 39 (2009) 1581–1616.

[38] J. SantaLucia, Jr., H. T. Allawi, P. A. Seneviratne, Improved nearest-neighbor parameters for predicting DNA duplex stability, Biochemistry 35 (1996) 3555–3562.

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

[40] C. Dwyer, V. Johri, M. Cheung, J. Patwardhan, A. Lebeck, D. Sorin, Design tools for a DNA-guided self-assembling carbon nanotube technology, Nanotechnology 15 (2004) 1240–1245.

[41] D. S. Lee, J. Svensson, S. W. Lee, Y. W. Park, E. E. B. Campbell, Fabrication of crossed junctions of semiconducting and metallic carbon nanotubes: A CNT-gated CNT-FET, Journal of Nanoscience and Nanotechnology 6 (2006) 1325–1330.

[42] A.-P. Eskelinen, A. Kuzyk, T. K. Kaltiaisenaho, M. Y. Timmermans, A. G. Nasibulin, E. I. Kauppinen, P. Törmä, Assembly of single-walled carbon nanotubes on DNA-origami templates through streptavidin–biotin interaction, Small 7 (2011) 746–750.

Appendix A. A design framework for carbon nanotube circuits affixed on DNA origami tiles

Recent years have witnessed a burst of experimental activity concerning algorithmic self-assembly of nanostructures, motivated at least in part by the potential of this approach as a radically new manufacturing technology. One of the presently most reliable self-assembling, programmable nanostructure architectures is DNA origami [15]. Several authors have announced the formation of DNA origami tiles, capable of further assembly into larger, fully addressable, 1D and 2D scaffolds [11, 16, 39]. Such scaffolds make possible the construction of highly complex structures on top of them [4], prospectively including nanocircuits. In Czeizler et al. [18], we proposed a generic framework for the design of Carbon Nanotube Field Effect Transistor (CNFET) circuits. The elements of these circuits are Carbon Nanotube Field Effect Transistors and Carbon Nanotube Wires. They are placed on top of different DNA origami tiles which self-assemble into any desired circuit.

Single-wall carbon nanotubes (CNs) can be fabricated as either metallic (m) or semiconducting (s). A cross-junction between an m-type and an s-type CN generates a structure with field effect transistor (FET) behaviour [40, 41]. In this way, both p-type and n-type FETs are realisable (a p-type FET is ON when input is "0", while an n-type FET is ON when input is "1"). Moreover, experimental implementations have been provided, affixing these structures on top of DNA origami [12, 42].

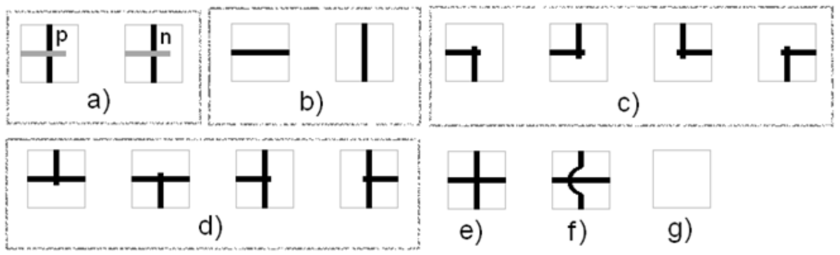


Figure A.17: The 14 tile types and the blank tile, out of which any CNFET circuit can be assembled: (a) p-type and n-type CNFETs, (b) straight CNWs, (c) corner CNWs, (d)–(e) 3-way and 4-way CNW junctions, (f) crossing but non-interacting CNWs and (g) blank tile.

Based on the above experimental results, we provided in Czeizler et al. [18] a "universal" set of 14 functionalised DNA origami tiles, such that, with a proper selection of "glues" on the tiles, any desired CNFET circuit can be self-assembled from this basis. These tile types are presented in Figure A.17 (the marks on the tiles indicate the arrangements of the CNs affixed on the respective DNA origami): (a) p-type and n-type CNFETs, (b) straight (horizontal or vertical) CN wires (CNWs), (c) corner CNWs, (d)–(e) 3-way and 4-way junction CNWs and (f) crossing but non-interacting CNWs. Additionally, when analysing fault tolerant architectures, it is convenient to introduce also (g) a blank tile. In order to design a particular nanocircuit, one first prepares the transistor circuit design using the 14 basis tiles indicated. Then, an optimal number of glues for these tiles is computed and finally, appropriate "sticky end" sequences for implementing the glues are designed for the DNA origami tiles. In Figure A.18 we present the designs for a CMOS inverter, NAND gate and full adder.

Some of the advantages of this approach are that it decouples the self-assembly aspects of the manufacturing process from the transistor circuit design and that it allows for a structured and clear circuit design. Moreover, it also supports efficient high-level analysis of the purported circuits, both by computer simulations and by analytical means. For instance, all assembly errors can at this level be treated as tiling errors, leading to a transparent design discipline for fault-tolerant architectures.

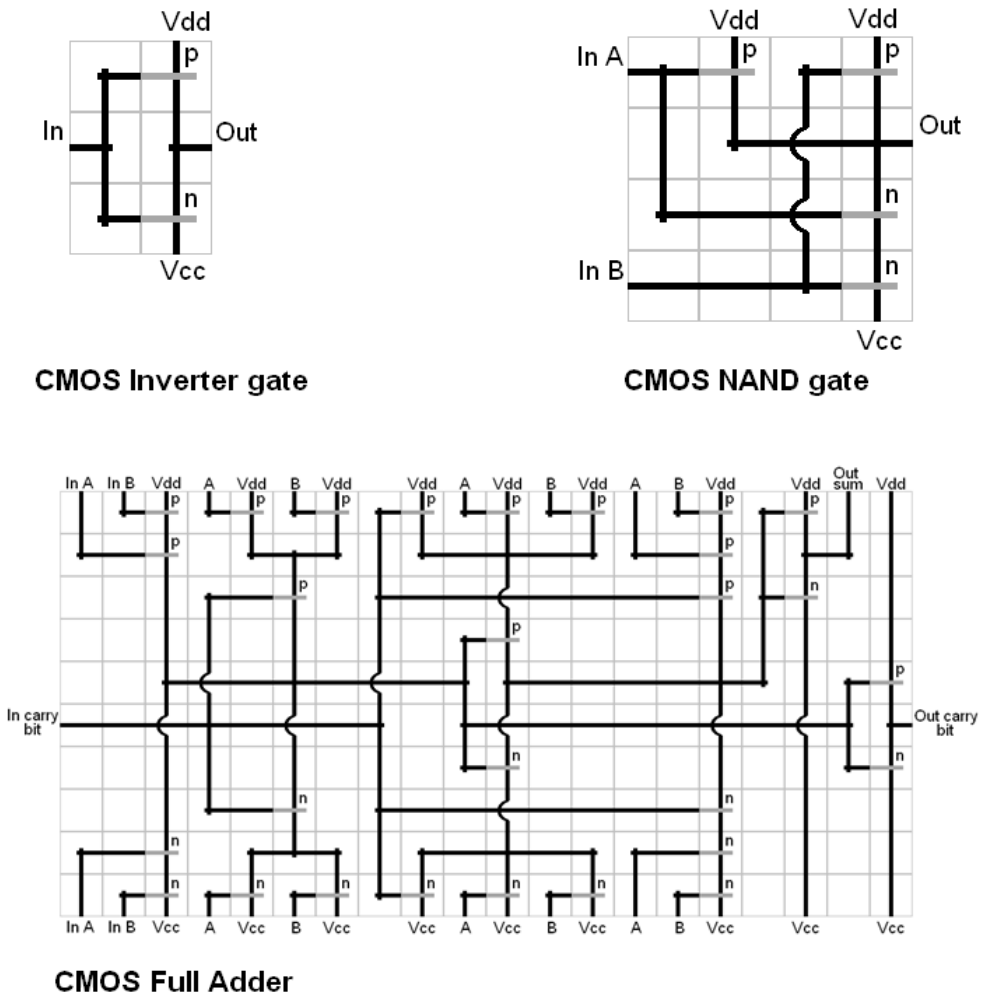

Vdd
p
In
Out
n
Vcc
CMOS Inverter gate
Vdd
Vdd
In A
p
p
Out
n
In B
n
Vcc
CMOS NAND gate
In A In B Vdd A Vdd B Vdd Vdd A Vdd B Vdd A B Vdd Vdd Out sum Vdd
In carry bit
Out carry bit
p
n
In A In B Vcc A Vcc B Vcc Vcc A Vcc B Vcc A B Vcc Vcc Vcc
CMOS Full Adder

Figure A.18: Examples of CNFET circuit design: an inverter gate, a NAND gate and a full adder.