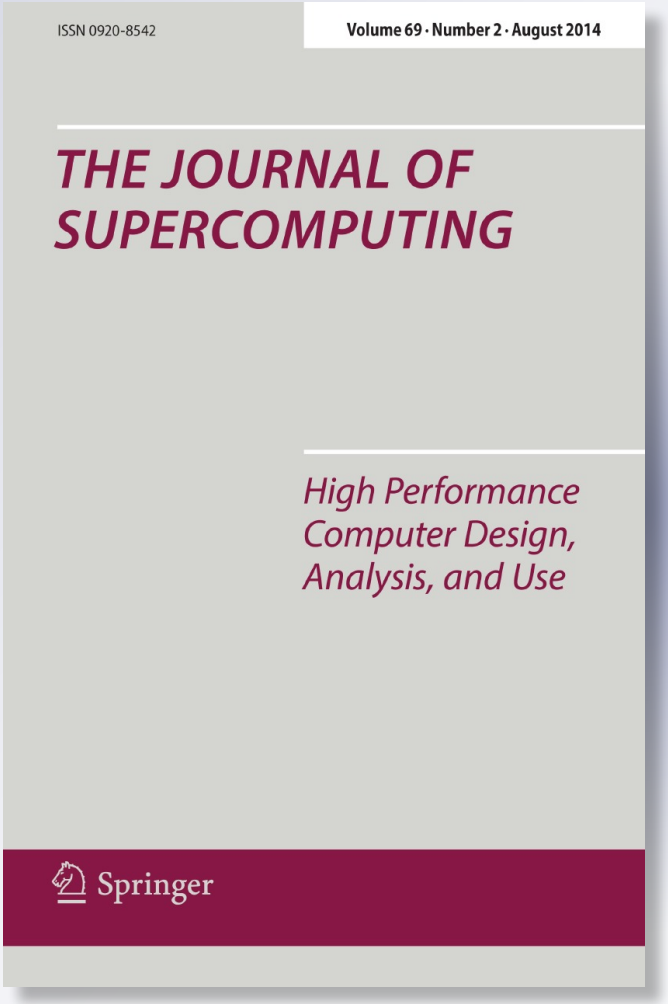
Molecular solutions for minimum and exact cover problems in the tile assembly model

Xu Zhou, YanTao Zhou, KenLi Li, Ahmed Sallam & Keqin Li

The Journal of Supercomputing
An International Journal of High-Performance Computer Design, Analysis, and Use

ISSN 0920-8542
Volume 69
Number 2

J Supercomput (2014) 69:976-1005
DOI 10.1007/s11227-014-1222-x



Springer

J Supercomput (2014) 69:976–1005
DOI 10.1007/s11227-014-1222-x

Molecular solutions for minimum and exact cover problems in the tile assembly model

Xu Zhou · YanTao Zhou · KenLi Li · Ahmed Sallam · Keqin Li

Published online: 28 June 2014

Abstract The tile assembly model is a novel biological computing model where information is encoded in DNA tiles. It is an efficient way to solve NP-complete problems due to its scalability and parallelism. In this paper, we apply the tile assembly model to solve the minimum and exact set cover problems, which are well-known NP-complete problems. To solve the minimum set cover problem, we design a MinSetCover system composed of three parts, i.e., the seed configuration subsystem, the nondeterministic choice subsystem, and the detection subsystem. Moreover, we improve the MinSetCover system and propose a MinExactSetCover system for solving the problem of exact cover by 3-sets. Finally we analyze the computation complexity and perform a simulation experiment to verify the effectiveness and correctness of the proposed systems.

Keywords NP-complete problem · Minimum set cover problem · Exact set cover problem · The tile assembly model · Parallel computing

1 Introduction

With the strength of parallel and high-density computing provided by molecules, DNA computing has been successfully used to solve many NP-complete problems such as

X. Zhou · Y. Zhou (✉) · KenLi Li · A. Sallam · Keqin Li
College of Information Science and Engineering, Hunan University, Changsha 410082, China
e-mail: yantao_z@hnu.edu.cn

X. Zhou
e-mail: happypanda2006@126.com

X. Zhou
College of Mathematics and Information Engineering, Jiaxing University, Jiaxing 314001, China

Keqin Li
Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

the RSA public-key cryptosystem [1,2], the discrete logarithm problem [3], the vertex coloring problem [4], the elliptic curve discrete logarithm problem [5], the classical Ramsey number problem [6], and the maximum clique problem [7]. Furthermore, the DNA algorithms for the clustering problem [8,9] and the decision making problem [10] have been proposed.

The tile assembly model is an important DNA computing model. Now, researchers are gradually paying more attention to the tile assembly model, because it is molecular-scalable, autonomous, and partially programmable [11–14]. Based on the mathematical concept of Wang tiles, Winfree et al. [11] first proposed an approach of computation by self-assembly tiles, and then developed a universal tile assembly system. In brief, the tile assembly model first uses the DNA tile's sticky ends to store a set of information and then processes this information by the DNA tile's self-assembly operations [12–14].

Since then, many biological systems have used the tile self-assembly model for solving NP-complete problems. Brun [15] showed the ability of DNA computing for arithmetic operations with the tile assembly model. Consequently, he was able to solve the subset sum problem [16], the factoring problem [17], and the satisfiability problem [18] using 2D DNA self-assembly tiles. Later, Zhang et al. [19] used the DNA self-assembly tiles to solve the graph coloring problem. Moreover, Cheng et al. [20] successfully solved the subset-product problem, and Cui et al. [21] also solved the maximum clique problem using the DNA tile assembly model.

On the other hand, the minimum set cover problem is a famous NP-hard combinatorial optimization problem. Recently, the authors of [22,23] proposed DNA computing algorithms to solve the minimum set cover problem. The proposed algorithms are all based on the sticker-based model's space solution and the Adleman–Lipton model's biological operations. Because of the model's limitations, the proposed algorithms have the disadvantages of high error rate, difficulty of operating, and the need of human intervention.

To the best of our knowledge, solving the minimum set cover problem with the tile assembly model has not been studied so far. In this paper, we utilize this model to solve two famous NP-complete problems, which are the minimum set cover problem and the problem of exact cover by 3-sets. Firstly, we propose a novel MinSetCover system for the minimum set cover problem. The MinSetCover system is composed of three parts, which are the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. Then we improve the present MinSetCover system and design a MinExactSetCover system in the tile assembly mode for solving the exact cover by 3-sets, which is also composed of three subsystems. Finally, we analyze the complexity of our proposed systems and do some experimental work to show our system's correctness and efficiency.

The rest of this paper is organized as follows. Section 2 introduces the tile assembly model in detail. Section 3 presents the MinSetCover system and its complexity analysis. Section 4 explains the MinExactSetCover system design and its complexity analysis. Section 5 shows the simulation experiment. The conclusions and future works are drawn in Sect. 6.

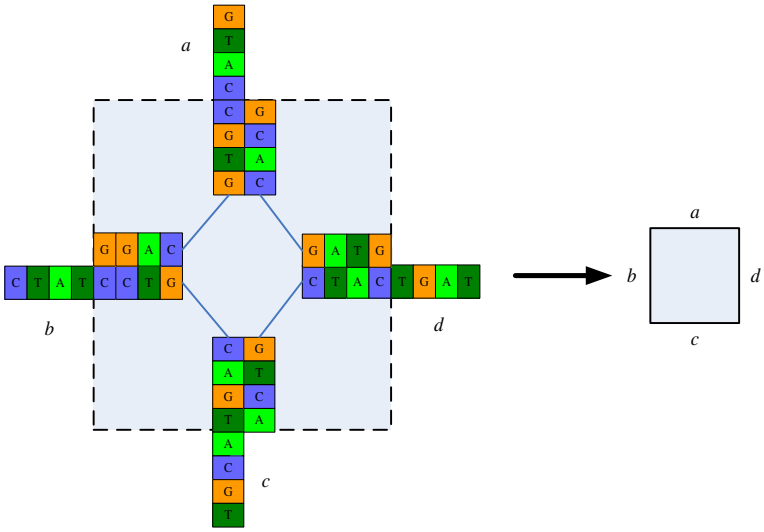


Fig. 1 DNA tile and its abstract representation

2 Tile assembly model

The tile assembly model is an extension of another model proposed by Wang [11], and was fully defined by Rothemund and Winfree. The tile assembly model which was designed to model self-assembly of molecules such as DNA is a formal model of crystal growth [11]. The definitions used here are similar to those defined in [16–18]. In this section, we present the DNA tile, the mathematical abstract model, and the relevant biological operations.

2.1 DNA tile

Different from other DNA computing models, in the tile assembly model information is encoded to different DNA tiles and then stored in the tiles' stick ends. Therefore, the DNA tiles can be implemented by double-crossover (DX) molecules with four sticky ends. These DNA tiles can stick together according to their four sides' binding domains. The four sticky ends of DX molecules can be encoded correspond to the labels on the four sides of the Wang tiles (see Fig. 1).

2.2 The mathematical abstract model

Let Σ be a finite alphabet representing the DNA tiles' binding domains where $null \in \Sigma$. We use the 4-triple $\langle \sigma_N, \sigma_E, \sigma_S, \sigma_W \rangle \in \Sigma^4$ to represent the four sticky ends of a tile in four directions. For convenience, we also denote an empty tile as $\langle null, null, null, null \rangle$ and there will be no tile sticking to the empty tile. A position is an element in the two-dimensional plane Z^2 and the set of the directions. A direc-

tion set $D = \{N, E, S, W\}$ is a set of 4 functions from one position to another; such that, $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$, $W(x, y) = (x - 1, y)$. Moreover, the positions (x, y) and $(\bar{x}, \bar{y})$ are neighbors if $\exists d \in D$ such that $d(x, y) = (\bar{x}, \bar{y})$. Also, If $d \in D$ for a tile t, we define $bd_d(t)$ as the binding domains of the tile t on the d's side and $bd_d(t) \in \Sigma$.

A strength function g: $\Sigma \times \Sigma \rightarrow R$, where g is commutative and R is the set of the natural numbers that denotes the strength of the binding domains. Let $\sigma, \sigma' \in \Sigma$, then g satisfies the following characteristics:

- $g(null, \sigma) = 0$
- if $g(\sigma, \sigma') = 0$, then $\sigma \neq \sigma'$
- $\forall \sigma \neq null$, then $g(\sigma, \sigma) = 1$
- $\forall \sigma \neq \sigma'$, then $g(\sigma, \sigma') = 0$

Let τ be the temperature such that for all σ if $g(\sigma, \sigma) = 1$ and $\tau = 2$ then a tile t can be attached only at positions with matching binding domains of other tiles in at least two adjacent positions.

Let T be a set of tiles. We define the configuration C of T as a function $A_C(x, y) : Z \times Z \rightarrow T$, which has the following characteristics: if $(x, y) \in Z^2$, and $A_C(x, y) \neq empty$ then $(x, y) \in C$; if there is only a finite number of different $(x, y) \in C$ then C is a finite set. Thus, if $C$is a configuration, then a tile t can be attached to C in a position (x, y) and generate a new configuration C' when the following conditions are satisfied:

- $(x, y) \notin C$
- $d \in D'$, $\overline{d} \in D$, $g(bd_d(t), bd_{d^{-1}}(A_C(\bar{d}(x, y)))) \geq \tau$
- $\forall (u, v) \in Z^2$, $(u, v) \neq (x, y)$, $A_{C'}(u, v) = A_C(u, v)$
- $A_{C'}(u, v) = t$

That is, a tile can attach to a configuration only at an empty position if and only if the total strength of the appropriate binding domains on the tiles in neighboring positions meets or exceeds the temperature τ.

Finally, according to the theoretical description of Wang, a tile system S can be denoted as a 3-triple $\langle T, g, \tau \rangle$, where T is the set of tiles, g is the energy function, τ is the temperature and $\tau \in \mathbb{N}$. Let *Seed* be the seed configuration that represents the input information. One may attach tiles of T to *Seed*, and then a series of different configurations are generated. After the repeated attachments of tiles, we gain a final configuration that contains the output information of our problem.

2.3 Tile assembly model's biochemical operations

In the tile assembly model, it needs the biological operations in the following:

(1) Amplify: Given a tube T, the operation, amplify (T, T_1, T_2), will produce two new tubes T_1 and T_2 by polymerase chain reaction (PCR), so that T_1 and T_2 are totally a copy of T (T_1 and T_2 are now identical) and T becomes empty tube. We usually make use of the amplify operation to get many copies of DNA tiles.

(2) Merge: Given tubes T_1 and T_2, the operation yields $\cup(T_1, T_2)$, where $\cup(T_1, T_2) = T_1 \cup T_2$. This operation is to pour two tubes into one, without any change in the individual strands.
(3) Extract: Given two test tubes T_1 and T_2, $T_2 = \text{extract}\ (T_1)$. This operation extracts DNA strands with specific features from the tube T_1 by the magnetic isolation method and put them into the tube T_2. In the tile assembly model, this operation is always used to extract the final configuration with special DNA tiles.
(4) Anneal: Given a test tube T, this operation is used to turn DNA molecules from single-stranded into double-stranded by lowering the temperature. This operation plays an important role in the process of self-assembly.
(5) Fluorescent labeling: Given a test tube T, this operation is used to identify the process of covalently attaching a fluorophore to the tiles.
(6) Agarose gel electrophoresis: Given a test tube T, this operation is used to separate the DNA molecules according to the molecular size.
(7) Read: Given a tube T, the operation can describe a single molecule contained in tube T by gene-sequencing technology such as chain termination method. Even if T contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them.

3 Solving the minimum set cover problem using tile assembly model

3.1 Definition of the minimum set cover problem

Minimum set cover problem (MinSCP) [22,23]: Given a finite set S, such that $S = \{s_1, s_2, \ldots, s_n\}$, and sets $C_1, C_2, \ldots, C_k$ are subsets of S. A set cover is a collection C of some of the sets from $C_1, C_2, \ldots, C_k$ whose union is the entire universe S. The minimum set cover problem is to find a minimum-size set cover for S.

For example, given a finite set $S\{1, 2, 3, 4\}$ and a collection $C\{\{1\}, \{2\}, \{1, 2\}, \{3, 4\}\}$. The minimum-size set cover for S is $\{\{1, 2\}, \{3, 4\}\}$.

It's worth mentioning that, finding a minimum-size set cover is a famous NP-complete problem [22], and it is widely used for applications such as fault diagnostic, pattern recognition, machine learning, and so on.

3.2 Solving minimum set cover problem using tile assembly model

For the sake of solving the minimum set cover problem with the tile assembly model, we should go through a few steps as follows.

Firstly, we design an initial subsystem to generate the seed configuration that stores the input information of our problem. In a seed configuration, we need to express all the elements of S and sets $C_1, C_2, \ldots, C_k$. For this purpose, we generate the tiles as shown in Fig. 2 and make multiple copies of them. Then we put these copies of tiles in a test tube by the merge operation. After the anneal operation, we get the seed configurations.

The second step is to design a nondeterministic choice subsystem to produce all the set cover schemes for our problem. To achieve this, we design the tiles shown in

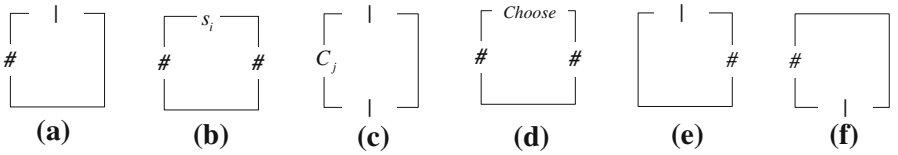


Fig. 2 The set of the seed tiles in the initial subsystem. **a** Tile Γ_{start}; **b** Tiles Γs_i where $0 \leq i \leq n$; **c** Tiles ΓC_j where $1 \leq j \leq k$; **d** Tile Γ_{Choose}; **e** tile Γ_{RowEnd}; **f** tile $\Gamma_{\text{ColumnEnd}}$

Fig. 3 The choice tiles in T_{Nondeter}. **a** Tiles ΓC_j^1 where $1 \leq j \leq k$; **b** Tiles ΓC_j^0 where $1 \leq j \leq k$; **c** Tile Γ_{Choose}^2

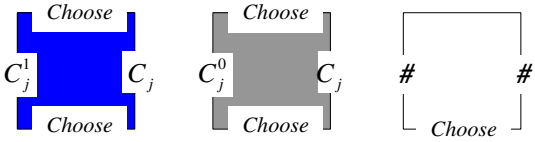


Fig. 3 where C_j^1 is used to express the situation that the set C_j is in the fixed set cover scheme and C_j^0 is used to describe the set C_j not in the fixed set cover scheme. By the amplify operation we make multiple copies of the tiles in Fig. 3. After the merge and the anneal operation, we get the solution configurations that express all the set cover schemes. Furthermore, the nondeterministic choice subsystem can get all the set cover schemes in parallel.

It is useful to identify the legal set cover schemes in a way that allows selecting the legal ones, so we design the detection subsystem. For some problems, only a small fraction of the exponentially many solution configurations would represent the results of our problem, and finding the small fraction would be difficult. Finding a successful computation by sampling the crystals at random would require time exponential in the input [24]. Thus, we will design a special identifier tile to identify the solution quickly. After applying the merge operation and the anneal operation, we can gain the solution configurations. By the fluorescent labeling operation and the agarose gel electrophoresis operation, we can separate the solution configurations according to the molecular size. By applying the exact operation, we can get the legal configurations. Lastly, the read operation is used to read out the results of our problem.

3.3 MinSetCover system implementation

Given a finite set $S\{s_1, s_2, \ldots, s_n\}$, and a collection $C\{C_1, C_2, \ldots, C_k\}$ where n is the number of elements in S and k is the number of sets in C. In this section, we introduce a MinSetCover system using the tile assembly model which is composed of the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. The three subsystems above are described in detail as follows.

3.3.1 The initial subsystem

To solve the MinSCP problem, we need to find out a collection C of some of the sets from $C_1, C_2, \ldots, C_k$ whose union is the entire universe S. Thus, every seed

configuration in the initial subsystem produced by the seed tiles should involve all the elements of S and the sets from C. In this subsystem, we encode the inputs as the L-configuration, in which the elements of the set S are encoded in the 0th row with a unique tile for each element and the sets $C_1, C_2, \ldots, C_k$ are encoded in the 0th column with a unique tile for each set. So the position $(-i-1, 0)$ represents the ith element of set S for $1 \leq i \leq n$, the position $(0, j)$ represents the jth set C_j of collection C where $1 \leq j \leq k$.

Figure 2 shows the seed tiles used by the initial subsystem. The tile Γ_{start} is used to express the start of the self-assembly. The tile $\Gamma s_i = \langle s_i, null, \#, \# \rangle$ is used to express the element s_i of the set S and it has the N, W and E sticky ends; where $bd_N(\Gamma s_i) = s_i$ and $1 \leq i \leq n$. The required number of the type of the tiles Γs_i is n.

The tile $\Gamma C_j = \langle |, |, C_j, null \rangle$ is applied to encode the set C_j of collection C where $bd_W(\Gamma C_j) = C_j$ and $1 \leq j \leq k$. The required number of the type of the tiles ΓC_j is k.

The tiles shown in Fig. 2c–f are four helper tiles which are the same for all n and k. The help tile $\Gamma_{\text{Choose}} = \langle Choose, null, \#, \# \rangle$ is used to help choosing the set C_j nondeterminately where $1 \leq j \leq k$.

Besides, the tiles Γ_{RowEnd} and $\Gamma_{\text{ColumnEnd}}$ are used to express the end of the assembly in the 0th column and the 0th row, respectively, and $\Gamma_{\text{RowEnd}} = \langle |, null, null, \# \rangle$, $\Gamma_{\text{ColumnEnd}} = \langle null, |, \#, null \rangle$.

As shown in Fig. 2, the number of types in T_{initial} is $O(n+k)$

Based on the design of the tiles above, we define the initial subsystem that encodes the input for the MinSCP problem as follows.

Theorem 3.3.1 *Let* $\Sigma_{\text{initial}} = \{s_i, C_j, \#, |\}$ *where* $1 \leq i \leq n$ *and* $1 \leq j \leq k$, T_{initial} *be the set of tiles over* Σ_{initial} *as shown in Fig. 2 where* $g_{\text{initial}} = 1$ *and* $\tau_{\text{initial}} = 2$. *Then the initial subsystem* $InitialSystem = \langle T_{\text{initial}}, g_{\text{initial}}, \tau_{\text{initial}} \rangle$ *generates the seed configuration.*

Proof Let T_{initial} be the set of tiles shown in Fig. 2 and $InitialSystem = \langle T_{\text{initial}}, g_{\text{initial}}, \tau_{\text{initial}} \rangle$. As shown in Fig. 2, the tiles in $InitialSystem$ are with the south and east sides as the input sides, the north and west sides as the output sides.

The $InitialSystem$ is designed to create the seed configuration for MinSetCover problem. The process of the self-assembly in the $InitialSystem$ is begin at the tile $\Gamma start$. Let the position of the tile $\Gamma start$ be (0, 0), then $N(0, 0) = (0, 1)$, $W(0, 0) = (-1, 0)$. As Fig. 2a shows, $bd_N(\Gamma start) = |$. According to the definition of function g, $g(bd_N(\Gamma start), bd_S(\Gamma C_j)) = 1$, after the anneal operation, the tiles ΓC_j will be attached to the tile $\Gamma start$ on its north side.

Consider the tiles that are in column 0 of the seed configuration. The tiles attached to the 0th column are used to encode the set C_j. Let C_{seed} be a seed configuration in the $InitialSystem$, $A_{C\text{seed}}(0, j) = \Gamma C_j$ and $bd_W(A_{C\text{seed}}(0, j)) = C_j$. Moreover, according to the set of tiles ΓC_j, bd_S, $(\Gamma C_j) = |$, where $1 \leq j \leq k$, the tiles that can attach in column 0, through positions (0, 1) to (0, k), are only the ΓC_j tiles, so $A_{C\text{seed}}(0, j) = \Gamma C_j$, where $1 \leq j \leq k$ and $A_{C\text{seed}}(0, k+1) = \langle null, |, \#, null \rangle$. After one of the boundary tile $\Gamma_{\text{ColumnEnd}}$, which is $\langle null, |, \#, null \rangle$, attached to the position (0, k+1), the self-assembly will end in the 0th column.

Consider the tiles that are included in the 0th row of the seed configuration. The tile $\Gamma Choose$ will be attached to the position $(-1, 0)$. Figure 2 shows, $bd_W(\Gamma start) = \#$. Then according to the seed tiles, $bd_W\ (\Gamma s_i) = \#$, so the only tiles Γs_i that can be attached in row 0, in positions $(-2, 0)$ through $(-n - 1, 0)$. That is $A_{C\text{seed}}\ (-i - 1, 0) = \Gamma s_i$ for $1 \leq\ i\ \leq\ n$, $A_{C\text{seed}}(-n - 2, 0)\ =\ \langle |, null, null, \# \rangle$. After one of the boundary tile Γ_{RowEnd}, which is $\langle |, null, null, \# \rangle$, attached to the position $(-n - 2, 0)$, the self-assembly will end in the 0th row.

Therefore, let C_{seed} be a seed configuration, then there are some position $(x, y) \in Z^2$ such that:

(1) $A_{C\text{seed}}(0, 0) = \Gamma_{\text{start}}$, where Γ_{start} is the start of the self-assembly.
(2) $A_{C\text{seed}}(-1, 0) = \Gamma_{\text{Choose}}$, where Γ_{Choose} is a helper tile.
(3) $A_{C\text{seed}}(-n - 2, 0) = \Gamma_{\text{RowEnd}}$, which means the end of self-assembly in row 0.
(4) $A_{C\text{seed}}(0, k+1) = \Gamma_{\text{ColumnEnd}}$, which means the end of self-assembly in column 0.
(5) For all $-n - 1 \leq x \leq -2$, $A_{C\text{seed}}(x, 0) = \Gamma s_i$, where Γs_i is used to encode the element of the set S.
(6) For all $1 \leq y \leq k$, $A_{C\text{seed}}(0, y) = \Gamma C_j$, where ΓC_j is used to encode the set C_j from the collection C.
(7) For all the other positions $(x, y) \in \mathbb{Z}^2$, $(x, y) \notin C_{\text{seed}}$.

3.3.2 The nondeterministic choice subsystem

The *NondeterChoiceSystem* is designed to create the solution configurations that express all the set cover schemes for MinSetCover problem based on the seed configurations. The nondeterministic choice subsystem *NondeterChoiceSystem* takes the seed configurations produced by the *InitialSystem* as its input. The main idea of the nondeterministic choice subsystem *NondeterChoiceSystem* is to have the choice tiles shown in Fig. 3 attach nondeterministically in column -1 of the seed configurations to select the sets $C_1, C_2, \ldots, C_k$.

The choice tiles shown in Fig. 3 are used to choose the set C_j nondeterministically and produce all the set cover schemes.

The blue tiles $\Gamma C_j^1 = \langle Choose, Choose, C_j^1, C_j \rangle$ are used to express the situation that the set C_j is in the fixed set cover scheme; where $bd_W(\Gamma C_j^1) = C_j^1$ and $1 \leq j \leq k$. The required number of the type of the tiles ΓC_j^1 is k.

The gray tiles $\Gamma C_j^0 = \langle Choose, Choose, C_j^0, C_j \rangle$ are applied to describe the set C_j not in the fixed set cover scheme, where $bd_w\ (bd_w(\Gamma C_j^0) = C_j^0$ and $1 \leq j \leq k$. The required number of the type of the tiles ΓC_j^0 is k.

The tile shown in Fig. 3c is a helper tile which is the same for all n and k. The help tile $\Gamma^2 Choose = \langle null, Choose, \#, \# \rangle$ is a boundary tile and used to end the assembly in column -1.

Based on the design of the tiles above, the number of types of the choice tiles is $O(k)$.

Theorem 3.3.2 *Let* $\Sigma_{\text{Nondeter}} = \{Choose, C_j^1, C_j^0, C_j, \#\}$, T_{Nondeter} *be the set of tiles over* Σ_{Nondeter} *as shown in Fig. 3 where* $g_{\text{Nondeter}} = 1$ *and* $\tau_{\text{Nondeter}} = 2$. *Then the nondeterministic choice subsystem* $NondeterChoiceSystem = \langle T_{\text{Nondeter}}, g_{\text{Nondeter}},$

$\tau_{\text{Nondeter}}\rangle$ *generates a non-certain solution space of the set cover problem and produce all the solution configurations that express the set cover schemes.*

Proof Let $NondeterChoiceSystem = \langle T_{\text{Nondeter}}, g_{\text{Nondeter}}, \tau_{\text{Nondeter}}\rangle$. As shown in Fig. 3, the tiles in *NondeterChooseSystem* are with south and east sides as the input sides, north and west sides as the output sides.

Note that $g_{\text{Detect}} = 1$, $\tau_{\text{Detect}} = 2$ and the seed configurations generated by the initial subsystem in the shape of L. A tile can only be attached to the seed configurations when its south and east neighbors exist, and if its appropriate binding domains match those neighbors' appropriate binding domains.

By Theorem 3.3.1, the initial system *InitialSystem* can generate the seed configuration. The seed configuration in the *InitialSystem* is in the shape of a horizontally reflected L. The tiles in the row 0 encode all the elements of the set S and the tiles in the column 0 express all the sets C_j of the collection C.

In the following, we examine the procedure of the self-assembly in the system *NondeterChoiceSystem*. According to the condition that a tile t which is attached to the configuration C to produce a new configuration.

$$d \in D', \bar{d} \in D, g(bd_d(t), bd_{d^{-1}}(A_C(\bar{d}(x, y)))) \geq \tau$$

So the position that the first tile can attach to a seed configuration is at the position $(-1, 1)$ and then growth is allowed along the north direction and west direction, respectively.

Let C_{seed} represent a seed configuration. Consider the tiles that can be attached to column -1. Since $bd_W(A_{C\text{seed}}(0, j)) = C_j$, where $1 \leq j \leq k$, the tiles that can stick to column -1, in position $(-1, 1)$ through $(-1, k)$, should be with the east binding domain C_j. Because $bd_E(\Gamma C_j^1) = bd_E(\Gamma C_j^0) = C_j$, the tiles ΓC_j^1 and ΓC_j^0 can be attached to C_{seed} nondeterministically in column -1, in position $(-1, 1)$ through $(-1, k)$, where $1 \leq j \leq k$.

Because the seed configuration has a north binding domain *Choose*, by induction, the tiles with $\sigma_N = \sigma_S = Choose$ can be attached. After the tiles ΓC_j^1 or ΓC_j^0 are attached, the *Choose* binding domain will propagate to the north of the tile in position $(-1, j)$. By attaching the tile ΓC_j^1 or ΓC_j^0, respectively, it is nondeterministic to choose the set C_j. The nondeterministic choice subsystem can get all the set cover schemes parallelly.

Let C_{Solution} be a solution configuration. So there are some position $(x, y) \in Z^2$ such that:

(1) For all $1 \leq y \leq k$, $A_{C\text{Solution}}(-1, y) = \Gamma C_j^1$ or ΓC_j^0, where ΓC_j^1 is used to express the situation that the set C_j is in the fixed set cover scheme, ΓC_j^0 is used to express the situation that the set C_j is not in the fixed set cover .
(2) $A_{C\text{Solution}}(-1, k + 1) = \Gamma^2 Choose$, where $\Gamma^2 Choose$ is a helper tile.

3.3.3 The detection subsystem

The detection subsystem is designed to find out the legal configurations. Taking the solution configurations produced by the *NondeterChoiceSystem* as its input, the main

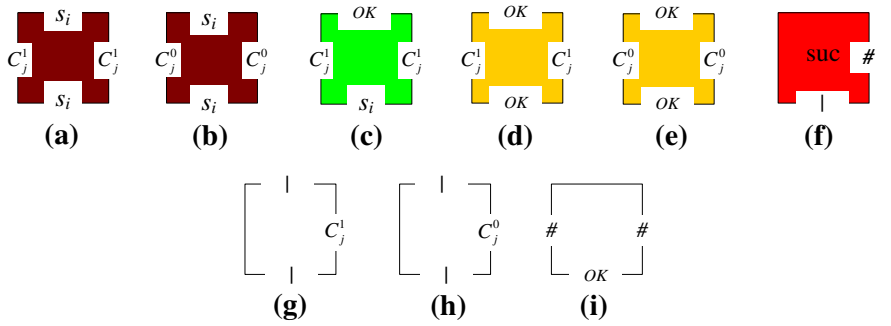


Fig. 4 The detection tiles in T_{Detect} **a–e** Tiles Γ_{detect}; **f** Tiles Γ_{suc}; **g–h** Tiles Γ_{output}; **i** Γ^2_{end}

idea of the detection subsystem *DetectSystem* is to have detection tiles shown in Fig. 4 attach nondeterministically to the solution configurations to generate the final configurations and identify the legal configurations.

The detection tiles shown in Fig. 4 are used to drive the decision whether a solution configuration, which is produced by the *NondeterChoiceSystem*, meets the conditions. In more details, we use the detection tiles showed in Fig. 4 to determine whether the union of a collection C of some of the sets from $C_1, C_2, \ldots, C_k$ is the entire universe S.

The scarlet tiles $\langle s_i, s_i, C_j^1, C_j^1\rangle$ and $\langle s_i, s_i, C_j^0, C_j^0\rangle$ represent that the set C_j does not cover the element s_i. The two tiles imply that there is no association between the set C_j and the element s_i, where $1 \leq i \leq n; 1 \leq j \leq k$. The number of these tile types is$n \times k$.

The green tile $\langle OK, s_i, C_j^1, C_j^1\rangle$ denotes that the set C_j covers the element s_i, $\sigma_N = OK, \sigma_N \neq \sigma_S$ and $\sigma_W = \sigma_E$. The Label OK is used to express that we have find out one set from the collection C which covers the element s_i. The required number of the green tiles is $n \times k$.

The two types of orange tile $\langle OK, OK, C_j^1, C_j^1\rangle$ and $\langle OK, OK, C_j^0, C_j^0\rangle$ show that one element s_i has been covered by some set from the collection C, $\sigma_N = \sigma_S = OK, \sigma_W = \sigma_E$ and the required number of tiles is k.

The red *SUC* tile $\langle null, |, null, \#\rangle$ is a special identifier tile to identify the legal configuration which are the same for all n and k.

The two white tiles $\langle |, |, null, C_j^1\rangle$ and $\langle |, |, null, C_j^0\rangle$ are the tiles which store the output information, $\sigma_N = \sigma_S = |, \sigma_E = C_j^0$ or C_j^1. The white tile $\langle null, OK, \#, \#\rangle$ is a help tile that is the same for all k. The required number of the white tiles equals $2k + 1$.

Therefore, the number of the type of the detection tiles is $O(n \times k)$.

Theorem 3.3.3 *Let* $\Sigma_{\text{Detect}} = \{s_i, C_j^1, C_j^0, OK, |, \#\}$, T_{Detect} *be the set of tiles over* Σ_{Detect} *as shown in Fig. 4 where* $g_{\text{Detect}} = 1$ *and* $\tau_{\text{Detect}} = 2$. *Then the detection subsystem* $DetectSystem = \langle T_{\text{Detect}}, g_{\text{Detect}}, \tau_{\text{Detect}}\rangle$ *takes the solution configurations generated by the NondeterChoiceSystem as its input and then produces the final configurations and identifies the legal configuration with the red SUC tile as the identifier tile.*

Proof Let $DetectSystem = \langle T_{\text{Detect}}, g_{\text{Detect}}, \tau_{\text{Detect}} \rangle$. In the following, we examine the procedure of the self-assembly in the system *DetectSystem*. According to the condition that a tile t which is attached to the configuration C to produce a new configuration.

$$d \in D', \bar{d} \in D, g(bd_d(t), bd_{d^{-1}}(A_C(\bar{d}(x, y)))) \geq \tau$$

So the position that the first tile can attach to a solution configuration is the $(-2, 1)$. Let C_{Solution} represent the solution configuration generated by the system *NondeterChoiceSystem*. The system *DetectSystem* can produce the final configurations by attaching the detection tiles to the solution configurations.

Note that, the detection tiles that can be attached to column -3 and column $-n-1$, are similar to the column -2. We examine the tiles that can be attached to column -2, in position $(-2, 1)$ through $(-2, k+1)$. Because in column -1, in position $(-1, 1)$ through $(-1, k+1)$, the west binding domains of the tiles are C_j^i where $1 \leq j \leq k$ and $i \in \{0, 1\}$, only the tile $t, bd_E(t) = C_j^i$, can be attached to the column -2.

Hence, if $bd_E(t) = C_j^0$ there are two kinds of tiles, $\langle s_i, s_i, C_j^0, C_j^0 \rangle$ and $\langle OK, OK, C_j^0, C_j^0 \rangle$ that can be attached to column -2. If the element s_i of the set S does not exist in the subset C_j or the subset C_j does not exist in the fixed set cover scheme, the tile $\langle s_i, s_i, C_j^0, C_j^0 \rangle$ will be attached to column -2. When the element s_i of the set S has been covered and the subset C_j does not exist in the fixed set cover scheme, the information '*OK*' will be delivered from button to the top by the tile $\langle OK, OK, C_j^0, C_j^0 \rangle$.

When $bd_E(t) = C_j^1$, there are three kinds of tiles, $\langle OK, s_i, C_j^1, C_j^1 \rangle$, $\langle s_i, s_i, C_j^1, C_j^1 \rangle$ and $\langle OK, OK, C_j^1, C_j^1 \rangle$ that can be attached to the column -2. The tile $\langle OK, s_i, C_j^1, C_j^1 \rangle$ will be attached to column -2 when the subset C_j^1 exists in the solution for the set cover problem and it also covers the element s_i of the set S. If the subset C_j covers the element s_i of the set S, the tile $\langle s_i, s_i, C_j^1, C_j^1 \rangle$ will be attached to column -2. When the element s_i of the set S has been covered and the subset C_j exists in the fixed set cover scheme, the information '*OK*' will be delivered from button to the top by the tile $\langle OK, OK, C_j^1, C_j^1 \rangle$.

Consider the tiles that can be attached to column $-n-2$. Since $bd_N(A_{C\text{seed}}(-n-2, 0)) = |$ The tiles that can stick to column $-n-2$, in position $(-n-2,1)$ through $(-n-2, k)$, should be with the south binding domain |. The white tile t with $bd_N(t) = bd_S(t) = |$ and $bd_E(t) = C_j^i$, where $1 \leq j \leq k$ and $i \in \{0, 1\}$ can be attached. By induction, after the tiles $\langle |, |, null, C_j^i, \rangle$ attaching, the | binding domain will propagate to the north of the tile in position $(-n-2, k)$.

Consider the tiles that can be attached to row $k+1$. Since $bd_W(A_{C\text{seed}}(-1, k+1)) = \#$, the tiles that can stick to row $k+1$ in position $(-2, k+1)$ through $(-n-2, k+1)$ should be with the east binding domain #. By induction, the white tile t with $bd_W(t) = bd_E(t) = \#$ and $bd_S(t) = OK$ can be attached. These tiles will propagate the # binding domain to the west of tile in position $(-n-1, k+1)$.

The *SUC* tile has an east binging domain # and south binding domain |. Thus, it can only be attached to position $(-n-2, k+1)$ and only if the west binding domain

of the tile in position $(-n-1, k+1)$ is #, so the tile can be attached if all the elements in the set S are covered by the fixed set cover scheme.

Since every possible set cover scheme will be explored nondeterministically, if every element in the set S is covered by the fixed set cover scheme, then the *SUC* tile will be attached. If there is one or more element in the set S not covered by the fixed set cover scheme, then the *SUC* tile will never be attached. The *SUC* tile can help us to identify the legal final configurations.

Let C_{Final} be a legal final configuration. Then there are some $(x, y) \in Z^2$ such that:

(1) For all $1 \leq y \leq k$, $A_{C\text{Final}}(-n-2, y) = \Gamma_{\text{output}}$ where Γ_{output} store the output information.
(2) For all $-n-1 \leq x \leq -1$, $1 \leq y \leq k$, $A_{C\text{Final}}(x, y) = \Gamma_{\text{detect}}$ where the tiles in Γ_{detect} are used to detect the configurations.
(3) For all $-n-1 \leq x \leq -1$, $A_{C\text{Final}}(x, k+1) = \Gamma^2_{\text{end}}$ where Γ^2_{end} means the end of the self-assembly in row $k+1$
(4) $A_{C_{\text{Final}}}(-n-2, k+1) = \Gamma_{\text{suc}}$ where Γ_{suc} is used to identify the legal configurations.
(5) For all the other positions $(x, y) \in \mathbb{Z}^2$, $(x, y) \notin C_{\text{Final}}$.

3.4 Complexity analysis of the MinSetCover system

The complexity of the presented system using the tile assembly model is commonly measured in terms of the assembly time, the computation space, and the types of tiles [16–18]. In this paper, we also analyze the successful rate of the proposed MinSetCover system.

Lemma 3.4.1 *The assembly time of the proposed set cover system is $O(k)$ and the space complexity is $O(n \times k)$ where n is the number of elements in S and k is the number of sets in C.*

Proof The proposed set cover system is composed of three parts which are the initial subsystem, the nondeterministic choice subsystem and the detection subsystem. The initial subsystem generates the seed configurations that are also L-configuration. The length of the button row is $n+3$ and the height of the rightmost column is $k+2$. Taking in account all the tiles in T_{detect}, we can get that the pairs $(bd_S(t), bd_E(t))$ are unique. Hence, by attaching the tiles from T_{detect} to a seed configuration, there will be a unique final configuration produced. The final configurations that contain the red *SUC* tile $\langle null, |, \#, null \rangle$ with the are the complete $(n+3) \times (k+2)$ rectangle. The complete rectangle's depth is $k+2$, so the assembly time is $O(k)$. The area of the complete rectangle is $(n+3) \times (k+2) = nk + 2n + 3k + 6$ so the space complexity is $O(n \times k)$.

Lemma 3.4.2 *For all the $n \in N$ and $k \in N$, we assume that each tile that can be attached to a configuration at a certain position with a uniform probability distribution. The probability that a single nondeterministic execution of the proposed set cover system succeeds in attaching a SUC tile is at least 0.5^k where k is the number of sets in C.*

Proof Only the tiles in column -1 are attached nondeterministically, and at each of these positions there are exactly two choices of tile t, which are $\langle Choose, Choose, C_j^0, j\rangle$ and $\langle Choose, Choose, C_j^1, j\rangle$. So the probability of the correct tile to be attached at each location is 0.5 and there are exactly k places where such tiles can be attached. Thus, if there exist an answer of the set cover problem, at least 0.5^k of the assemblies attaching a *SUC* tile.

Lemma 3.4.3 *The proposed set cover system uses $O(n \times k)$ distinct tiles where n is the number of elements in S and k is the number of sets in C.*

Proof The proposed set cover system consists of three parts, which are the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. The initial subsystem needs $n+k+4$ distinct tiles, the nondeterministic choice subsystem needs $2q+1$ distinct tiles, and the detection subsystem needs $2n \times k + 4 \times k + 2$ distinct tiles. Hence, the proposed set cover system needs $(n+k+4)+(2k+1)+(2nk+4k+2) = 2n \times k + 7 \times k + n + 7$ distinct tiles. Therefore, the proposed set cover system uses $O(n \times k)$ distinct tiles.

4 Solving the problem of exact cover by 3-sets using the tile assembly model

The minimum 3-set exact cover problem is also a famous NP-complete problem. In this section, we introduce a new solution for this problem using the tile assembly model.

4.1 Definition of the 3-set exact cover problem [23]

Assume S is a finite set, $S = \{s_1, s_2, \ldots, s_3n\}$ and s_i is an element in S for $1 \le i \le 3n$. We denote as the cardinality of S by $|S|$ such that $|S| = 3n$. Suppose that a collection C is $\{C_1, C_2, \ldots, C_k\}$, where C_j is a subset of S for $1 \le j \le k \quad \le j \le \quad k$ and Cj contains three elements. We denote the cardinality of C by $|C|$ such that $|C| = k$.

Minimum 3-set extract cover problem: A 3-set extract cover for S is a subcollection $C_{\text{sub}} \subseteq C$ and $C_{\text{sub}} = \{C'_1, C'_2, \ldots, C'_m\}$ where $|C_{\text{sub}}| = m$. On the assumption that $\cup C'_i = S$, C_{sub} is a 3-set extract cover for S and every element in S occurs in exactly one member of C_{sub}. The minimum 3-set exact cover problem is to find a minimum-size 3-set exact cover for S.

For example, a finite set S is $\{1, 2, 3, 4, 5, 6\}$ and a collection C is $\{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{4, 5, 6\}\}$ for S. The minimum-size 3-set exact cover for S is $\{\{1, 2, 3\}, \{4, 5, 6\}\}$. It's worth mentioning that finding a minimum-size 3-set exact cover is proved to be a NP-complete problem in [23].

4.2 Solution implementation

In this section, we design a MinExactSetCover system using the tile assembly model which is also composed of the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. The seed configuration and the nondetermin-

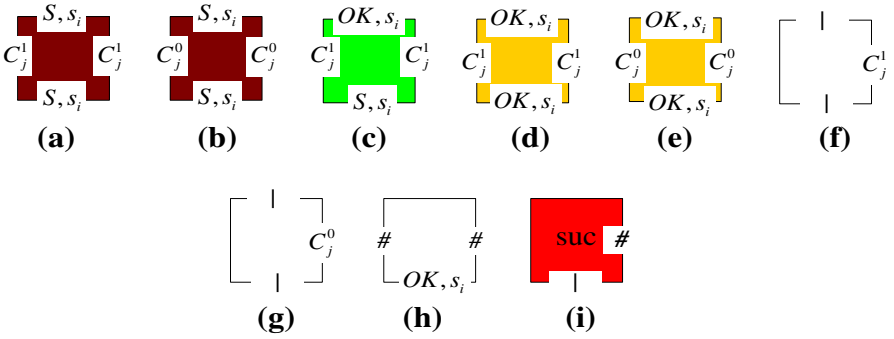


Fig. 5 The detection tiles in $T_{\text{ExtractDetect}}$ **a–e** Tiles $\Gamma_{\text{Extractdetect}}$; **f–g** Tiles $\Gamma_{\text{Extractoutput}}$; **i** $\Gamma_{\text{Extractsuc}}$

istic choice subsystem are similar to the corresponding subsystem of the proposed MinSetCover system. In the initial subsystem of MinSetCover system, it uses the tile $\langle s_i, null, \#, \# \rangle$ to represent the different element in set S while the tile $\langle (S, s_i), null, \#, \# \rangle$ is used to represent the different element in the corresponding subsystem of the MinExactSetCover system. The nondeterministic choice subsystems of the MinSetCover system and the MinExactSetCover system are the same. Furthermore, we design a new detection subsystem to identify the legal configurations in the following.

The detection subsystem is also designed to find out the legal configurations. Taking the solution configurations produced by the *NondeterChoiceSystem* as its input, the main idea of the detection subsystem *ExtractDetectSystem* is to have detection tiles shown in Fig. 5 attach nondeterministically to the solution configurations to generate the final configurations and identify the legal configurations.

The detection tiles shown in Fig. 5 is used to drive the decision whether a solution configuration, which produced by the *NondeterChoiceSystem*, meets the conditions.

The scarlet tiles $\langle (S, s_i), (S, s_i), C_j^1, C_j^1 \rangle$ and $\langle (S, s_i), (S, s_i), C_j^0, C_j^0 \rangle$ express that the set C_j does not cover the element s_i. The two tiles imply that there is no association between the set C_j and the elementss_i, where$1 \leq i \leq 3n$; $1 \leq j \leq k$. The number of these tile types is 3 $n \times k$.

The green tile $\langle (S, s_i), (OK, s_i), C_j^1, C_j^1 \rangle$ denotes that the set C_j covers the element s_i. The Label OK is used to express that we have find out one set from the collection C which covers the element s_i. The required number of the green tiles is $3n \times k$.

The two types of orange tile $\langle (OK, s_i), (OK, s_i), C_j^1, C_j^1 \rangle$ and $\langle (OK, s_i), (OK, s_i), C_j^0, C_j^0 \rangle$ show that one element s_i has been covered by some set from the collection C and the set C_j does not contain the element s_i, and the required number of tiles is $3n \times k$.

The two white tiles $\langle |, |, null, C_j^1 \rangle$ and $\langle |, |, null, C_j^0 \rangle$ are the tiles which store the output information. The white tile $\langle null, (OK, s_i), \#, \# \rangle$ is a help tile that is the same for all k. The required number of the white tiles equals $2k + 3n$.

The red *SUC* tile $\langle null, |, null, \# \rangle$ is a special identifier tile to identify the legal configuration which are the same for all n and k.

Therefore, the number of types of the detection tiles is $O(n \times k)$.

Lemma 4.2.1 *Let* $\Sigma_{\text{ExtractDetect}} = \{S, s_i, C_j^1, C_j^0, OK, |, \#\}$, $T_{\text{ExtractDetect}}$ *be the set of tiles over* $\Sigma_{\text{ExtractDetect}}$ *as shown in Fig. 5 where* $g_{\text{ExtractDetect}} = 1$ *and* $\tau_{\text{ExtractDetect}} = 2$. *Then the new detection subsystem* $ExtractDetectSystem = \langle T_{\text{ExtractDetect}}, g_{\text{ExtractDetect}}, \tau_{\text{ExtractDetect}} \rangle$ *identifies the legal configurations and produce the final configurations.*

Proof Let $T_{\text{ExtractDetect}}$ be the set of tiles shown in Fig. 5 where $g_{\text{ExtractDetect}} = 1$, $\tau_{\text{ExtractDetect}} = 2$, and $ExtractDetectSystem = \langle T_{\text{ExtractDetect}}, g_{\text{ExtractDetect}}, \tau_{\text{ExtractDetect}} \rangle$.

Note that $g_{\text{ExtractDetect}} = 1$, $\tau_{\text{ExtractDetect}} = 2$ and the seed configurations generated by the initial subsystem are in the shape of L. A tile can only be attached to the seed configurations when its south and east neighbors exist, and if its appropriate binding domains match those neighbors' appropriate binding domains.

By Theorem 3.3.1, the initial system *InitialSystem* can generate the seed configuration. The seed configuration in the *InitialSystem* is in the shape of a horizontally reflected L. The tiles in row 0 encode all the elements of the set S and the tiles in column 0 express all the sets C_j of the collection C. By Theorem 3.3.2, the *NondeterChoiceSystem* takes the seed configurations produced by the *InitialSystem* as its input and is designed to create the solution configurations that express all the extract set cover schemes for MinExactSetCover problem based on the seed configurations.

Let $C_{\text{ExactSolution}}$ represent the solution configuration generated by the system *NondeterChoiceSystem*. The system *ExtractDetectSystem* can produce the final configurations by attaching the detection tiles to the solution configurations. In the following, we examine the procedure of the self-assembly in the system *ExtractDetectSystem*.

Consider the tiles that attached to column -3 to column $-n-1$. The tiles which can be attached to column -3 to column $-n-1$ are similar to the column -2. We examine the tiles that can be attached to column -2, in position $(-2, 1)$ through $(-2, k+1)$. Because in column -1, position $(-1, 1)$ through $(-1, k+1)$, the west binding domains of the tiles is C_j^i where $1 \leq j \leq k$ and $i \in \{0,1\}$, only the tile t, $bd_E(t) = C_j^i$ can be attached to the column -2.

Hence, when $bd_E(t) = C_j^1$ there are three kinds of tiles, $\langle (S, s_i), (S, s_i), C_j^1, C_j^1 \rangle$, $\langle (OK, s_i), (S, s_i), C_j^1, C_j^1 \rangle$ and $\langle (OK, s_i), (OK, s_i), C_j^1, C_j^1 \rangle$ which can be attached to column -2. The tile $\langle (OK,s_i), (S, s_i), C_j^1, C_j^1 \rangle$ can be attached to column -2 when the subset C_j exists in the solution for the 3-set extract cover and it also covers the element s_i of the set S. If the C_j covers the elementss_i of the set S, after the tile $\langle (OK, s_i), (S, s_i), C_j^1, C_j^1 \rangle$ attached to column -2, the information (OK, s_i) will be delivered from button to top by the tile $\langle (OK, s_i), (OK, s_i), C_j^1, C_j^1 \rangle$.

If $bd_E(t) = C_j^0$ there are two kinds of tiles, $\langle (S, s_i), (S, s_i), C_j^0, C_j^0 \rangle$ and $\langle (OK,s_i), (OK, s_i), C_j^0, C_j^0 \rangle$ which can be attached to column -2. If the element s_i of the set S does not exist in C_j, or C_j does not exist in the solution for the set cover problem for $1 \leq i \leq 3n$, $1 \leq j \leq k$, after the tile $\langle (S, s_i), (S, s_i), C_j^0, C_j^0 \rangle$ or $\langle (OK,s_i), (OK,s_i), C_j^0, C_j^0 \rangle$ attached to column -2, the information (S, s_i) or (OK, s_i) will be delivered from button to top.

Consider the tiles that can be attached to column $-3n-2$. Since $bd_N(A_{C\text{ExtractDetect}}(-3n-2, 0)) = |$. The tiles that can stick to column $-3n-2$ in position $(-3n-2, 1)$

through $(-3n-2, k)$ should be with the south binding domain |. The white tile t with $bd_N(t) = bd_S(t) =$ | and $bd_E(t) = C_j^i$, where $1 \leq j \leq k$ and $i \in \{0,1\}$ can be attached. By induction, after the tiles $\langle |, |, null, C_j^i, \rangle$ attaching, the | binding domain will propagate to the north of the tile in position $(-3n-2, k)$.

Consider the tiles that attached to row $k+1$. Because $bd_W(A_{\text{CExtractDetect}}(0, k+1)) = \#$, the tile t with $bd_W(t) = bd_E(t) = \#$ and $bd_S(t) = (OK, s_i)$ can be attached to row $k+1$. By induction, the tile t with $bd_W(t) = bd_E(t) = \#$ will propagate the # binding domain to the west of tile in position $(-3n-1, k+1)$.

The *SUC* tile has an east binging domain # and south binding domain |; thus, it can only be attached in position $(-3n-2, k+1)$ and only if $bd_W(A_{\text{CExtractDetect}}(-3n-1, k+1)) = \#$ and $bd_N(A_{\text{CExtractDetect}}(-3n-2, k)) =$ |, so the *SUC* tile can be attached if all the elements in the set S are covered by the fixed set cover scheme.

Since every possible set cover scheme will be explored nondeterministically, if every element in the set S is covered by the fixed set cover scheme only once, the *SUC* tile will be attached. Otherwise, if there is one or more element in the set S that is not covered by the fixed set cover scheme or covered more than once, then the *SUC* tile will never be attached. The *SUC* tile can help us to identify the legal final configurations.

Let $C_{\text{ExtractDetect}}$ be a legal final configuration. So there are some $(x, y) \in Z^2$ such that:

(1) For all $1 \leq y \leq k$, $A_{C\text{ExtractDetect}}(-3n-2, y) = \Gamma_{\text{ExtractOutput}}$ where $\Gamma_{\text{ExtractOutput}}$ stores the output information for MinExactSetCover
(2) For all $-3n-1 \leq x \leq -1$ and $1 \leq y \leq k$, $A_{C\text{ExtractDetect}}(x, y) = \Gamma_{\text{Extractdetect}}$ where the tiles in $\Gamma_{\text{Extractdetect}}$ are used to detect the configurations.
(3) For all $-3n-1 \leq x \leq -1$, $A_{C\text{ExtractDetect}}(x, k+1) = \langle null, (OK, s_i), \#, \# \rangle$
(4) $A_{C\text{ExtractDetect}}(-3n-2, k+1) = \Gamma_{\text{ExtractSuc}}$ where $\Gamma_{\text{ExtractSuc}}$ contains the *SUC* tile $\langle null, |, \#, null \rangle$.
(5) For all other positions $(x, y) \in \mathbb{Z}^2$, $(x, y) \notin C_{ExtractDetect}(x, y)$.

4.3 Complexity analysis of the MinExtractSetCover system

In this section, we will analyze the assembly time, the computation space, and the distinct tiles of the proposed MinExtractSetCover system. Besides, we will also analyze the solution space and the successful rate of the proposed set cover system.

Lemma 4.3.1 *The assembly time of the proposed MinExtractSetCover system is $O(k)$ and the space complexity is $O(n \times k)$ where 3n is the number of elements in S and k is the number of sets in C.*

Proof The proof is similar to that of Theorem 3.3.1.

Lemma 4.3.2 *For all the $n \in N$ and $k \in N$, we assume that each tile that may be attach to a configuration at a certain position with a uniform probability distribution. The probability that a single nondeterministic execution of the proposed MinExtract-SetCover system succeeds in attaching a SUC tile is at least 0.5^k where k is the number of sets in C.*

Proof The proof is similar to that of Theorem 3.3.2.

Lemma 4.3.3 *In the proposed 3-set extract cover system, it uses $O(n \times k)$ distinct tiles where n is the number of elements in S and k is the number of sets in C.*

Proof The proposed set cover system consists of three parts which are the seed configuration subsystem, the nondeterministic choice subsystem and the detection subsystem. The seed configuration subsystem needs $3n + k+4$ distinct tiles, the nondeterministic choice subsystem needs $2k+1$ distinct tiles and the detection subsystem needs $6nk+4k+2$ distinct tiles. Hence, the proposed set cover system needs $(3n + k+4)+(2k+1)+(6nk+4k+2) = 6nk+7k + 3n+3$ distinct tiles. Thus, the proposed set cover system uses $O(n \times k)$ distinct tiles.

5 Experiments

In this section, we will simulate the proposed systems in Sects. 3 and 4. According to the methods of Winfree's [11] and Brun's [13–18] to simulate the tile assembly system, we divide our experiments into three steps: firstly, to design the DNA tiles used; secondly, to explain the experiment procedure; and finally, to utilize Xgrow Simulator [24,25] developed by Winfree's research group to test the effectiveness of our systems.

5.1 Solving the minimum set cover problem

Consider that a finite set S is $\{1, 2, 3, 4\}$ and a collection C is $\{\{1\}, \{2\}, \{1, 2\}, \{3, 4\}\}$. Assume s_i is an element in S for $1 \leq i \leq 4$ and the set $C_j \subseteq C$ where $1 \leq j \leq 4$. In this section, we will simulate the proposed MinSetCover system.

5.1.1 Tile code for MinSetCover problem

The MinSetCover system is composed of the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. The tiles used in the three subsystems are designed, respectively, as follows.

To encode the elements of the set S on the horizontal row and the sets C_j on the vertical column, we design nineteen initialization tiles which are illustrated in Fig. 6. Consider our example of the MinSetCover problem, there are twelve tiles also used in the initial subsystem (Fig. 6).

The nondeterministic choice subsystem uses three types of tile which are shown in Fig. 3. For our example of the MinSetCover problem, there are nine tiles used in the nondeterministic choice subsystem (Fig. 7). The blue tiles are used to express the situation that the set C_j is in the set cover scheme and the gray tiles are applied to describe the set C_j not in the set cover scheme.

The detection subsystem uses nine types of tile which are shown in Fig. 4. Based on the tile's application, we divide the tiles of the detection subsystem to three groups as follows.

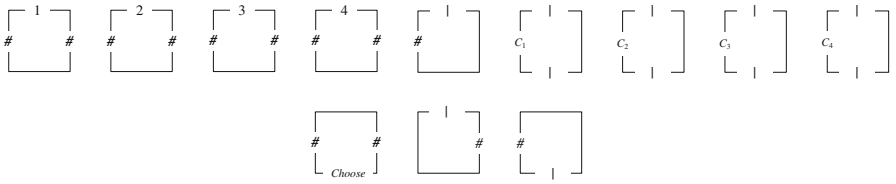


Fig. 6 The tiles used in the initial subsystem for our problem

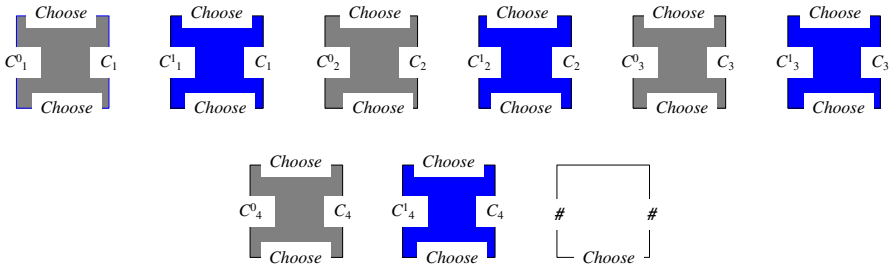


Fig. 7 The tiles used in the nondeterministic choice subsystem for our problem

In the first group, there are some tiles used to detect whether or not C_j covers the element s_i for $s_i \subseteq \{1, 2, 3, 4\}$ and $C_j \subseteq \{\{1\}, \{2\}, \{1, 2\}, \{3, 4\}\}$. For our example of the MinSetCover problem, we have the tiles shown in Fig. 8a. Shown in Fig. 8a, the crimson tiles represent C_j does not cover s_i and the green tiles represent C_j covers s_i for $s_i \subseteq \{1, 2, 3, 4\}$ and $C_j \subseteq \{\{1\}, \{2\}, \{1, 2\}, \{3, 4\}\}$.

In the second group, there are some tiles used to deliver the covering state of the elements s_i for $s_i \subseteq \{1, 2, 3, 4\}$. As shown in Fig. 8b, the yellow tiles are used to transit the covering state of s_i for $s_i \subseteq \{1, 2, 3, 4\}$.

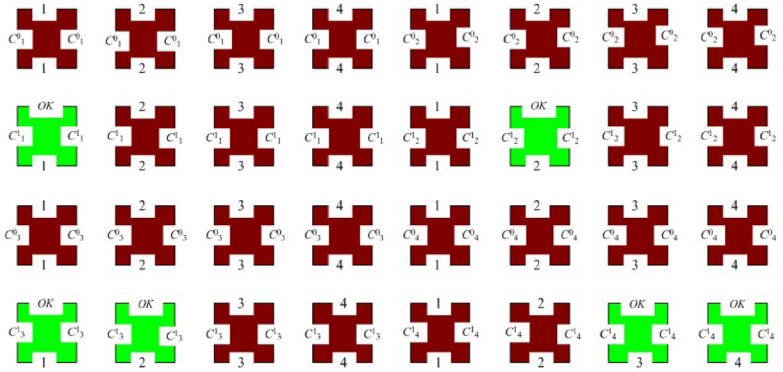
In the last group, as shown in Fig. 8c there are some white tiles used as the boundary tiles and there is a red tile with value *SUC* used to identify the legal configurations.

5.1.2 MinSetCover solution procedure

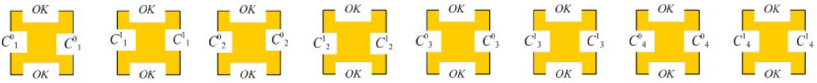
The MinSetCover system proposed here is composed of initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. In the following, we will describe the process of our problem by the MinSetCover system.

Firstly, by running the initial subsystem, we will get the seed configuration for our problem (see Fig. 9). The seed configuration is a L-Configuration and the 0th row mainly encode the elements s_i of the set S for $1 \leq i \leq 4$, the 0th column is mainly encode the sets C_j for $1 \leq j \leq 4$.

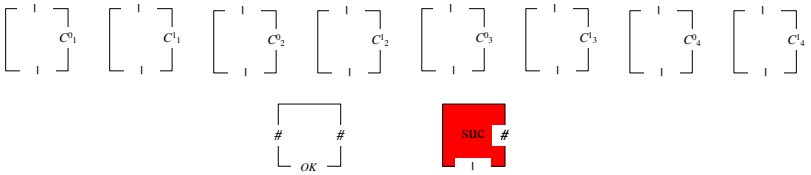
Secondly, after obtaining the seed configurations, we will execute the nondeterministic choice subsystem and choose C_j by attaching the tiles shown in Fig. 7 to the seed configurations nondeterminately. Thus, we will get all the solution configurations which represent the solution space for our problem. Figure 10 shows two of the solution configurations which represent the solution $\{C_3, C_4\}$ and $\{C_1, C_4\}$, respectively.



(a) The tiles used to detect whether or not C_j covers the element s_i for $s_i \subseteq \{1, 2, 3, 4\}$ and $C_j \subseteq \{\{1\}, \{2\}, \{1, 2\}, \{3, 4\}\}$



(b) The yellow tiles used to deliver the covering state of the element s_i for $s_i \subseteq \{1, 2, 3, 4\}$



(c) The white tiles used as the boundary tiles and the red tile used to identify the legal configurations

Fig. 8 The tiles used in the detection subsystem of our problem

Lastly, the detection subsystem is used to identify the legal configurations and get the legal solution space of our problem. The final configurations are generated by the detection subsystem and the configurations with red tile $\langle null, |, null, \# \rangle$ with value *SUC* are the legal configurations and represent the satisfiable solutions for the MinSetCover problem. As shown in Fig. 11, the final configurations represent the solution $\{C_3, C_4\}$ and $\{C_1, C_4\}$, respectively, Where the final configuration for $\{C_3, C_4\}$ contains a red tile $\langle null, |, null, \# \rangle$ with value *SUC*. Thus, $\{C_3, C_4\}$ is the satisfiable solution for our problem.

5.1.3 MinSetCover system simulation

In this section, we will use the Xgrow Simulator [24] developed by Winfree's research group to test the effectiveness of the proposed MinSetCover system.

Figure 12a shows the solution configuration that represents the collection $\{C_3, C_4\}$ and after attaching the corresponding DNA tiles we get final configuration shown in Fig. 12b. Because the red *SUC* tile involved in the final configuration of collection $\{C_3, C_4\}$, the collection $\{C_3, C_4\}$ is a satisfiable answer of our problem. Figure 12c shows the solution configuration of the collection $\{C_1, C_4\}$ and its final configura-

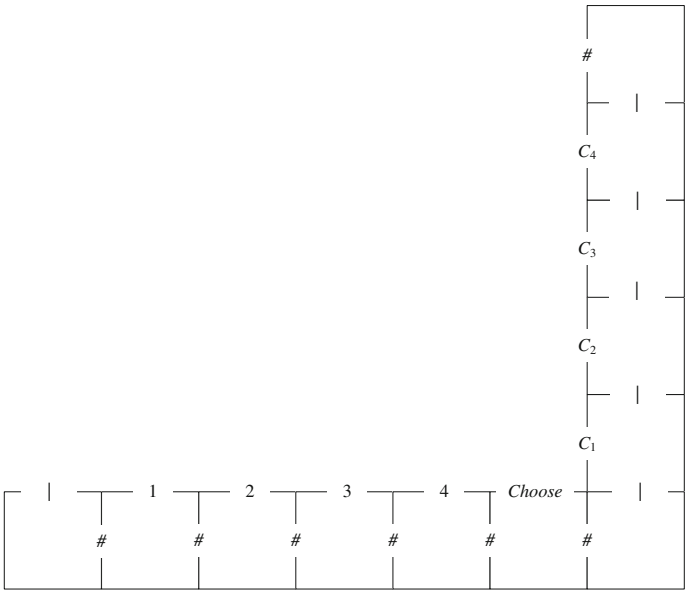


Fig. 9 The seed configuration generated by the initial subsystem for our problem

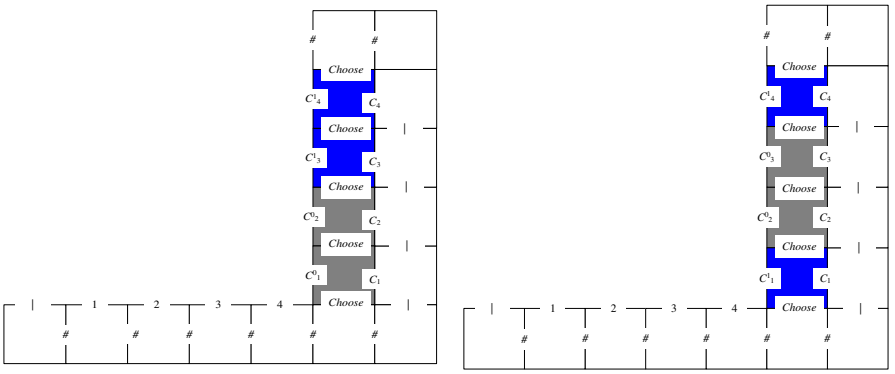


Fig. 10 The solution configuration for $\{C_3, C_4\}$ and $\{C_1, C_4\}$, respectively

tion shown in Fig. 12d and does not contain the red *SUC* tile. Hence, the collection $\{C_1, C_4\}$ is not an answer of our problem.

5.2 Solving the extract cover by 3-sets problem

Assume we have a finite set S $\{1, 2, 3, 4, 5, 6\}$, a collection C is $\{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{4, 5, 6\}\}$ where s_i is an element in S for $1 \leq i \leq 6$ and the set $C_j \subseteq C$ for $1 \leq j \leq 4$. In this section, we will simulate the proposed MinExtractSetCover system.

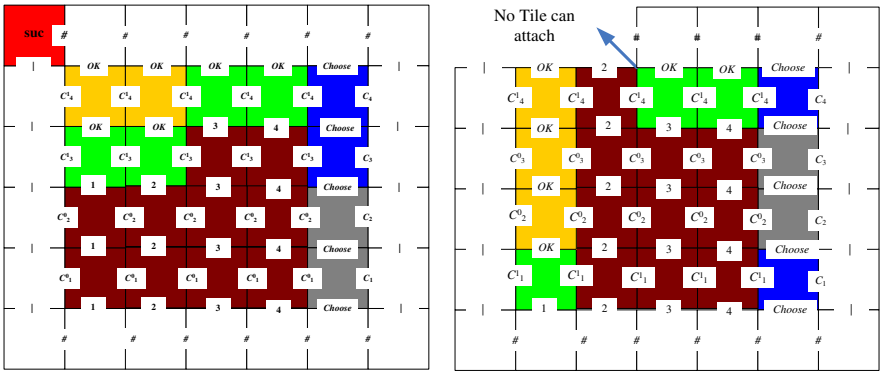
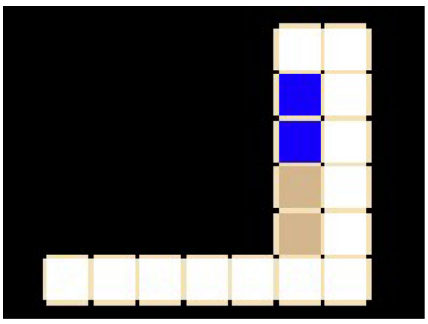
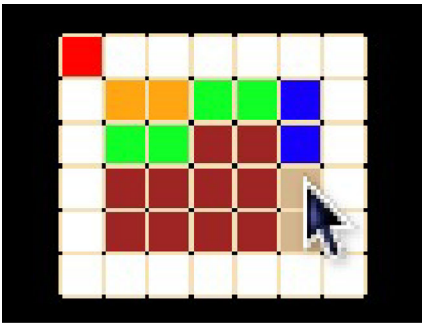


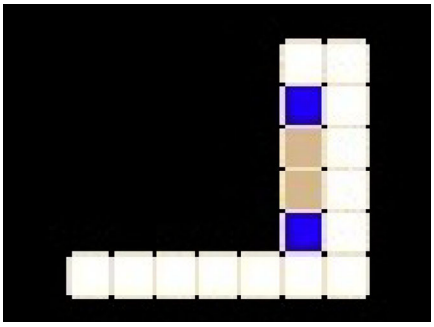
Fig. 11 The final configuration for $\{C_3, C_4\}$ and $\{C_1, C_4\}$, respectively



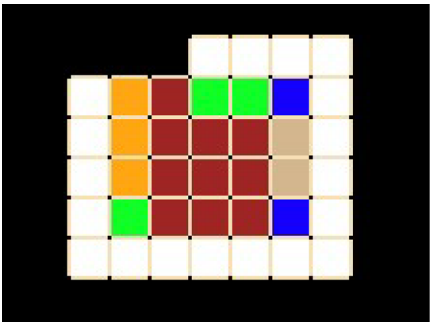
(a) The solution configuration of $\{C_3, C_4\}$



(b) The final configuration of $\{C_3, C_4\}$



(c) The solution configuration of $\{C_1, C_4\}$



(d) The final configuration of $\{C_1, C_4\}$

Fig. 12 The results of simulating the MinSetCover system by Xgrow

5.2.1 Tile code for MinExactSetCover problem

The proposed MinExactSetCover system is composed of the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem.

The seed configuration and the nondeterministic choice subsystem are similar to the corresponding subsystem of MinSetCover system. In the initial subsystem of MinSetCover system, it uses the tile $\langle s_i, null, \#, \# \rangle$ to represent different elements in set

S while the tile $\langle (S, s_i), null, \#, \# \rangle$ is used to represent different elements in the corresponding subsystem of the MinExactSetCover system where s_i is an element in S.

The tiles used in the initial subsystem here are shown in Fig. 13. You can notice that, the tiles $\langle (S, s_i), null, \#, \# \rangle$ are used to represent the elements in the set S where $s_i \subseteq \{1, 2, 3, 4, 5, 6\}$. The tiles $\langle |, |, C_j, \# \rangle$ are used to represent the different sets C_j where $C_j \subseteq \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{4, 5, 6\}\}$ and $1 \leq j \leq 4$. The left four types of tile which are shown in Fig. 13 are used as the boundary tiles.

The nondeterministic subsystems of the MinSetCover system and the MinExactSetCover system are the same. The tiles used in the MinExactSetCover system for our problem are shown in Fig. 14.

Lastly, in the proposed MinExactSetCover system, we have designed a new detection subsystem to identify the legal configurations. The detection subsystem involves the tiles shown in Fig. 14 which are used to detect whether or not C_j covers the element s_i for $s_i \subseteq \{1, 2, 3, 4, 5, 6\}$,$C_j \subseteq \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{4, 5, 6\}\}$, $1 \leq i \leq 6$ and $1 \leq j \leq 4$.

As shown in Fig. 15a, the tiles are used to detect the legal solutions. If C_j exists in the solution and it also covers s_i, then the green tile $\langle (OK, s_i), (S, s_i), C_j^1, C_j^1 \rangle$ is attached to the seed configuration. If C_j does not exist in the solution or it does not cover s_i, the crimson tile $\langle (S, s_i), (S, s_i), C_j^1, C_j^1 \rangle$ and $\langle (S, s_i), (S, s_i), C_j^0, C_j^0 \rangle$ will be attached to the seed configuration, respectively.

The yellow tiles shown in Fig. 15b are used to deliver the covering state of s_i. At last the white tiles and the *SUC* tile in Fig. 15c are used to detect and identify the legal configurations.

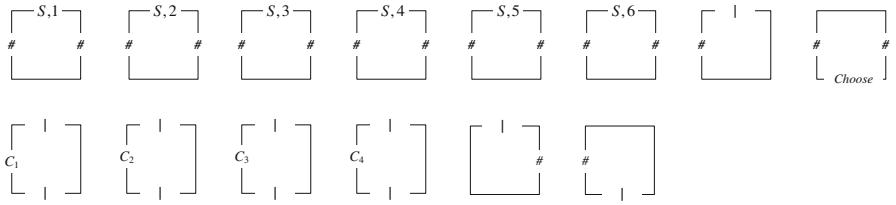


Fig. 13 The tiles used in the initial subsystem for MinExactSetCover problem

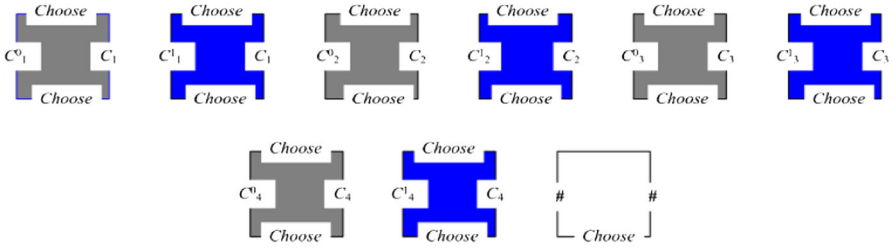
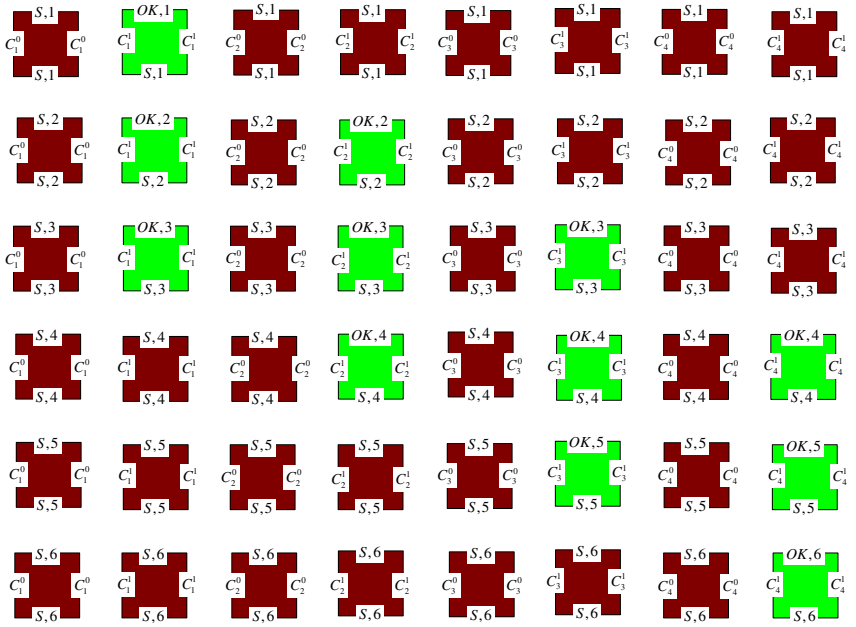
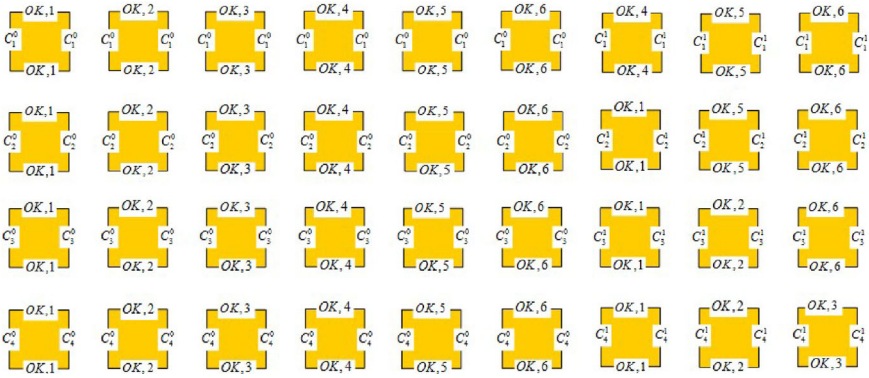


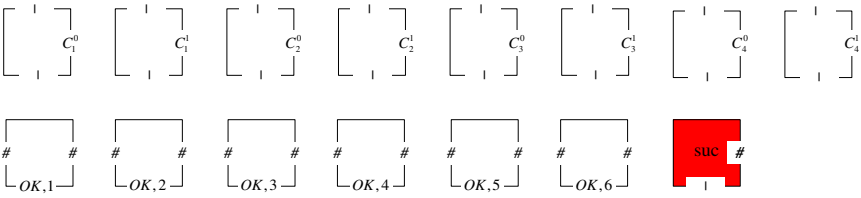
Fig. 14 The tiles used in the nondeterministic choice subsystem for MinExactSetCover problem



(a) The tiles used to detect whether or not C_j covers the element s_i for $s_i \subseteq \{1, 2, 3, 4, 5, 6\}$, $C_j \subseteq \{\{1, 2, 3\}, \{2, 3, 4\}, \{3, 4, 5\}, \{4, 5, 6\}\}$



(b) The tiles used to deliver the covering state of the element s_i for $s_i \subseteq \{1, 2, 3, 4, 5, 6\}$



(c) The gray tiles used as the boundary tiles and the red tile used to identify the legal configurations

Fig. 15 The tiles used in the detection subsystem

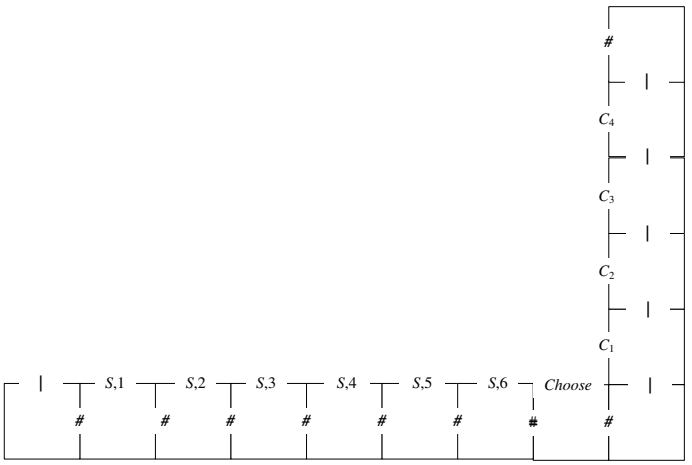


Fig. 16 The seed configuration for our problem which are generated by the initial subsystem

5.2.2 MinExtractSetCover solution procedure

The MinExtractCover system here is composed of the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. In the following, we will describe the solution procedure.

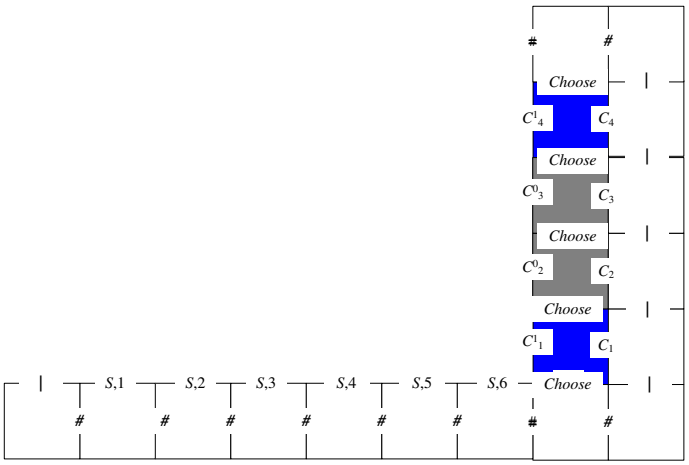
Firstly, by running the initial subsystem, we will get the seed configuration for our problem (see Fig. 16). The seed configuration is a L-Configuration and the 0th row mainly encode the elements s_i of the set S for $1 \leq i \leq 6$, the 0th column is mainly encode the sets C_j for $1 \leq j \leq 4$.

Secondly, after obtaining the seed configurations, we will execute the nondeterministic choice subsystem and choose C_j by attaching the tiles shown in Fig. 14 to the seed configurations nondeterminately. Thus, we will get all the solution configurations which represent the solution space for our problem. Figure 17 shows two of the solution configurations which represent the solution $\{C_1, C_4\}$ and $\{C_1, C_2, C_4\}$, respectively.

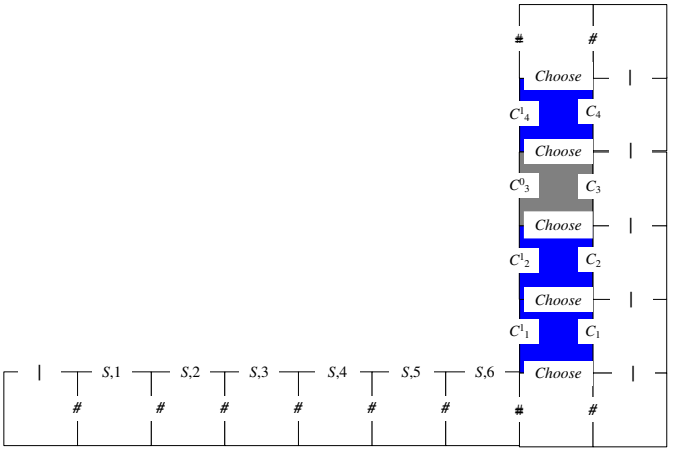
The final configurations are generated by the detection subsystem and the configurations with red *SUC* tile $\langle null, |, null, = \rangle$ are the legal configurations and represent the satisfiable solutions for the MinExtractSetCover problem. As shown in Fig. 18, the final configurations represent the solution $\{C_1, C_4\}$ and $\{C_1, C_2, C_4\}$, respectively. The final configuration for $\{C_1, C_4\}$ contains a red tile $\langle null, |, null, = \rangle$ with value *SUC*, so $\{C_1, C_4\}$ is the satisfiable solution for our problem.

5.2.3 MinExtractSetCover system simulation

In this section, we will also use the Xgrow to test the effectiveness of the MinExtractSetCover system.



(a) The solution configuration for $\{C_1, C_4\}$

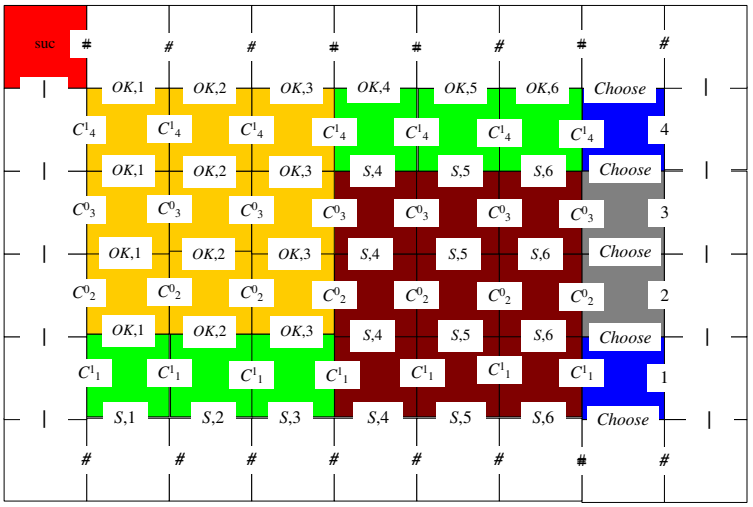


(b) The solution configuration for $\{C_1, C_2, C_4\}$

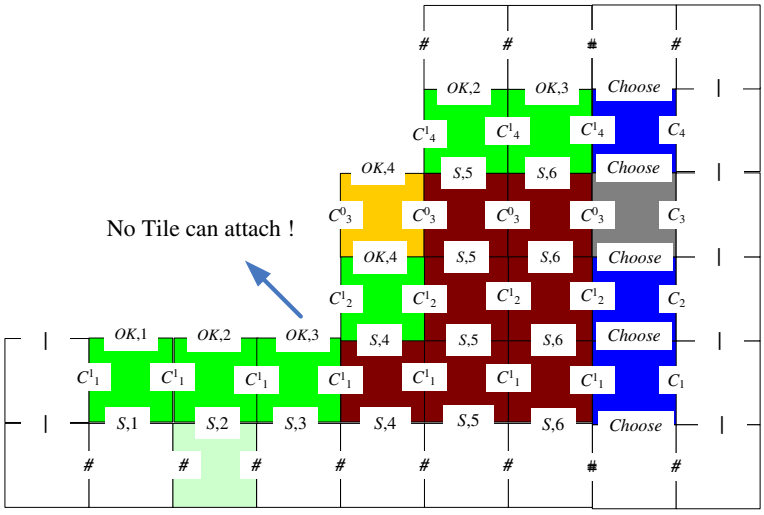
Fig. 17 The solution configurations for $\{C_1, C_4\}$ and $\{C_1, C_2, C_4\}$ which are generated by the nondeterministic subsystem

Figure 19a shows the solution configuration that represents the collection $\{C_1, C_4\}$. After attaching the corresponding DNA tiles we get final configuration shown in Fig. 19b. Because of the red *SUC* tile containing in the final configuration of collection $\{C_1, C_4\}$, the collection $\{C_1, C_4\}$ is a satisfiable answer of our problem.

Figure 19c shows the solution configuration of the collection $\{C_1, C_2, C_4\}$ and Fig. 19d shows its final configuration which does not contain the red *SUC* tile. Hence, the collection $\{C_1, C_2, C_4\}$ is not an answer of our problem.



(a) The final configuration for $\{C_1, C_4\}$

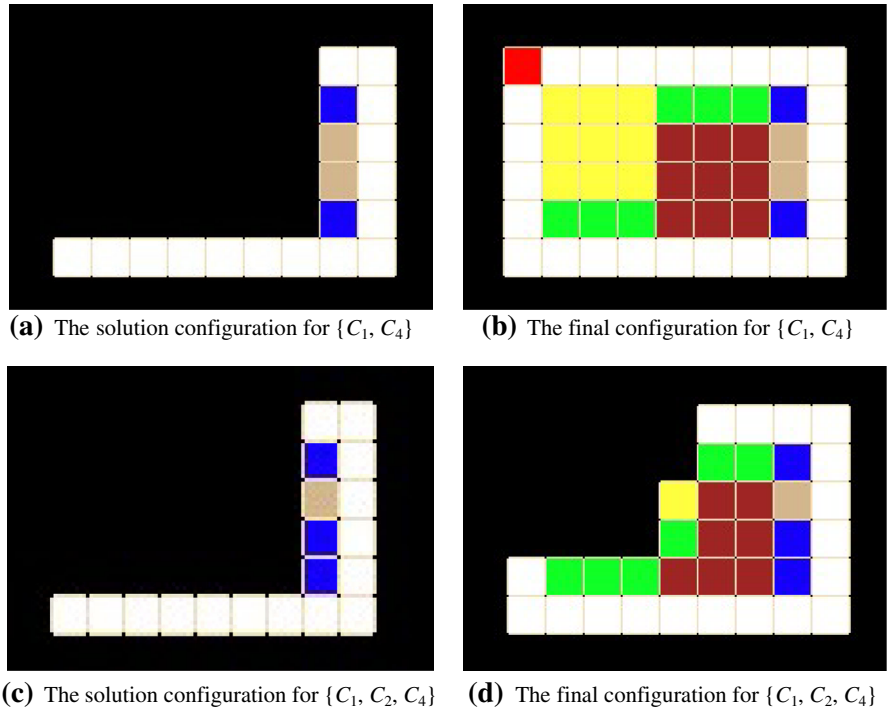


(b) The final configuration for $\{C_1, C_2, C_4\}$

Fig. 18 The final configuration for $\{C_1, C_4\}$ and $\{C_1, C_2, C_4\}$, respectively

6 Conclusion and future work

In this paper, we designed a MinSetCover system using the tile assembly model where the MinSetCover system is composed of three subsystems which are the initial subsystem, the nondeterministic choice subsystem, and the detection subsystem. Also, we have proved that the MinSetCover system uses $O(nk)$ distinct tiles with $O(k)$ assembly time, $O(nk)$ space complexity, and the least successive rate 0.5^k where n is equal



(a) The solution configuration for $\{C_1, C_4\}$ **(b)** The final configuration for $\{C_1, C_4\}$

(c) The solution configuration for $\{C_1, C_2, C_4\}$ **(d)** The final configuration for $\{C_1, C_2, C_4\}$

Fig. 19 The results of simulating the MinExtractSetCover system by Xgrow

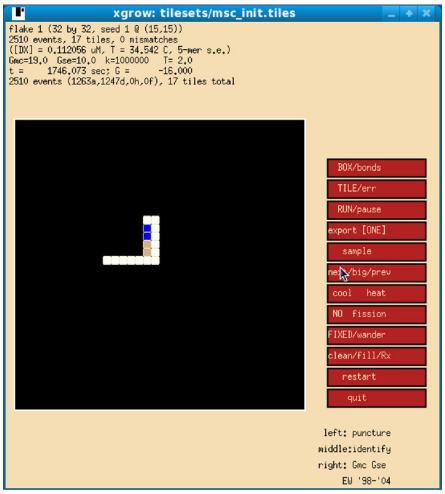
to $|S|$ and k is equal to $|C|$. Then, by improving the proposed MinSetCover system, we also designed a MinExtractSetCover system which needs $O(nk)$ distinct tiles, $O(k)$ assembly time, $O(nk)$ space complexity, and the least successful rate 0.5^k, where n is equal to $|S|$ and k is equal to $|C|$. Finally, we verified our contribution through a set of simulation experiments using Xgrow kit.

According to the well-known research achievements about the tile assembly model [11–18], we can find out that the number of distinct tiles needed in the tile assembly systems is one of the most important performance indexes. It is worthwhile to reduce the distinct tiles requested in the tile assembly's system [18]. In [18], Brun have solved the satisfiability problem using the tile assembly model with a constant-size tile set. Thus, based on the work of [21] and the development of DNA computing [25–27], our future work is to improve the presented systems and design more effective systems with a constant-size tile set.

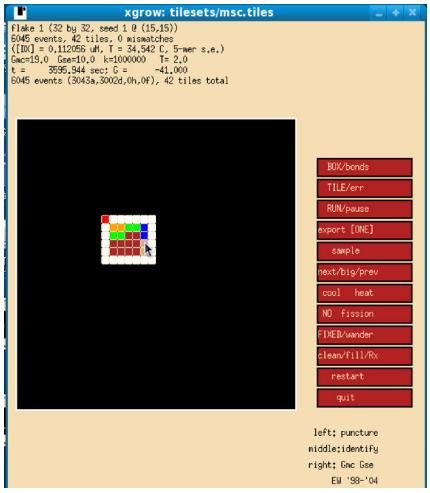
Acknowledgments This research is supported by the key Project of National Natural Science Foundation of China under grant 61133005, the Project of National Natural Science Foundation of China under grant 61173013 and 61202109, the Project of the Office of Education in Zhejiang Province under grant Y201226110.

Appendix

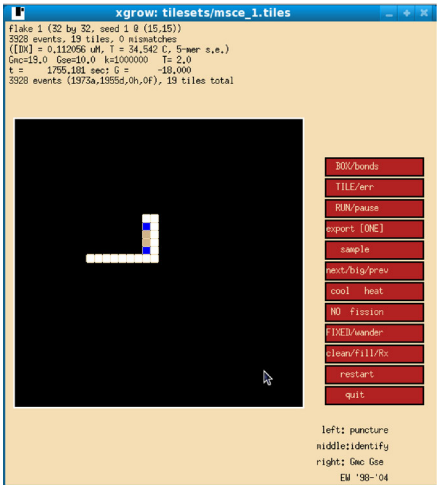
The simulation results by running the Xgrow simulator for solving our problem above (Figs. 20, 21).



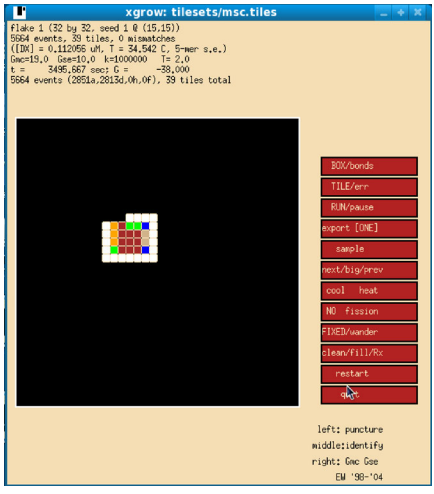
(a) The solution configuration for $\{C_3, C_4\}$



(b) The final configuration for $\{C_3, C_4\}$

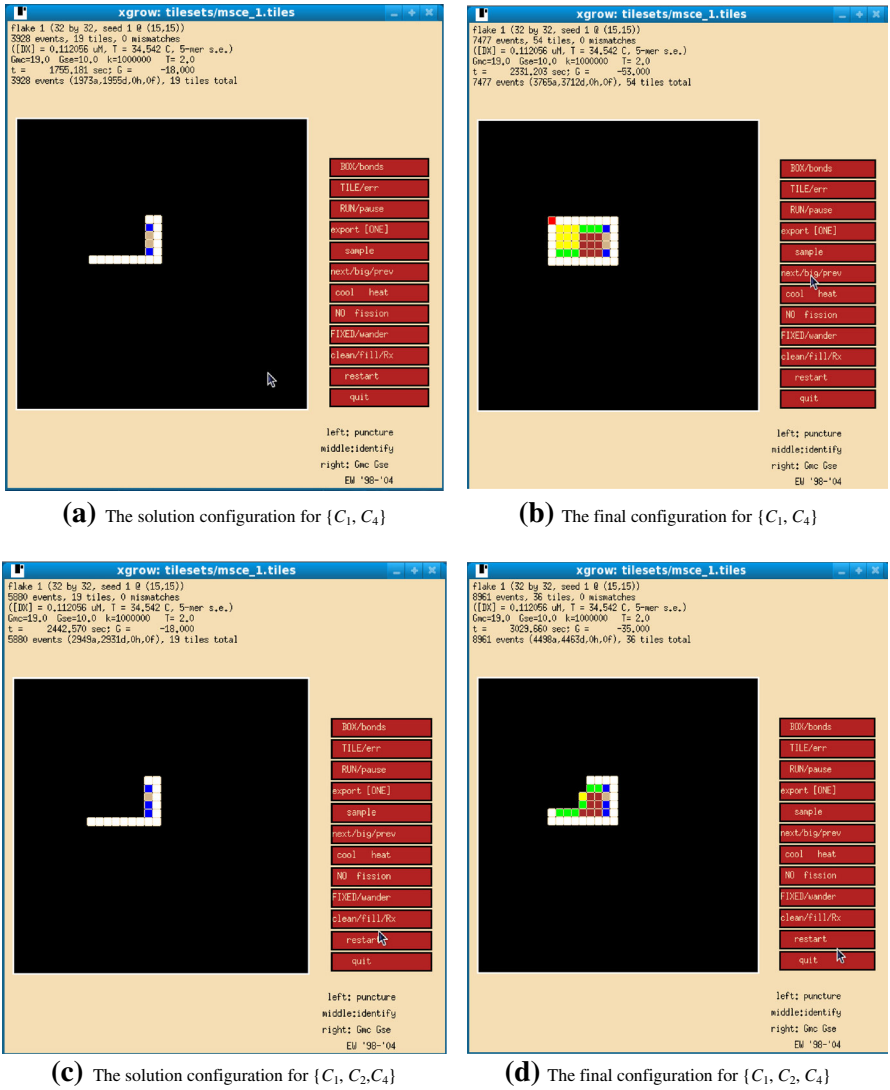


(c) The solution configuration for $\{C_1, C_4\}$



(d) The final configuration for $\{C_1, C_4\}$

Fig. 20 The results of simulating the MinSetCover system by Xgrow



(a) The solution configuration for $\{C_1, C_4\}$ **(b)** The final configuration for $\{C_1, C_4\}$

(c) The solution configuration for $\{C_1, C_2, C_4\}$ **(d)** The final configuration for $\{C_1, C_2, C_4\}$

Fig. 21 The results of simulating the MinExtractSetCover system by Xgrow

References

1. Chang WL, Lin KW, Chen JC et al (2012) Molecular solutions of the RSA public-key cryptosystem on a DNA-based computer. J Supercomput 61(3):642–672
2. Chang WL (2012) Fast parallel DNA-based algorithm for molecular computation: quadratic congruence and factoring integers. IEEE Trans Nanobiosci 11(1):62–69
3. Chang WL, Huang SC, Lin WC et al (2011) Fast parallel DNA-based algorithms for molecular computation: discrete logarithm. J Supercomput 56:129–163
4. Jin X, Xiaoli Q, Yan Y et al (2011) An unenumerative DNA computing model for vertex coloring problem. IEEE Trans Nanobiosci 10(2):94–98

5. Li K, Zou S, Xu J (2008) Fast parallel molecular algorithms for DNA-based computation: solving the elliptic curve discrete logarithm problem over GF(2n). J Biomed Biotechnol 1:1–10
6. Zhou Xu, Li Kenli et al (2011) A novel approach for the classical Ramsey number problem on DNA-based super-computing. Match Commun Math Comput Chem 66(1):347–370
7. Daniel M, Alfonso R-P, Petr S (2011) On the scalability of biocomputing algorithms: the case of the maximum clique problem. Theor Comput Sci 412(51):7075–7086
8. Bakar RAB, Watada J, Pedrycz W (2008) DNA approach to solve clustering problem based on a mutual order. Biosystems 91(1):1–12
9. Ikno K, Junzo W, Wutikd P et al (2012) Pattern clustering with statistical methods using a DNA-based algorithm. IEEE Trans Nanobiosci 11(2):100–110
10. Ikno K, Junzo W (2009) Decision making with an interpretive structural modeling method using a DNA-based algorithm. IEEE Trans Nanobiosci 8(2):181–191
11. Winfree E (1998) Design and self-assembly of two-dimensional DNA crystals. Nature 394(8):1223–1226
12. Adleman LM, Cheng Q, Goel A et al (2001) Running time and program size for self-assembled squares. In: Proceedings of the thirty-third annual ACM symposium on theory of computing, STOC 2001, Hersonissos, Greece. ACM, pp 740–748
13. Summers S (2012) Reducing tile complexity for the self-assembly of scaled shapes through temperature programming. Algorithmica 63:1–20
14. Woods D, Chen HL, Goodfriend S et al (2013) Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In: Innovations in theoretical computer science, ITCS 2013, Berkeley, California, January 10–12
15. Brun Y (2006) Arithmetic computation in the tile assembly model: addition and multiplication. Theor Comput Sci 378:17–31
16. Brun Y (2008) Solving NP-complete problems in the tile assembly model. Theor Comput Sci 395(1):31–46
17. Brun Y (2008) Nondeterministic polynomial time factoring in the tile assembly model. Theor Comput Sci 395(1):3–23
18. Brun Y (2008) Solving satisfiability in the tile assembly model with a constant-size tile set. J Algorithms 63(4):151–166
19. Zhang XC, Niu Y et al (2009) Application of DNA self-assembly on graph coloring problem. J Comput Theor Nanosci 6(5):1067–1074
20. Cheng Z, Huang YF et al (2009) Algorithm of solving the subset-product problem based on DNA tile self-assembly. J Comput Theor Nanosci 6(5):1161–1169
21. Guangzhao C, Cuiling L et al (2009) Application of DNA self-assembly on maximum clique problem. Adv Intell Soft Comput 116:359–368
22. Jie L, Lei Y, Kenli L (2008) An $O(1.414^n)$ volume molecular solutions for the exact cover problem on DNA-based supercomputing. J Inf Comput Sci 5(1):153–162
23. Chang WL, Guo M (2003) Solving the set-cover problem and the problem of exact cover by 3-sets in the Adleman–Lipton model. BioSystems 72(2):263–275
24. Fan Wu, Li Kenli, Sallam Ahmed et al (2013) A molecular solution for minimum vertex cover problem in tile assembly model. J Supercomput 66:148–169
25. Winfree E The xgrow simulator. http://dna.caltech.edu/Xgrow/
26. Tsai CC, Huang HC, Lin SC (2011) FPGA-based parallel DNA algorithm for optimal configurations of an omnidirectional mobile service robot performing fire extinguishment. IEEE Trans Ind Electron 58(3):1016–1026
27. Lulu Q, Erik W, Bruck J (2011) Neural network computation with DNA strand displacement cascades. Nature 475:368–372