View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Helsingin yliopiston digitaalinen arkisto

# The PATS Problem: Search Methods and Reliability

Tuomo Lempiäinen

Minor subject thesis
UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, 21st May 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

| | |
|---|---|
| Tiedekunta — Fakultet — Faculty<br>Faculty of Science | Laitos — Institution — Department<br>Department of Computer Science |

Tekijä — Författare — Author
Tuomo Lempiäinen

Työn nimi — Arbetets titel — Title
The PATS Problem: Search Methods and Reliability

Oppiaine — Läroämne — Subject
Computer Science

| | | |
|---|---|---|
| Työn laji — Arbetets art — Level<br>Minor subject thesis | Aika — Datum — Month and year<br>21st May 2015 | Sivumäärä — Sidoantal — Number of pages<br>27 |

Tiivistelmä — Referat — Abstract

This work studies an NP-hard combinatorial optimisation problem, the Pattern self-Assembly Tile set Synthesis (PATS) problem, which stems from the field of DNA self-assembly. In this problem, we are given a coloured rectangular pattern as input, and the task is to find a minimal set of unit square tiles that self-assemble that pattern in the abstract Tile Assembly Model (aTAM).

We present two new search methods for the PATS problem: a heuristic algorithm that conducts a search in the lattice of partitions of the input grid, and a declarative approach that uses the Answer Set Programming (ASP) paradigm. The former is based on a previous algorithm by Göös and Orponen (DNA 2010), and performs better in finding relatively small solutions even for quite large input patterns. The latter proves to find the optimal solution quickly in cases where it is small.

In addition to the search procedures, we develop a method for estimating the reliability of solutions to the PATS problem from a stochastic point of view. It turns out that tile sets found by our procedures, as well as small tile sets in general, have a higher probability of error-free assembly compared to those that can be found by previous methods.

Avainsanat — Nyckelord — Keywords
DNA self-assembly, tile assembly model, pattern assembly, tile set synthesis

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Campus Library and the digital repository HELDA

Muita tietoja — Övriga uppgifter — Additional information

## Contents

# 1 Introduction

The idea of manipulating matter and performing computation at the molecular scale was first proposed by Richard Feynman in his visionary 1959 talk *There's Plenty of Room at the Bottom.* Nanotechnology as a field emerged in the 1980s, and has extensively advanced ever since. Due to its ability to store information, DNA is often the material of choice for producing molecular-scale structures and devices. The physical properties of DNA, as well as methods to synthesise it, are already well understood [5].

Since the traditional top-down approach to control material is often simply not possible at molecular scales, self-assembly—the process of individual components assembling themselves into complex structures based on local interactions—has emerged as a viable alternative. Self-assembly also has the advantage of enabling massive parallelism: millions of copies of a structure could be formed simultaneously in a single test tube.

DNA self-assembly can be viewed from two different angles: on one hand, as a method to build nanoscale structures, and on the other hand, as a model of computation. In the former, the goal is to design DNA sequences that automatically assemble into the desired structure—to be used, for example, as a scaffold for other particles. In the latter, the goal is to encode the input of a computational problem into DNA strands such that interactions between the strands eventually produce an encoding of the desired output.

Seeman [25] did pioneering work with DNA nanotechnology in the 1980s, while the field of algorithmic self-assembly was initiated by the seminal experiment of Adleman [1] in 1994. Since the concept of autonomous self-assembly is intrinsically related to computation, the desire to build ever more complex structures in a systematic manner has brought these two approaches together in recent years.

A prominent direction of research is the attempt to build two-dimensional DNA templates, to which functional units can be attached. One way to implement such a template is to self-assemble it from square units called *tiles.* A tile is a DNA complex that has a short single-stranded *sticky end* on each of its four edges. Several methods to build such tiles out of DNA are known, the most important being double-crossover tiles [32] and DNA origami [22]. Sticky ends with Watson-Crick complementary sequences can hybridise and hence bind tiles to each other. This forms the physical background for the work in this thesis. Our considerations take place within the *abstract Tile Assembly Model (aTAM)* of Winfree [29–31], which provides a simplified combinatorial model for the behaviour of DNA tiles.

To enable the formation of aperiodic structures, such as electronic circuits, where different components will be attached to different tiles in the template, the template needs to be addressable. Each position of the desired lattice can be assigned a colour based on the component that will be attached to that position. Also tile types, that is, combinations of four sticky ends, can be coloured based on the component attached to tiles of that type. To self-assemble the target structure, one then needs to find a set of coloured tile types that will implement the desired colour pattern.

A straightforward way to construct such a set of tile types is to use a unique tile type for each position of the pattern. However, it is often possible to find a significantly smaller tile set, where some tile types are used in several positions of the pattern—essentially, they are doing computation to assemble the structure. This reduces the amount of laboratory work required for chemical implementation of the tile system, since the cardinality of the tile set is proportional to the number of distinct DNA strands that need to be synthesised. Ma and Lombardi [19] formulated this as a combinatorial optimisation problem called the *Pattern self-Assembly Tile set Synthesis (PATS)* problem and proposed two greedy algorithms for finding solutions.

Göös and Orponen [10] continued the work of Ma and Lombardi by developing an exhaustive PS-BB (partition search branch-and-bound) algorithm for finding minimum-size tile sets. However, their algorithm is feasible only for small patterns—at most roughly $7 \times 7$ tiles. This is to be expected, since the PATS problem was subsequently proved to be $\mathsf{NP}$-hard [4].

In this thesis, we continue the quest for small tile sets by presenting two new search methods. First, we adapt the so-called partition-search framework of Göös and Orponen to construct a heuristic PS-H (partition-search with heuristics) algorithm that can often find rather small, but not necessarily minimal, tile sets for large pattern sizes. We also note that the performance of the heuristic algorithm can be improved significantly by running several independent short runs with different random seeds. Second, we formulate the PATS problem as an Answer Set Programming (ASP) [17] task, and make use of a generic ASP solver to find solutions to it. We investigate the performance of both methods experimentally and show that they outperform the baseline. Our methods are complementary to each other in the sense that the ASP approach suits well for large input patterns that have a very small solution, while the heuristic algorithm as able to find reasonably small tile sets also in more general situations.

Since the physical self-assembly process is inherently stochastic, it is of interest to asses the reliability of tile sets, that is, the probability that they assemble the desired target pattern without errors. As our third contribution, we develop a method for estimating this quantity, based on Winfree's analysis of a kinetic version of the tile assembly model [31]. We present data on the reliability of tile sets found by the PS-BB and PS-H algorithms, and obtain that our PS-H approach constitutes an improvement over PS-BB also in this respect.

**Acknowledgements**

This thesis is largely based on a conference report [16] by myself and co-authors. The work has also appeared in a subsequent journal article [9]. The experimental performance results reported in this work were obtained on the Triton computing cluster provided by the Science-IT project of Aalto University.

I would like to thank Pekka Orponen and Eugen Czeizler for guidance and fruitful collaboration, Jukka Suomela for allowing me to spend a portion of my working hours on the aforementioned publications while being employed at another project, and finally, Jyrki Kivinen and Matti Järvisalo for examining this thesis as well as their patience with the rather slow writing process.

# 2 Preliminaries

In this section, we first review the abstract Tile Assembly Model (aTAM) introduced by Winfree [29–31], and then define the PATS problem introduced by Ma and Lombardi [19]. Our presentation of the aTAM is largely inspired by that of Rothemund and Winfree [23].

## 2.1 The Abstract Tile Assembly Model

The aTAM is an extension of the theory of Wang tiles [11, 28], designed specifically for modelling the self-assembly of molecules, such as DNA. The basic building blocks of the aTAM are unit square tiles, with different glues on their edges. Each glue represents a molecular binding domain that can bind the tile to another tile having a compatible glue on its abutting edge. There is a finite number of different tile types (sequences of four glues), each available in an unlimited number of copies. The tiles grow an assembly on a two-dimensional grid by addition of a single tile at a time, starting from a given seed assembly.

More formally, we work on the two-dimensional grid of integer positions, $\mathbb{Z} \times \mathbb{Z}$. We use the four cardinal directions $\mathcal{D} = \{N, E, S, W\}$ (north, east, south and west) as functions from the grid to itself: $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$ and $W(x, y) = (x - 1, y)$. Note that $N = S^{-1}$ and $E = W^{-1}$. Let $\Sigma$ be a finite set of *glue types*. A *tile type* $t$ over $\Sigma$ is a quadruple $t = (\sigma_N(t), \sigma_E(t), \sigma_S(t), \sigma_W(t)) \in \Sigma^4$ of glue types for each of the four sides. A *tile set* $T \subseteq \Sigma^4$ is a finite collection of different tile types. A *glue strength function* is a function $s\colon \Sigma \times \Sigma \to \mathbb{N}$ such that $s(\sigma, \sigma') = s(\sigma', \sigma)$ for all $\sigma, \sigma' \in \Sigma$, that is, it associates a non-negative integer strength to each pair of glue types. In this work we consider only glue strength functions $s$ such that $s(\sigma, \sigma') > 0$ only if $\sigma = \sigma'$. For each $\sigma \in \Sigma$, we call $s(\sigma, \sigma)$ the *strength* of the glue type $\sigma$.

A *tile assembly* $\mathcal{A}$ is a partial function $\mathcal{A}\colon \mathbb{Z} \times \mathbb{Z} \to \Sigma^4$ that describes a collection of tiles positioned on certain locations of the grid. The domain of $\mathcal{A}$, denoted by $\operatorname{dom}(\mathcal{A})$, is the set of locations $(x, y)$ to which a tile has been assigned in $\mathcal{A}$. A *tile assembly system* (TAS) $\mathscr{T}$ is a quadruple $\mathscr{T} = (T, \mathcal{S}, s, \tau)$ consisting of a tile set $T$, a tile assembly $\mathcal{S}$ (the *seed assembly*), a glue strength function $s$ and a *temperature* parameter $\tau \in \mathbb{N}$. Given an existing tile assembly, such as the seed assembly $\mathcal{S}$, a tile of type $t \in T$ can adjoin the assembly if the total binding strength between the tile and the assembly, given by the function $s$, is at least as large as the temperature threshold $\tau$. Tiles cannot be rotated, that is, when a tile of type $t$ adjoins the assembly, its glue $\sigma_D(t)$ is matched against a glue in direction $D$ for each $D \in \mathcal{D}$. Note that the types of tiles in the seed assembly $\mathcal{S}$ do not need to be in $T$, but that an assembly can be further extended only by tiles of a type from $T$.

Formally, given a TAS $\mathscr{T}$, we define self-assembly as a relation $\to_{\mathscr{T}}$ between tile assemblies. We say that assembly $\mathcal{A}$ *produces directly* assembly $\mathcal{A}'$, in symbols $\mathcal{A} \to_{\mathscr{T}} \mathcal{A}'$, if there exists a location $(x, y) \in \mathbb{Z} \times \mathbb{Z} \setminus \operatorname{dom}(\mathcal{A})$ and a tile type $t \in T$ such that $\operatorname{dom}(\mathcal{A}') = \operatorname{dom}(\mathcal{A}) \cup \{(x, y)\}$, $\mathcal{A}'(x, y) = t$, $\mathcal{A}'(x', y') = \mathcal{A}(x', y')$ for all $(x', y') \in \operatorname{dom}(\mathcal{A})$ and

$$\sum_{\substack{D \in \mathcal{D} \\ D(x,y) \in \operatorname{dom}(\mathcal{A})}} s(\sigma_D(t), \sigma_{D^{-1}}(\mathcal{A}(D(x, y)))) \geq \tau.$$

Let $\to^*_{\mathscr{T}}$ be the reflexive transitive closure of $\to_{\mathscr{T}}$. The TAS $\mathscr{T}$ *produces* an assembly $\mathcal{A}$ if $\mathcal{S} \to^*_{\mathscr{T}} \mathcal{A}$, that is, if $\mathcal{A}$ is an extension of the seed assembly $\mathcal{S}$. Let $\operatorname{Prod}(\mathscr{T}) = \{\mathcal{A} \mid \mathcal{S} \to^*_{\mathscr{T}} \mathcal{A}\}$ be the set of all assemblies produced by $\mathscr{T}$. A *terminal assembly* is an assembly $\mathcal{A} \in \operatorname{Prod}(\mathscr{T})$ such that there does not exist any assembly $\mathcal{A}'$ satisfying $\mathcal{A} \to_{\mathscr{T}} \mathcal{A}'$. We denote the set of all terminal assemblies by $\operatorname{Term}(\mathscr{T})$. The TAS $\mathscr{T}$ is *deterministic* if for each assembly $\mathcal{A} \in \operatorname{Prod}(\mathscr{T})$ and each location $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ there exists at most one tile type $t \in T$ that can adjoin $\mathcal{A}$ at $(x, y)$.

## 2.2 The PATS Problem

Given natural numbers $m, n \in \mathbb{N}$, we denote by $[m]$ and $[m, n]$ the sets $\{1, 2, \ldots, m\}$ and $\{m, m + 1, \ldots, n\}$, respectively. A surjective function $c\colon [m] \times [n] \to [k]$ defines a *$k$-coloured pattern* of size $m \times n$. Given a TAS $\mathscr{T}$ with tile set $T$, a *tile colouring* is a function $d\colon T \to [k]$ that associates each tile type with one of the $k$ colours.

In the PATS problem, we are given a finite pattern, and our task is to find a TAS $\mathscr{T}$ admitting a tile colouring such that each terminal assembly of $\mathscr{T}$ implements the desired pattern. It is of course trivial to construct a TAS with a distinct tile for each location of the pattern. However, we aim to minimise $|T|$, the number of distinct tiles needed. In this context, we

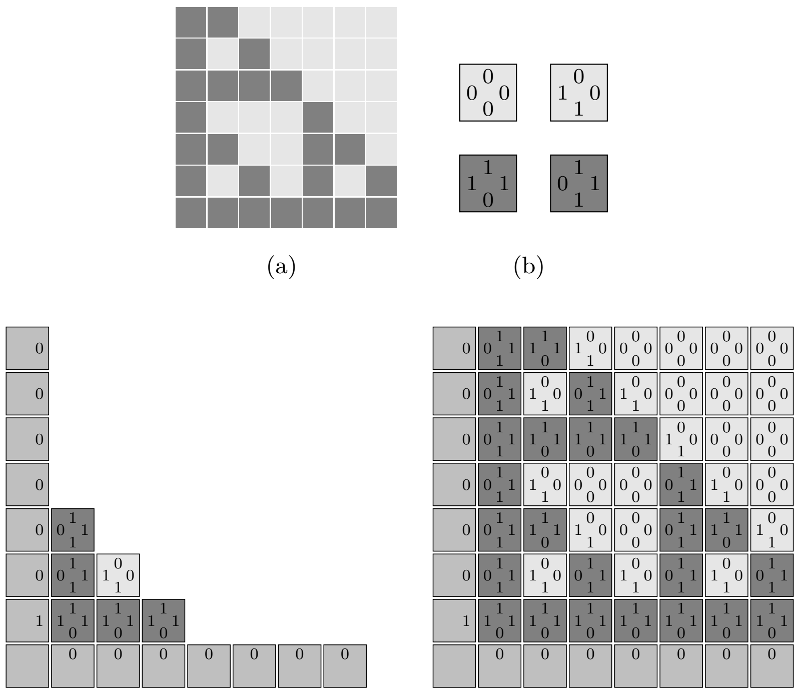

(a) (b) (c) (d)

Figure 1: (a) A $7 \times 7$ instance of the PATS problem (the Sierpinski triangle pattern). (b) A set of four tiles that self-assembles the pattern in (a). (c) An L-shaped seed and unfinished assembly process. (d) The finished assembly.

assume the seed assembly is L-shaped, that is, it consists of the west and south borders. See Figure 1 for an illustration. Formally, we define the PATS problem as a decision problem as follows.

**Definition 1** (The PATS Problem)**.**

*Given:* A $k$-coloured pattern $c\colon [m] \times [n] \to [k]$ and an integer $K$.

*Find:* A TAS $\mathscr{T} = (T, \mathcal{S}, s, 2)$ that satisfies the following conditions.

P1. All tile types in $T$ have only glue types of 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 $d\colon T \to [k]$ such that each terminal assembly $\mathcal{A} \in \mathrm{Term}(\mathscr{T})$ satisfies $d(\mathcal{A}(x, y)) = c(x, y)$ for all $(x, y) \in [m] \times [n]$.

P4. The size of $T$ is at most $K$.

By using an L-shaped seed assembly, we make a clear distinction between the complexity of assembling the boundaries and the complexity of the pattern itself. It is possible to self-assemble the boundaries using $m + n + 1$ tiles of distinct types having glues of strength 2 on their abutting edges. However, our approach also allows using other techniques, such as DNA origami [22], for constructing the borders. One experiment utilising such design for DNA tile assembly is due to Fujibayashi et al. [6].

Due to the L-shaped seed assembly and constraint P1, a tile can adjoin an assembly at location $(x, y)$ only if the assembly already has tiles at locations $W(x, y)$ and $S(x, y)$. From this we get the following characterisation for the determinism of solutions to the PATS problem [10].

**Lemma 2.** *A TAS $\mathscr{T} = (T, \mathcal{S}, s, 2)$ that satisfies conditions* P1 *and* P2*, and where each tile type appears in at least one terminal assembly, is deterministic if and only if for all pairs of glue types $(\sigma_1, \sigma_2) \in \Sigma \times \Sigma$ there exists at most one tile type $t \in T$ such that $\sigma_W(t) = \sigma_1$ and $\sigma_S(t) = \sigma_2$.*

Using the above lemma, it is easy to prove the following result [10], which simplifies the search for optimal solutions to the PATS problem.

**Lemma 3.** *All optimal solutions to the PATS problem are deterministic.*

*Proof.* Let $\mathscr{T} = (T, \mathcal{S}, s, 2)$ be an optimal solution to the PATS problem. Assume that $\mathscr{T}$ is not deterministic. By Lemma 2 there exist tile types $t_1, t_2 \in T, t_1 \neq t_2$, such that $\sigma_W(t_1) = \sigma_W(t_2)$ and $\sigma_S(t_1) = \sigma_S(t_2)$. Consider now a TAS $\mathscr{T}' = (T \setminus \{t_1\}, \mathcal{S}, s, 2)$ and a terminal assembly $\mathcal{A} \in \text{Term}(\mathscr{T}')$. Since $t_1$ can adjoin assembly $\mathcal{A}$ at location $(x, y)$ if and only if $t_2$ can adjoin $\mathcal{A}$ at $(x, y)$, we have $\mathcal{A} \in \text{Term}(\mathscr{T})$ and hence $\text{Term}(\mathscr{T}') \subseteq \text{Term}(\mathscr{T})$. Thus also $\mathscr{T}'$ satisfies conditions P1–P3. Now $\mathscr{T}$ is not optimal, a contradiction. $\square$

Observe that for each TAS $\mathscr{T} = (T, \mathcal{S}, s, 2)$, the pair $(\text{Prod}(\mathscr{T}), \rightarrow^*_{\mathscr{T}})$ is a partially ordered set. If $\mathscr{T}$ is deterministic and satisfies conditions P1 and P2, it is straightforward to see that $(\text{Prod}(\mathscr{T}), \rightarrow^*_{\mathscr{T}})$ is a lattice, and there is only one maximal element in $\text{Prod}(\mathscr{T})$, that is, $\text{Term}(\mathscr{T}) = \{\mathcal{P}\}$ for some assembly $\mathcal{P} \in \text{Prod}(\mathscr{T})$. In that case we say that $\mathscr{T}$ *uniquely produces* $\mathcal{P}$.

## 2.3 Computational Complexity and Problem Variants

The PATS problem was originally claimed to be NP-complete by Ma and Lombardi [18, 19]. To the best of our knowledge, the first adequate proof was provided by Czeizler and Popa [3, 4].

Seki [26] defined a more restricted problem, $k$-PATS, where the input pattern consists of at most $k$ colours for some constant $k$. He proved a stronger result, stating that the $k$-PATS problem is NP-complete for $k = 60$. Subsequently, Kari, Kopecki and Seki [15] defined a new problem variant, *multiple bound PATS*, where an upper bound for the number of tile types of each colour is given. They showed that this problem is NP-complete for 3-colour patterns. They also presented a proof more concise than the previous ones for the NP-completeness of the original PATS problem.

It was conjectured that 2-PATS is NP-complete, but it took some time to reach that goal. The first improvement from 60-PATS was the proof of Johnsen, Kao and Seki [12] showing that 29-PATS is NP-complete. In a recent breakthrough, Kari et al. [14] finally obtained that 2-PATS is NP-complete. A crucial part of their result relies on a computer-assisted proof, which requires approximately one year of computation time. Subsequently, a manually-checkable proof for 11-PATS was discovered [13].

## 3 The Search Space of Partitions

In this section, we present the partition-search framework that was introduced by Göös and Orponen [10] as a basis for their branch-and-bound algorithm and that also serves as a basis for our heuristic algorithm.

Let $X$ be the set of all partitions of the set $[m] \times [n]$. We say that partition $P \in X$ is *coarser* than partition $P' \in X$ or that $P'$ is a *refinement* of $P$, and write $P \sqsubseteq P'$, if for all partition classes $p' \in P'$ there is a partition class $p \in P$ such that $p' \subseteq p$.

We associate a partition $P(c)$ of the set $[m] \times [n]$ with each $k$-colouring $c\colon [m] \times [n] \to [k]$ by defining

$$P(c) = \left\{ c^{-1}(i) \;\middle|\; i \in [k] \right\}.$$

Here $c^{-1}(i) = \{(m,n) \mid c(m,n) = i\}$ is the preimage of $i$ for each $i \in [k]$. We also associate a partition $P(\mathscr{T})$ of the set $[m] \times [n]$ with every deterministic TAS $\mathscr{T} = (T, \mathcal{S}, s, 2)$ that satisfies conditions P1 and P2. We set

$$P(\mathscr{T}) = \left\{ \mathcal{P}^{-1}(t) \cap [m] \times [n] \;\middle|\; t \in \mathcal{P}([m] \times [n]) \right\},$$

where $\mathcal{P}$ is the assembly uniquely produced by $\mathscr{T}$. If all tile types $t \in T$ appear in the terminal assembly $\mathcal{P}$, we have $|P(\mathscr{T})| = |T|$. Now we can restate condition P3 in the definition of the PATS problem in terms of partitions. TAS $\mathscr{T}$ satisfies P3 if and only if

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

We say that a partition $P$ is *constructible* if $P = P(\mathscr{T})$ for some deterministic TAS $\mathscr{T}$ that satisfies P1 and P2.

To sum up, we can rephrase the PATS problem as a problem of finding a certain kind of partition as follows.

**Lemma 4.** *An optimal solution $\mathscr{T}$ to the PATS problem given by colouring $c\colon [m] \times [n] \to k$ corresponds to a partition $P \in X$ such that $P$ is constructible, $P(c) \sqsubseteq P$ and $|P|$ is minimal.*

### 3.1 Most General Tile Assignments

Next we will describe a polynomial-time algorithm for deciding the constructibility of a given partition in $X$. For this we need the following concept.

**Definition 5.** Let $P$ be a partition of $[m] \times [n]$. A *most general tile assignment* (MGTA) is a function $f\colon P \to \Sigma^4$ such that the following conditions hold.

A1. If $x, y \in [m] \times [n]$, $D(x) = y$ for some $D \in \mathcal{D}$ and $x \in p_x \in P$, $y \in p_y \in P$, then $\sigma_D(f(p_x)) = \sigma_{D^{-1}}(f(p_y))$.

A2. For all functions $g\colon P \to \Sigma^4$ that satisfy condition A1 we have

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

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

Informally, condition A1 states that any two adjacent positions agree on the glue on their abutting edges, and condition A2 states that there are no other assignments that express more variation in glues while still satisfying A1.

Given a partition $P \in X$ and an assignment $f\colon P \to \Sigma^4$, we can *merge glues* $a \in \Sigma^4$ and $b \in \Sigma^4$ to obtain from $f$ a new assignment $g\colon P \to \Sigma^4$ that is defined as follows:

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

for all $(p, D) \in P \times \mathcal{D}$.

We can generate a MGTA for a given partition $P \in X$ as follows. Let $x_1, x_2, \ldots, x_{mn}$ be some enumeration of all the positions in $[m] \times [n]$. Let $f_0\colon P \to \Sigma^4$ be a function that assigns a unique glue type to each tile type edge, that is, the mapping $(p, D) \mapsto \sigma_D(f(p))$ injective. Assuming that $f_{i-1}$ is defined for some $i \in [mn]$, let $p_i$ be the class of $P$ for which $x_i \in p_i$. If $N(x_i) \in [m] \times [n]$, let $p_i^N$ be the class of $P$ for which $N(x_i) \in p_i^N$ and merge the glues $\sigma_N(f_{i-1}(p_i))$ and $\sigma_S(f_{i-1}(p_i^N))$ to obtain assignment $f_i'$ from $f_{i-1}$. Otherwise, set $f_i' = f_{i-1}$. Similarly, if $E(x_i) \in [m] \times [n]$, let $p_i^E$ be the class of $P$ for which $E(x_i) \in p_i^E$ and merge the glues $\sigma_E(f_i'(p_i))$ and $\sigma_W(f_i'(p_i^E))$ to obtain assignment $f_i$ from $f_i'$. This iterative definition gives us a sequence of assignments $f_0, f_1, \ldots, f_{mn}$.

**Lemma 6.** *The assignment $f_{mn}\colon P \to \Sigma^4$ generated by the above algorithm is a MGTA.*

*Proof.* The glues on each pair of abutting edges are merged in some step of the process generating $f_{mn}$. Thus $f_{mn}$ satisfies condition A1. On the other hand, glues are only merged if they appear on abutting edges. This implies that if $f_{mn}$ assigns the same glues to two edges, then any assignment $g$ satisfying A1 also does, and hence $f_{mn}$ satisfies also condition A2. □

**Lemma 7.** *For each partition $P$ of $[m] \times [n]$, MGTAs $f : P \to \Sigma^4$ are unique up to permuting the glues.*

*Proof.* Given an MGTA $f : P \to \Sigma^4$, we associate with it a partition $P_f$ on $P \times \mathcal{D}$ such that $(p_1, D_1)$ and $(p_2, D_2)$ are in the same equivalence class of $P_f$ if and only if $\sigma_{D_1}(f(p_1)) = \sigma_{D_2}(f(p_2))$. Consider two MGTAs $f, g : P \to \Sigma^4$. Definition 5 implies that for all $(p_1, D_1), (p_2, D_2) \in P \times \mathcal{D}$ we have $\sigma_{D_1}(f(p_1)) = \sigma_{D_2}(f(p_2))$ if and only if $\sigma_{D_1}(g(p_1)) = \sigma_{D_2}(g(p_2))$. It follows that we have $P_f = P_g$. Now there is a permutation $h : \Sigma \to \Sigma$ such that $h(\sigma_D(f(p))) = \sigma_D(g(p))$ for all $(p, D) \in P \times \mathcal{D}$. □

Due to the above lemma, we can choose for each $P \in X$ some canonical representative from the class of all MGTAs $f : P \to \Sigma^4$. We call it *the MGTA for $P$.*

**Lemma 8.** *A partition $P \in X$ is constructible if and only if the MGTA $f : P \to \Sigma^4$ for $P$ is injective and the set $f(P)$ of tiles is deterministic.*

*Proof.* Assume that $P \in X$ is constructible. Let $\mathscr{T}$ be a deterministic TAS satisfying conditions P1 and P2 such that $P = P(\mathscr{T})$. Let $g : P \to \Sigma^4$ be the tile assignment induced by the uniquely produced assembly of $\mathscr{T}$. Clearly $g$ satisfies condition A1. Now, if $f$ is not injective, condition A2 implies that $g$ is not injective, which is a contradiction by the definition of $g$. If the tile set $f(P)$ is not deterministic, condition A2 implies that $\mathscr{T}$ is not deterministic, a contradiction.

For the other direction, assume that MGTA $f : P \to \Sigma^4$ is injective and $f(P)$ is deterministic. Define $s : \Sigma \times \Sigma \to \mathbb{N}$ such that $s(\sigma, \sigma') = 1$ if $\sigma = \sigma'$ and $s(\sigma, \sigma') = 0$ otherwise. As $f(P)$ is deterministic, it is possible to define a seed assembly $\mathcal{S} : ([0, m] \times \{0\}) \cup (\{0\} \times [0, n]) \to \Sigma^4$ such that tiles in $f(P)$ can adjoin $\mathcal{S}$ only in accordance with $f$. Consider now the TAS $\mathscr{T} = (f(P), \mathcal{S}, s, 2)$. By definition, $\mathscr{T}$ is deterministic and satisfies P1 and P2. It is easy to see that $\mathscr{T}$ uniquely produces a terminal assembly $\mathcal{P}$ that agrees with $f$, and consequently, $P(\mathscr{T}) \sqsubseteq P$. Since $f$ is injective, we have $|P| = |f(P)| = |P(\mathscr{T})|$, which implies $P(\mathscr{T}) = P$. □

## 3.2 A Partition-Search Algorithm

Now we are ready to describe how the search for tile sets is carried out. The basic idea, originating from Ma and Lombardi [19] and the experimental work of Park et al. [21], is as follows. First, form an *initial tile set* of size $mn$, such that there is a different tile type for each position in $[m] \times [n]$. Then, merge tiles types in some order to reduce the number of different tiles used. Using the auxiliary concepts and results above, we can formalise this approach as an exhaustive search in the set of all partitions of $[m] \times [n]$. Assume that we are given a fixed $k$-coloured pattern $c : [m] \times [n] \to [k]$.

The search starts from the initial partition $I = \{\{x\} \mid x \in [m] \times [n]\}$ that is clearly always constructible. In the beginning, we construct an MGTA for $I$, and then update it incrementally for every visited partition. We will define the set $C(P) \subseteq X$ of children of $P$ for every partition $P \in X$. The children are formed by merging classes of the partition that is currently being visited, and hence $P' \in C(P)$ implies $P' \sqsubseteq P$. When visiting a partition $P \in X$, the algorithms determines if it is constructible, and proceeds as follows:

C1. $P$ is constructible:

1. If $P(c) \not\sqsubseteq P$, we drop this branch of the search, since for all descendants $P' \sqsubseteq P$ we have $P(c) \not\sqsubseteq P'$, and hence $P'$ does not give us a solution.
2. If $P(c) \sqsubseteq P$, we can use the MGTA $f$ for $P$ to obtain a feasible solution $\mathscr{T} = (f(P), \mathcal{S}, s, 2)$ for the PATS problems instance $c$. Then, we consider partitions $P[p_1, p_2]$ where the classes $p_1, p_2 \in P$ are merged. If $p_1$ and $p_2$ are of different colour, we have $P(c) \not\sqsubseteq P[p_1, p_2]$, and thus it is enough to visit the following children of $P$:

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

C2. $P$ is not constructible: We can find classes $p_1, p_2 \in P$, $p_1 \neq p_2$ such that $\sigma_S(f(p_1)) = \sigma_S(f(p_2))$ and $\sigma_W(f(p_1)) = \sigma_W(f(p_2))$. Now we merge $p_1$ and $p_2$, that is, $C(P) = \{P[p_1, p_2]\}$.

It is not immediately clear that we do not lose any constructible partitions in the case C2. The following lemma shows that this does not happen.

**Lemma 9.** *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 $\sigma_S(f(p_1)) = \sigma_S(f(p_2))$ and $\sigma_W(f(p_1)) = \sigma_W(f(p_2))$. Then all constructible partitions $P' \sqsubseteq P$ satisfy $P' \sqsubseteq P[p_1, p_2]$.*

*Proof.* Let $g\colon P' \to \Sigma^4$ be the MGTA for $P'$. Since $P' \sqsubseteq P$, we can define a tile assignment $g'\colon P \to \Sigma^4$ such that for each $p \in P$ we have $g'(p) = g(q)$, where $q \in P'$ is the unique class with $p \subseteq q$. Now $g'$ satisfies condition A1, and thus condition A2 together with the assumption implies that $\sigma_S(g'(p_1)) = \sigma_S(g'(p_2))$ and $\sigma_W(g'(p_1)) = \sigma_W(g'(p_2))$. Since $P'$ is constructible, we cannot have $\sigma_S(g(q_1)) = \sigma_S(g(q_2))$ and $\sigma_W(g(q_1)) = \sigma_W(g(q_2))$ for any distinct classes $q_1, q_2 \in P'$. It follows that $p_1 \subseteq q$ and $p_2 \subseteq q$ for some $q \in P'$. This shows that $P' \sqsubseteq P[p_1, p_2]$. $\square$

Göös and Orponen [10] use the above framework to build their branch-and-bound algorithm, which we call PS-BB. They define a method to prune the search space of partitions so that no partition is visited twice, that is, the search structure is a tree. In addition, they give an efficient lower bound to the size of partitions that can be found in a given search branch. This makes it possible to drop certain branches that are not going to yield an optimal solution. Our approach is rather different, as we will see in the next section.

## 4 A Heuristic Partition-Search Algorithm

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 $p$ and $q$ such that $\sigma_N(f(p)) = \sigma_N(f(q))$ and $\sigma_E(f(p)) = \sigma_E(f(q))$, the glues on the west and south 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.

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

**Definition 10.** Given a partition $P$ and an MGTA $f$ for $P$, the *number of common glues* between classes $p, q \in P$ is defined by the function $G\colon P \times P \to [0, 4]$,

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

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

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.

1. Set $K \coloneqq H$.
2. Maximise the number of common glues:
$$K \coloneqq \{\{p, q\} \in K \mid G(p, q) \geq G(u, v) \text{ for all } \{u, v\} \in K\}.$$
3. Maximise the size of the larger class:
$$K \coloneqq \{\{p, q\} \in K \mid \max\{|p|, |q|\} \geq \max\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}.$$
4. Maximise the size of the smaller class:
$$K \coloneqq \{\{p, q\} \in K \mid \min\{|p|, |q|\} \geq \min\{|u|, |v|\} \text{ for all } \{u, v\} \in K\}.$$
5. Pick some pair $\{p, q\} \in K$ at random and visit the partition $P[p, q]$.
6. 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.

## 4.1 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. Our implementation of the PS-H algorithm is based on the DFS implementation of the PS-BB algorithm used by Göös and Orponen [10]. We consider several different finite 2-coloured input patterns, two of which were analysed also previously [10] using the PS-BB algorithm: the discrete Sierpinski triangles of sizes $32 \times 32$ (Figure 2(a)) and $64 \times 64$, and the binary counter of size $32 \times 32$ (Figure 2(b)). We introduce a 2-coloured "tree" pattern of size $23 \times 23$ (Figure 2(c)). Furthermore, as a potentially more practical example, we run our experiments on a 15-coloured pattern of size $20 \times 10$, which is based on a CMOS full adder design proposed in the context of carbon nanotube circuits [2, 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 3 presents the evolution of the "current best solution" as a function of time for the (a) $32 \times 32$ and (c) $64 \times 64$ Sierpinski triangle patterns. To allow fair comparison, Figures 3(b) and 3(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[1] conducted, the PS-H algorithm was able to find the optimal 4-tile solution for the $32 \times 32$ Sierpinski triangle pattern in less than 30 seconds. A similar percentage for the $64 \times 64$ 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 $32 \times 32$ and $64 \times 64$ 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 4 we present the corresponding results for the $32 \times 32$ binary counter and $23 \times 23$ 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 5(a)–5(b) show the results for the $20 \times 10$ 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.

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

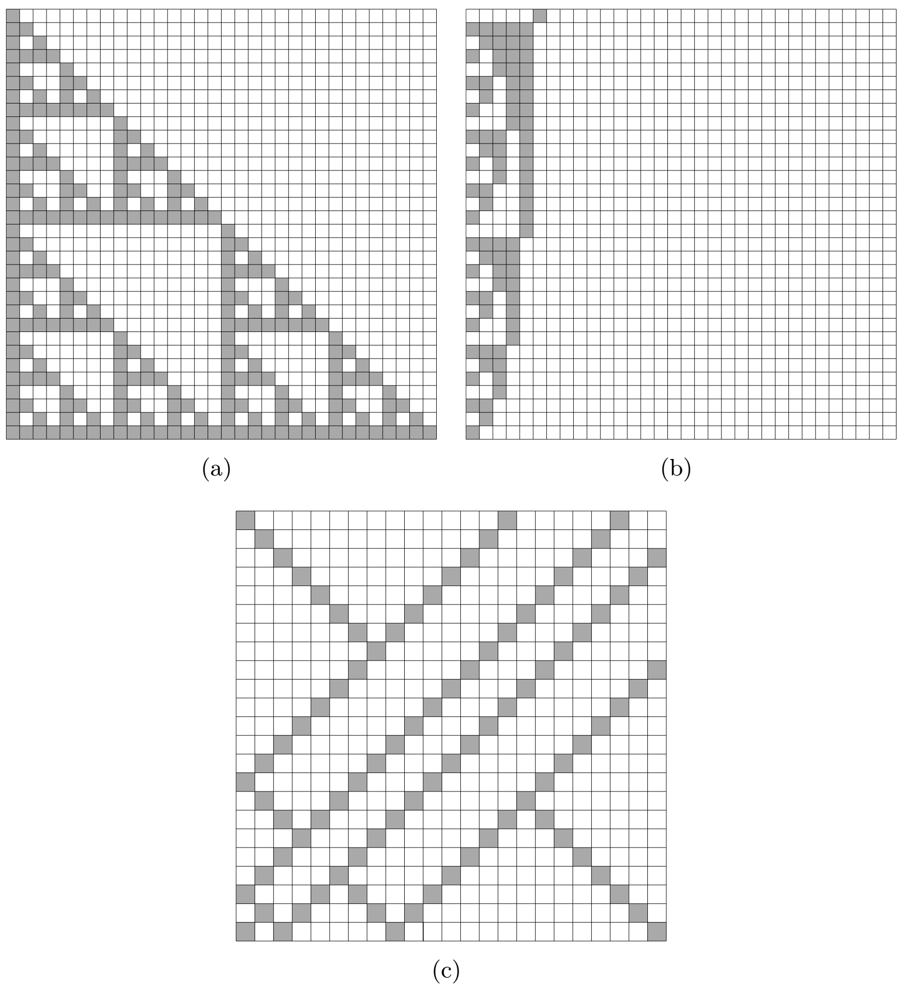


Figure 2: (a) The $32 \times 32$ Sierpinski triangle pattern. (b) The $32 \times 32$ binary counter pattern. (c) The $23 \times 23$ “tree” pattern.

## 5 Answer Set Programming

### 5.1 An ASP model for PATS

Answer Set Programming (ASP) [17] 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 [27] or GRINGO [7]. The ASP code is given in Listing 1.

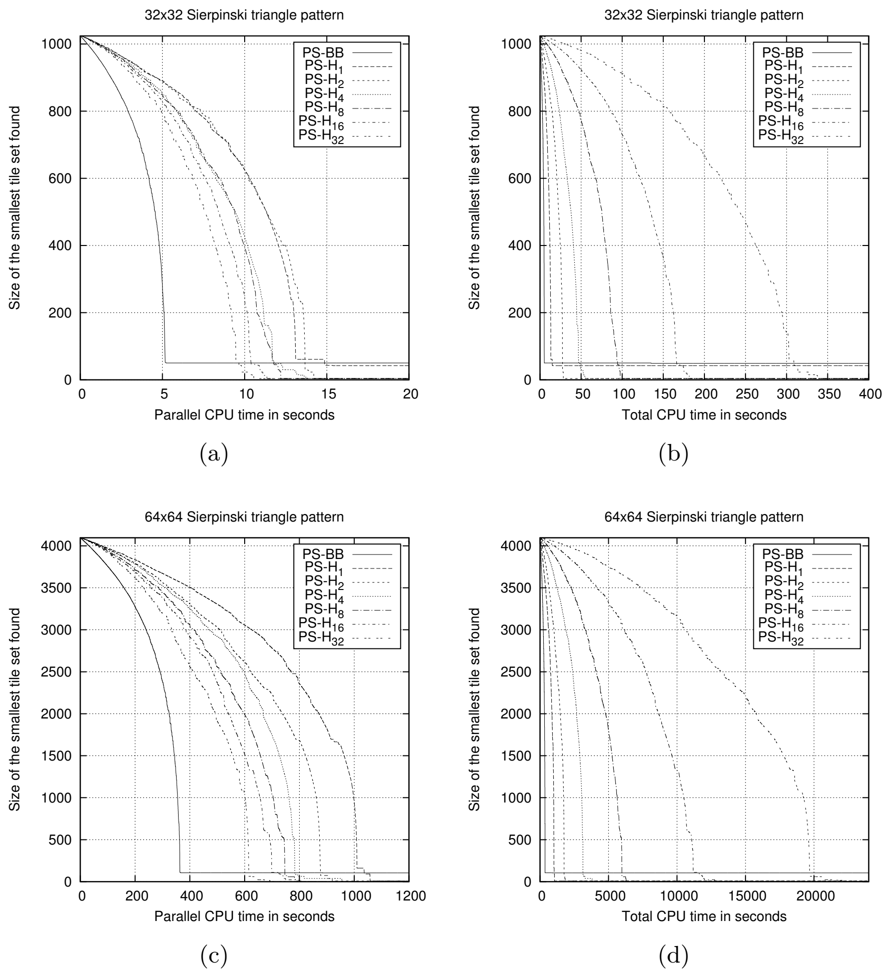
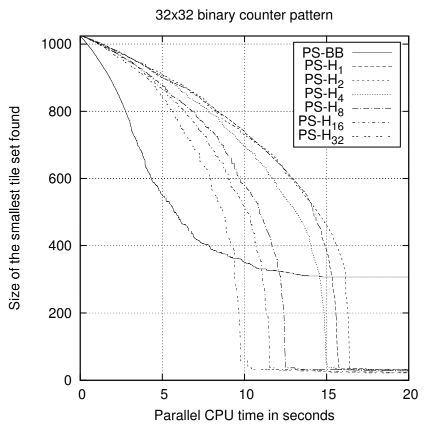
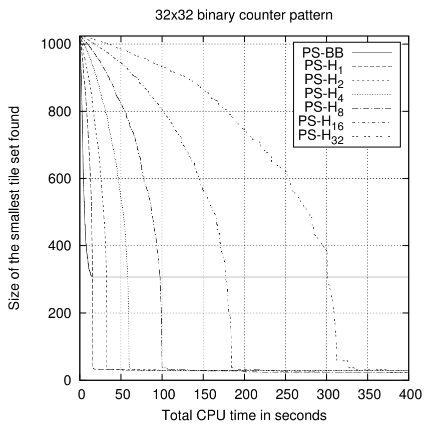


Figure 3: Evolution of the smallest tile set found for the $32 \times 32$ and $64 \times 64$ Sierpinski triangle patterns as a function of time. The time axes measure (a), (c) CPU time and (b), (d) CPU time multiplied by the number of parallel executions.

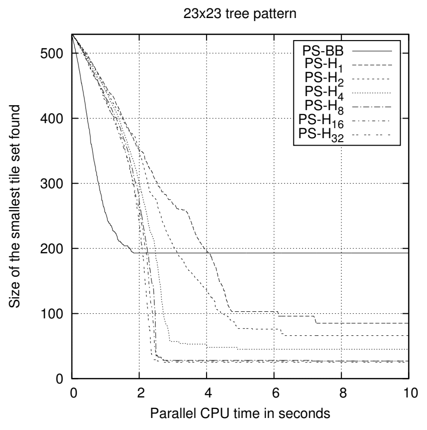
First, we define a constant for each position of the grid $[m] \times [n]$, each colour, each available tile type and each available glue type. After that, a number of choice rules are introduced to associate a tile type with each position of the grid, a glue type with each of the four sides of the tile 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 to make the tile system deterministic, that is, 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.



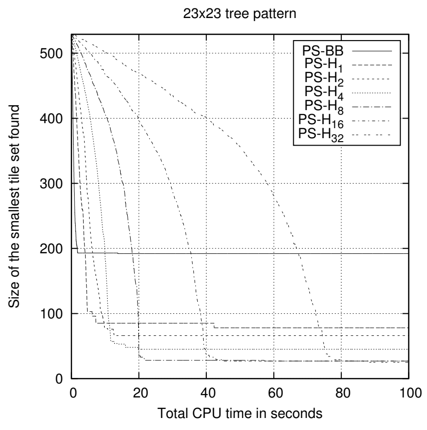
(a)



(b)



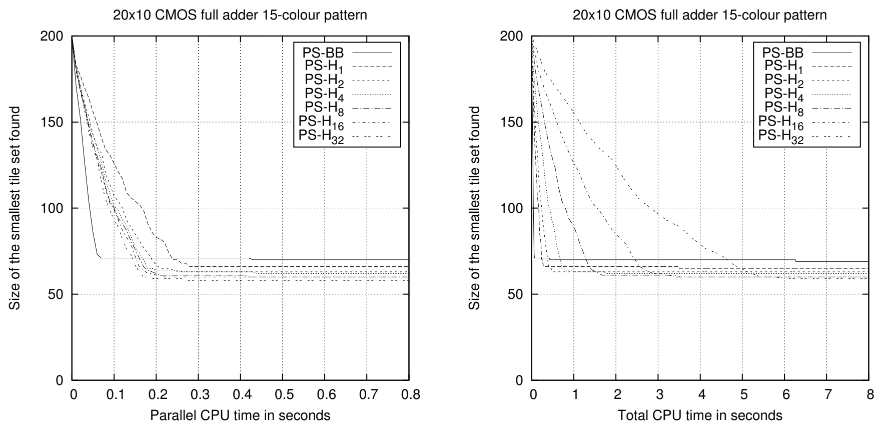
(c)



(d)

Figure 4: Evolution of the smallest tile set found for the $32 \times 32$ binary counter and $23 \times 23$ 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.

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)

Figure 5: Evolution of the smallest tile set found for the $20 \times 10$ 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.

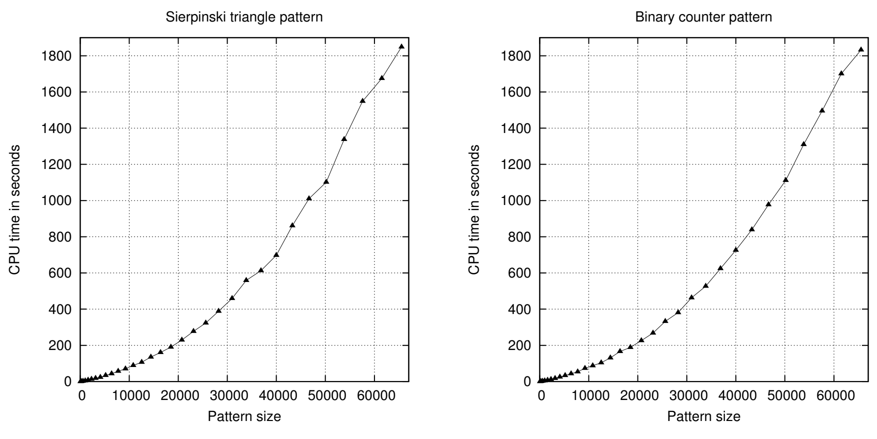
## 5.2 Results

We used grounder GRINGO 3.0.5 [7] and answer set solver CLASP 2.1.3 [8] with default settings to run our experiments. A traditional solver, SMODELS [20], 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 6(a) for the Sierpinski triangle pattern and in Figure 6(b) for the binary counter pattern. The results include the running time of both the grounder and 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 $256 \times 256$ Sierpinski triangle pattern and the $256 \times 256$ 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 4.1.

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 $23 \times 23$ tree pattern or the $20 \times 10$ CMOS full adder pattern using the ASP approach.

```
pos(1..x, 1..y).
col(1..k).
tile(1..t).
glue(1..g).

1 { pos_tile(X, Y, T) : tile(T) } 1 :- pos(X, Y).

1 { glue_w(T, G) : glue(G) } 1 :- tile(T).
1 { glue_s(T, G) : glue(G) } 1 :- tile(T).
1 { glue_n(T, G) : glue(G) } 1 :- tile(T).
1 { glue_e(T, G) : glue(G) } 1 :- tile(T).

1 { tile_c(T, C) : col(C) } 1 :- tile(T).

:- glue(Gw ; Gs), tile(T1 ; T2), T1 != T2, glue_w(T1, Gw), glue_w(T2, Gw),
    glue_s(T1, Gs), glue_s(T2, Gs).

glue_s(T2, G) :- pos_tile(X, Y, T1), pos_tile(X, Y+1, T2), glue_n(T1, G),
    glue(G), tile(T1 ; T2), pos(X, Y).
glue_w(T2, G) :- pos_tile(X, Y, T1), pos_tile(X+1, Y, T2), glue_e(T1, G),
    glue(G), tile(T1 ; T2), pos(X, Y).
```

Listing 1: An encoding of the PATS problem as an ASP instance.



(a) (b)

Figure 6: 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.

# 6 The Reliability of Tile Systems

In this section, we first review the kinetic Tile Assembly Model (kTAM) and then utilise it to develop a method to compute the reliability of a given tile system.

## 6.1 The Kinetic Tile Assembly Model

The kinetic Tile Assembly Model (kTAM) was introduced by Winfree [29, 31] to take into account the stochastic nature of self-assembly. In kTAM, reactions are allowed only between an assembly and a single tile. The reactions in question are *association* of new tiles to the existing assembly (forward reaction) and *dissociation* of tiles from the assembly (reverse reaction). It is assumed that association of tiles does not depend on the compatibility of glues on abutting edges; tiles can be attached to any position with at least one adjacent tile in place. The rate of the forward reaction is denoted by $r_f$ and it is proportional to the concentration of single tiles in the solution. The rate $r_{r,b}$, $b \in \{0, 1, 2, 3, 4\}$, of the reverse reaction, on the other hand, depends exponentially on the strength of the bonds formed by glues. Tiles that are connected to the assembly by fewer matching sticky ends have a higher probability of dissociation than those connected by stronger bonds.

For a tile type $t$, the rate $r_f$ of the association reaction of $t$ is

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

where $[t]$ is the concentration of single tiles of type $t$ in the solution and $k_f$ is a parameter that depends on temperature as well as the kind of tiles used. According to Winfree [31], the parameter is in the case of DNA double-crossover (DX) tiles given by

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

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). This is what we will use in our computations.

The rate of the dissociation reaction for a tile that is connected to the assembly by total bond strength of $b$ (in the sense of aTAM) is given by

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

where $\Delta G_b^o$ is the standard free energy that is needed to break $b$ bonds. In the case of DX tiles, where the glues are implemented using single-stranded DNA molecules having a length of 5 bases, we can estimate $\Delta G_b^o$ using the nearest-neighbour model [24] to be

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

Here $b = 0$ corresponds to the case where a tile has no bonds at all with the assembly, whereas for example $b = 2$ corresponds to a situation where a tile has formed bonds using two of its sticky ends.

To make handling of the model easier, the free parameters in the formulas of the rate constants $r_f$ and $r_{r,b}$ are re-distributed into just two dimensionless parameters, $G_{mc}$ and $G_{se}$. The first parameter depends on tile concentration at the beginning of the self-assembly process, while the second depends on the temperature. The rate constants can then be written as follows:

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

For DX tiles, $\hat{k}_f = e^3 k_f$ is needed in order to take into account possible entropic factors, such as orientation or location of tiles in the solution. This kind of re-distribution of parameters is possible, since in kTAM it is assumed that all tile types have initially similar concentrations and that the decrease of concentrations during the assembly process is negligible [31].

## 6.2 Computing the Reliability of a Tile System

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 [31]. 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 [31], 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 [31] gives us a continuous-time Markov process with 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 $r^* = r_f - r_{r,2}$. Thus, the overall off-rates for C, A and I are $r_{r,2} + r^*$, $r_{r,1} + r^*$ and $r_{r,0} + r^*$, respectively.

Let $p_S(t)$ denote the probability of the site being in state $S$ after $t$ seconds for all $S \in \{\mathrm{E}, \mathrm{C}, \mathrm{A}, \mathrm{I}, \mathrm{FC}, \mathrm{FI}\}$. We have $p_{\mathrm{E}}(0) = 1$ and $p_S(0) = 0$ for all $S \neq \mathrm{E}$, since the site is initially empty. To compute the frozen distribution, we write the rate equations for the model of kinetic trapping from Winfree [31] and obtain the following master equation:

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

where $M\mathbf{p}(t) = \dot{\mathbf{p}}(t)$, that is, the derivative of $\mathbf{p}$ with respect to time, and each element $M(i,i)$ represents the off-rate of the $i$th state, while each element $M(i,j)$, $i \neq j$, represents the transition rate from the $j$th state to the $i$th state.

To compute the steady-state probability of the site being frozen with the correct tile, that is, $p_{\mathrm{FC}}(\infty)$, we make use of the steady state of the related flow problem [31] by setting

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

that is, there is one unit of new material flowing into state E, and different amounts of it accumulate in states FC and FI. This gives us a system of linear equations. By solving it we obtain a value for $p_{\mathrm{FC}}(\infty)$, namely

$$p_{\mathrm{FC}}(\infty) = \frac{\frac{1}{r^* + r_{r,2}}}{\frac{1}{r^* + r_{r,2}} + \frac{M^1_{i,j}}{r^* + r_{r,1}} + \frac{M^2_{i,j}}{r^* + r_{r,0}}}.$$

This equals the probability of a tile with two matching glues being frozen on the site, that is,

$$\Pr(C_{i,j} \mid C_{i-1,j} \cap C_{i,j-1}) = p_{\mathrm{FC}}(\infty),$$

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, that is, the reliability of that pattern:

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

We have computed the probability in terms of $G_{mc}$ and $G_{se}$. Given the desired assembly 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

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

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

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

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

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

### 6.3 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 7(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.

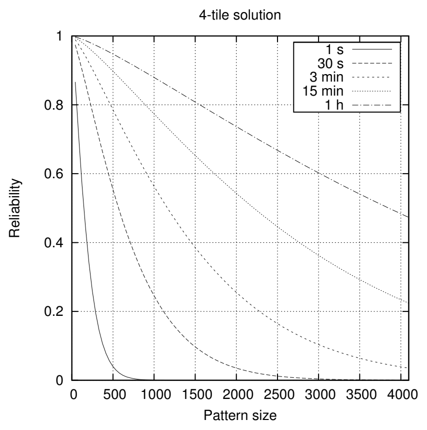
Figures 7(b)–7(d) present the reliability of tile sets found by the PS-H and PS-BB algorithms for the $32 \times 32$ Sierpinski triangle pattern, with assembly times of one hour, one day (24 hours) and one week. The runs were repeated 100 times; the mean reliability of each tile set size as well as the 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.

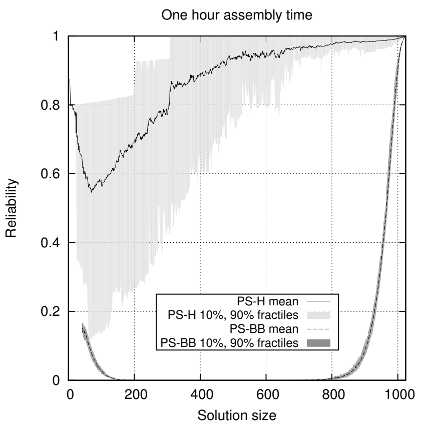
## 7 Conclusions

We have presented two new methods for constructing solutions to the PATS problem: the heuristic PS-H algorithm and the declarative ASP approach. Our experiments show that ASP is very efficient when the instance happens to have a solution consisting only of a few tile types, whereas the PS-H algorithm outperforms previous methods in a more general setting. We have also developed a method for estimating the reliability of tile sets, and found out that the PS-H approach is an improvement also in this respect.

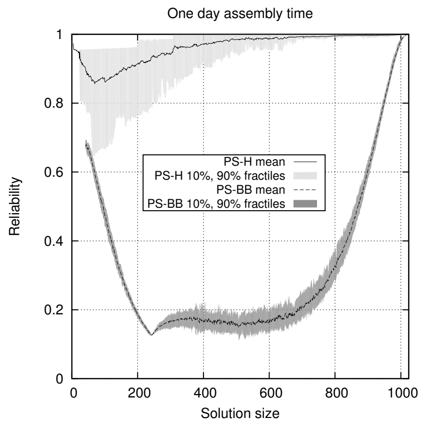
Future work could include developing polynomial-time algorithms with a guaranteed approximation ratio for the PATS problem. The declarative approach could probably be applied to instances with larger optimal solutions by developing a more efficient ASP or Boolean satisfiability encoding.



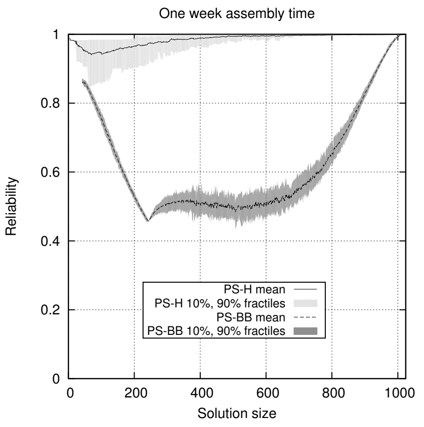
(a)



(b)



(c)



(d)

Figure 7: (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 $32 \times 32$ Sierpinski triangle pattern found by the PS-H and PS-BB algorithms, allowing assembly time of one hour, one day and one week.

## References

[1] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. In *Science* 266.5187 (1994), pages 1021–1024. DOI: 10.1126/science.7973651.

[2] Eugen Czeizler, Tuomo Lempiäinen and Pekka Orponen. A design framework for carbon nanotube circuits affixed on DNA origami tiles. In *Proceedings of the 8th Annual Conference on Foundations*

*of Nanoscience: Self-Assembled Architectures and Devices (FNANO 2011)*, pages 186–187. Poster abstract. 2011.

[3] Eugen Czeizler and Alexandru Popa. Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In *Proceedings of the 18th International Conference on DNA Computing and Molecular Programming (DNA 2012)*, pages 58–72. Springer, Berlin, Germany, 2012. DOI: 10.1007/978-3-642-32208-2_5.

[4] Eugen Czeizler and Alexandru Popa. Synthesizing minimal tile sets for complex patterns in the framework of patterned DNA self-assembly. In *Theoretical Computer Science* 499 (2013), pages 23–37. DOI: 10.1016/j.tcs.2013.05.009.

[5] David Doty. Theory of algorithmic self-assembly. In *Communications of the ACM* 55.12 (2012), pages 78–88. DOI: 10.1145/2380656.2380675.

[6] Kenichi Fujibayashi, Rizal Hariadi, Sung Ha Park, Erik Winfree and Satoshi Murata. Toward reliable algorithmic self-assembly of DNA tiles: a fixed-width cellular automaton pattern. In *Nano Letters* 8.7 (2008), pages 1791–1797. DOI: 10.1021/nl0722830.

[7] Martin Gebser, Roland Kaminski, Arne König and Torsten Schaub. Advances in *gringo* series 3. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*, pages 345–351. Springer, Berlin, Germany, 2011. DOI: 10.1007/978-3-642-20895-9_39.

[8] Martin Gebser, Benjamin Kaufmann, André Neumann and Torsten Schaub. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 386–392. AAAI, Menlo Park, California, USA, 2007.

[9] Mika Göös, Tuomo Lempiäinen, Eugen Czeizler and Pekka Orponen. Search methods for tile sets in patterned DNA self-assembly. In *Journal of Computer and System Sciences* 80.1 (2014), pages 297–319. DOI: 10.1016/j.jcss.2013.08.003.

[10] Mika Göös and Pekka Orponen. Synthesizing minimal tile sets for patterned DNA self-assembly. In *Proceedings of the 16th International Conference on DNA Computing and Molecular Programming (DNA 2010)*, pages 71–82. Springer, Berlin, Germany, 2011. DOI: 10.1007/978-3-642-18305-8_7.

[11] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, New York, NY, USA, 1987.

[12] Aleck C. Johnsen, Ming-Yang Kao and Shinnosuke Seki. Computing minimum tile sets to self-assemble color patterns. In *Proceedings of the 24th International Symposium on Algorithms and Computation (ISAAC 2013)*, pages 699–710. Springer, 2013. DOI: `10.1007/978-3-642-45030-3_65`.

[13] Aleck Johnsen, Ming-Yang Kao and Shinnosuke Seki. *A manually-checkable proof for the NP-hardness of 11-color pattern self-assembly tile set synthesis.* 2014. arXiv: `1409.1619 [cs.DM]`.

[14] Lila Kari, Steffen Kopecki, Pierre-Étienne Meunier, Matthew J. Patitz and Shinnosuke Seki. Binary pattern tile set synthesis is NP-hard. In *Proceedings of the 42nd International Colloquium on Automata, Languages, and Programming (ICALP 2015)*. 2015. Forthcoming.

[15] Lila Kari, Steffen Kopecki and Shinnosuke Seki. 3-color bounded patterned self-assembly. In *Proceedings of the 19th International Conference on DNA Computing and Molecular Programming (DNA 2013)*, pages 105–117. Springer, 2013. DOI: `10.1007/978-3-319-01928-4_8`.

[16] Tuomo Lempiäinen, Eugen Czeizler and Pekka Orponen. Synthesizing small and reliable tile sets for patterned DNA self-assembly. In *Proceedings of the 17th International Conference on DNA Computing and Molecular Programming (DNA 2011)*, pages 145–159. Springer, Berlin, Germany, 2011. DOI: `10.1007/978-3-642-23638-9_13`.

[17] Vladimir Lifschitz. What is answer set programming? In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 1594–1597. AAAI, Menlo Park, California, USA, 2008.

[18] Xiaojun Ma and F. Lombardi. On the computational complexity of tile set synthesis for DNA self-assembly. In *IEEE Transactions on Circuits and Systems II: Express Briefs* 56.1 (2009), pages 31–35. DOI: `10.1109/TCSII.2008.2010161`.

[19] Xiaojun Ma and Fabrizio Lombardi. Synthesis of tile sets for DNA self-assembly. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.5 (2008), pages 963–967. DOI: `10.1109/TCAD.2008.917973`.

[20] Ilkka Niemelä and Patrik Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1997)*, pages 420–429. Springer, Berlin, Germany, 1997. DOI: `10.1007/3-540-63255-7_32`.

[21] Sung Ha Park, Hao Yan, John H. Reif, Thomas H. LaBean and Gleb Finkelstein. Electronic nanostructures templated on self-assembled DNA scaffolds. In *Nanotechnology* 15.10 (2004), S525–S527. DOI: `10.1088/0957-4484/15/10/005`.

[22] Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. In *Nature* 440 (2006), pages 297–302. DOI: 10.1038/nature04586.

[23] Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 459–468. ACM, New York, NY, USA, 2000. DOI: 10.1145/335305.335358.

[24] John SantaLucia Jr., Hatim T. Allawi and P. Ananda Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. In *Biochemistry* 35.11 (1996), pages 3555–3562. DOI: 10.1021/bi951907q.

[25] Nadrian C. Seeman. Nucleic acid junctions and lattices. In *Journal of Theoretical Biology* 99.2 (1982), pages 237–247. DOI: 10.1016/0022-5193(82)90002-9.

[26] Shinnosuke Seki. Combinatorial optimization in pattern assembly. In *Proceedings of the 12th International Conference on Unconventional Computation and Natural Computation (UCNC 2013)*, pages 220–231. Springer, Berlin, Germany, 2013. DOI: 10.1007/978-3-642-39074-6_21.

[27] Tommi 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.

[28] Hao Wang. Proving theorems by pattern recognition – II. In *Bell System Technical Journal* 40.1 (1961), pages 1–41.

[29] Erik Winfree. Algorithmic Self-Assembly of DNA. PhD thesis. California Institute of Technology, 1998. URL: http://resolver.caltech.edu/CaltechETD:etd-05192003-110022.

[30] Erik Winfree. On the computational power of DNA annealing and ligation. In *Proceedings of a Mini DIMACS Workshop on DNA Based Computers (1995)*, pages 199–221. American Mathematical Society, Providence, RI, USA, 1996.

[31] Erik 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.

[32] Erik Winfree, Furong Liu, Lisa A. Wenzler and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. In *Nature* 394 (1998), pages 539–544. DOI: 10.1038/28998.