Journal of Computational and Theoretical Nanoscience
Vol. 7, 246–253, 2010

3D DNA Self-Assembly Model for Graph Vertex Coloring

Minqi Lin[1,*], Jin Xu[2], Dafang Zhang[1], Zhihua Chen[3], Xuncai Zhang[3], Zhen Cheng[3], Yufang Huang[3], and Yanbiao Li[1]

[1] *Laboratory of Dependable Systems and Network, College of Software, Hunan University, Changsha, 410082, China*
[2] *School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China*
[3] *Department of Control Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074, China*

DNA self-assembly technology has brought novel inspirations to the development of DNA computing. Diversified computational models based on DNA self-assembly have been used to solve various NP problems. In this paper, a 3D DNA self-assembly model is presented to solve the Graph Vertex Coloring problem. With the capacity of DNA molecules in massive parallel computation, the model can simulate a non-deterministic algorithm and solve the problem in linear time: $\Theta(n)$. The number of distinct tiles used in the model is $\Theta(k^2)$, where k is the size of the color set. For the vertex 3-coloring problem, the model requires only 22 types of distinct tiles. Our work makes a significant attempt for exploring the computational power of 3D DNA self-assembly.

Keywords: Vertex Coloring, 3D DNA Self-Assembly, Non-Deterministic Algorithm, DNA Computing.

1. INTRODUCTION

The graph vertex coloring problem can be simply described as follow: color for all the vertices of a graph, such that no adjacent vertices are assigned with the same color. It is a difficult combinatorial optimization problem involved in many fields, such as timetabling, scheduling, circuit laying and storage. As a famous NP-complete problem, it has not been solved effectively by traditional algorithms.

Since the seminal work of Adleman on the Hamilton path problem (HPP),[1] the field of DNA based computing has experienced a flowering growth. Diverse DNA computing methods have been demonstrated to solve the graph vertex coloring problem or its equivalences. In 1995 Adleman discussed the 3-colorability problem in the restricted model.[2] Then, Bach improved the computing model developed by Adleman and gave a more general model of DNA algorithms,[3] which decreased the time complexity of the 3-coloring problem. Liu and Jin discussed the graph coloring problem with DNA algorithms based on surfaces.[4,5] Wang described a DNA sticker algorithm for vertex-coloring problem by converting the problem into vertex-independent sets problem and vertex-partition problem.[6] Sun constructed a new-style pre-hairpin probe, with which the solution space of the problem is reduced.[7] Yang gave a solution of graph vertex coloring problem implying the multi-separation techniques,[8] with less experiment steps and less DNA codes.

However, most of the approaches described above implements the computation by performing a series of biochemical reactions on a set of DNA molecules, which require human interventions in each step. Thus, one difficulty with such methods for DNA computing is the number of laboratory procedures, each time consuming and error-prone, growing with the size of the problem.

In this paper, we propose a 3D DNA self-assembly model for graph vertex coloring problem. The basic idea is to exploit the massive parallelism possible DNA operations in order to emulate a non-deterministic system that solves the problem in linear time. In fact, the innumerable self-assembled DNA molecules exhaust all the possible coloring schemes, and eliminate the non-solutions gradually along with the growth of self-assembly, but not the experimental operations, which result in a reduction of the manual intervention and error rate. As the analysis shows that our model can be achieved by a constant number of tile types in a linear time and polynomial space.

2. DNA BASED SELF-ASSEMBLY

Self-assembly is a ubiquitous and autonomous process in which small objects combine together to form lager complexes without any human interventions. Because of its

*Author to whom correspondence should be addressed.

nature biochemical properties, DNA molecule has been accepted to be the emphasis of self-assembly research. DNA based self-assembly technology has been widely applied in many fields such as nanofabrication, molecular circuit and supermolecular material. Another significant application is to use the DNA self-assembly structures to perform parallel universal computations. Actually, the DNA computing method by which Adleman solved the HPP implied the idea of self-assembly. In his method, the DNA molecules representing all possible paths through the target graph were assembled by DNA hybridization in a single step. However, the problem is that only simple computations can be performed with linear self-assembly. Winfree then discussed several string tile models for DNA computing by self-assembly.[9] Mao firstly implemented the XOR calculation experimentally by assembling the one-dimensional DNA three-triple molecule.[10]

In order to solve more complicated problems, more effective computational system based on DNA self-assembly is developed. As an extension for the tilling theory of Wang tiles,[11] the Tile Assembly Model (TAM) was firstly proposed by Winfree in 1996,[12] which used a DNA structure with four sticky ends to be a "Wang tile" in the square. Later, Lagoudakis give an example of 2D self-assembly for the Satisfiability problem.[13] Also with the TAM, Brun implemented the adding and multiplying operation,[14] based on which, he developed a model for the SubSetSum problem.[15] In his model, the distinct tile types became a constant "49". Based on a similar model, he gave a solution for the Satisfiability problem, still with a constant number of distinct tile types.[16] Comparing with Lagoudakis' model, Brun's model seems to be more advanced.

Also, the computational power of 3D self-assembly model was not ignored. Jonoska firstly implied the 3D DNA structure into the process of computing.[17] In his model, a computational problem was converted into a 3D graph problem. That is, the solution of the problem is determined from the 3D graphs, which were formed by vertex building blocks consisting of branched junction molecules. He presented the construction procedures for the 3-SAT and 3-vertex-colorability. As an upgrade of TAM, Reif introduced a type of 3D polyhedron tile for parallel prefix computation in his local parallel self-assembly model,[18] the tile design is given in Figure 1(a).

Along with these theoretical self-assembly model for computation, diversified DNA self-assembly structures were produced in laboratories, owing to the well-established biochemistry used to manipulate DNA molecule. In 1998, Winfree designed 2D DNA crystals using the DNA double-crossover molecule.[19] In 2008, Yin demonstrated the method of programming the self-assembly pathway,[20] by which the self-assembly process can be designed according to the target DNA structures. In the same year, Crocker proposed a method to assemble

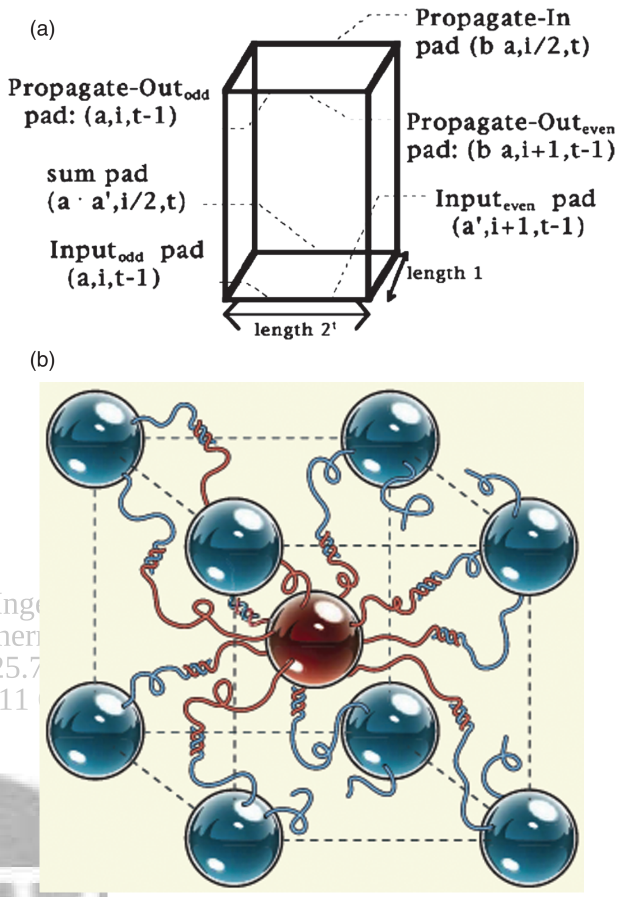
(a)
Propagate-In pad (b a,i/2,t)
Propagate-Out$_{odd}$ pad: (a,i,t-1)
Propagate-Out$_{even}$ pad: (b a,i+1,t-1)
sum pad (a · a',i/2,t)
Input$_{even}$ pad (a',i+1,t-1)
Input$_{odd}$ pad (a,i,t-1)
length 1
length 2^t
(b)

Fig. 1. Examples of 3D self-assembly. (a) A type of 3D polyhedron tile. (b) A cube structure self-assembled by gold spheres with long DNA strands covalently grafted onto their surfaces.

a cube structure through the transient pairings of complementary DNA strands,[21] the cube he designed is shown in Figure 1(b). He's research demonstrated the symmetric supramolecular polyhedra fabricated by hierarchical self-assembly of DNA.[22] As we can see that the experimental research on self-assembly has been extended from 2D to 3D.

3. 3D DNA SELF-ASSEMBLY MODEL FOR GRAPH VERTEX COLORING

2D self-assembly structures have shown comparative computational capability theoretically and experimentally. However, as Winfree demonstrated that the self-assembly of double crossover molecules into 2D sheets or 3D solids is theoretically capable of universal computation, and 3D self-assembly augments computational power.[23] As the development of biochemical technique and further understanding of self-assembly phenomenon, diverse 3D self-assembly structures were fabricated in laboratories. The feasibility of computational model based on 3D self-assembly is becoming evident. In this section, we focus on the graph vertex coloring problem and do some exploration on algorithmic modeling by 3D DNA self-assembly structure.

Here, we introduce a non-deterministic algorithm for graph vertex coloring, which ensured that, with certain non-deterministic choice, the appropriate scheme of vertex coloring could be obtained. By separating the adjacent information and coloring information, we get the input of the algorithm. And then, two referenced table: the adjacent table and the coloring table, were imported to indicate the inputs and assist designing the components of self-assembly. At last, the non-deterministic algorithm is simulated by the self-assembly model based on the components we design.

3.1. Problem Description

For better illustration, we give a formal description of the graph vertex coloring problem:

Given a simple undirected graph G with vertices set $V(G)$ and edges set $E(G)$, to find the coloring function f: $V(G) \to \{1, 2, \ldots, k\}$, which denotes a mapping from vertex to the color set, and satisfies that: $\forall E_{uv} \in E(G)$, $u, v \in V(G)$, $f(u) \neq f(v)$. It is called the graph vertex k-coloring problem.

Here, for the definiteness of our researching objective, we consider the case in which $k = 3$ and the color set is $\{\mathrm{r}, \mathrm{b}, \mathrm{y}\}$, where r denotes red, b denotes blue and y denotes yellow, namely, the graph vertex 3-coloring problem. Nevertheless, it is obvious that our model works in cases when k is other values.

3.2. Modeling Preparation

There are some preparative works to do before we implement the self-assembly model. At first, we introduce a non-deterministic algorithm for vertex coloring. Then, two referenced tables are employed to represent the inputs of the non-deterministic algorithm.

3.2.1. Non-Deterministic Algorithm

To solve this problem, we introduce the non-deterministic algorithm for vertex coloring, described as follow:

```
Non-Deterministic Algorithm($G, f$):
(1) For each $V_i \in V(G)$ {
(2)     Color for $V_i$ : $f(i) \to \{\mathrm{r, b, y}\}$
(3)     Check all $E_{uv} \in E(G)$ if exist $(i = u \vee i = v)$
      and $f(u) = f(v)$.
(4)         Break and return failure
(5) }
(6) If all $V_i \in V(G)$ are colored
(7) Return success and output $f(G)$
(8) Else return failure
```

A non-deterministic algorithm implies that there are some non-deterministic choices at some steps of the algorithm (like an oracle would tell what the right choice is). Although most of these choices directly output a failure, while the times of choosing turn into infinite, there must be a certain choice that will result in a return of success. The high memory density and massive parallelism of DNA computing endow the ability of making simultaneous choices in a unit time and space, which guarantees a result meeting the restriction of the algorithm. Notice that step 2 is the non-deterministic step.

As we can see that the input of the non-deterministic algorithm includes two parts: G, denotes the graph information, f denotes the coloring mapping. Thus, we employ two referenced tables to represent these inputs, as well as, to assist self-assembly modeling. The detailed illustrations are given in the following two sections.

3.2.2. Adjacent Table: T_a

According to the graph theory,[24] a graph with n vertices corresponds to an $n \times n$ matrix, indicating all the edges of the graph, called adjacent matrix. The following is an adjacent matrix of an n-vertices graph.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{bmatrix}$$

Since that we only consider the simple undirected graph, it is taken for granted that, the main diagonal and upper triangle part of the matrix remain blank, the variable a_{ij} in the under triangle part take a value among $\{0, 1\}$.

For example, a graph G_0, as is presented in Figure 2, it can be described in form of adjacent matrices as follow:

$$\begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ 1 & 0 & 1 & & \\ 1 & 0 & 0 & 1 & \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

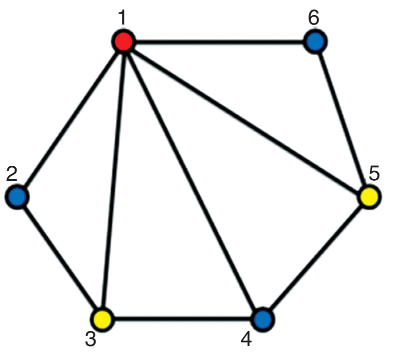

Fig. 2. A graph of 3-coloring.

Table I. T_a of G_0.

	1	2	3	4	5	6
1	**1**					
2	1	**2**				
3	1	1	**3**			
4	1	0	1	**4**		
5	1	0	0	1	**5**	
6	1	0	0	0	1	**6**

Table I presented an equivalence form to the above matrix. The first row and column of the table, as well as the cells in the main diagonal, represent the vertices of the graph. Cells in the under triangle part denotes the adjacent status of vertices. If two vertices are connected by an edge, the corresponding cell in the table will take a value "1"; otherwise, there will be a value "0". For instance of G_0, there is an edge between V_2 and V_3, thus the cell (3, 2) takes the value "1".

3.2.3. Coloring Table: T_c

As an adjacent table can completely indicates the vertex set and edge set of a simple undirected graph, a coloring table is used for implying a coloring scheme, which meets the restriction that the two vertices of each edge being assigned with different colors. For the same example of G_0 In Figure 1, its 3-coloring scheme "rbybyb" is presented in Table II.

Similarly, the first row and column of T_c, as well as the cells in the main diagonal, represent the vertex set of the graph. Cells in the lower triangle part denote the color values of the vertices in their rows and columns. Take cell (3, 2) for example, while the V_2 is assigned with b (blue), and V_3 is assigned with y (yellow), therefore, the cell (3, 2) takes a value of "by" (no sequence between b and y). Apparently, a coloring table uniquely implies a vertex coloring scheme.

Consequently, the vertex coloring problem is converted to be as follow: given T_a of a graph, to find a T_c whose sells take a value of different colors on condition that the value of T_a's cells in the same position is 1. With well designed self-assembly model, the problem can be solved with facility.

Table II. T_c of G_0.

1	2	3	4	5	6	
1	**1**					
2	rb	**2**				
3	ry	by	**3**			
4	rb	bb	yb	**4**		
5	ry	by	yy	by	**5**	
6	rb	bb	yb	bb	yb	**6**

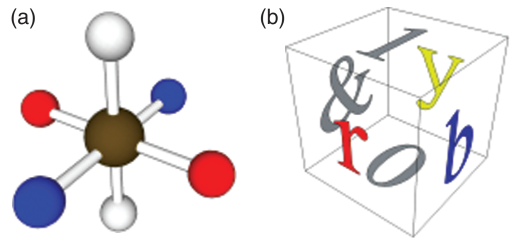

Fig. 3. (a) Molecular model of 3D DNA tile. (b) Hexahedron model abstracted from the molecular model.

3.3. Design of 3D DNA Tiles

Complexes consisted of multiply DNA strands with unpaired ends of DNA strands sticking out can attach themselves with other complexes having the Watson-Crick complementary sticky end. With the logical equivalence between DNA sticky ends and Wang tile edges, these complexes are used to simulate the Wang tiles, called DNA tiles. These tiles can stick with one-another to assemble into complex superstructures, through this process they can compute in a way similar to Wang tiles.

Unlike the square tile in the 2D self-assembly model, in order to process the self-assembly in 3D space, the DNA tiles should be DNA structures with six sticky ends. Figure 3(a) shows the molecular model of such a structure. The tile can be abstracted as a hexahedron with labels on its surfaces. Each label indicates a particular kind of sticky end. Two sticky ends that can match and ligate correspond to identical labels. Each tile can have any from 1 to 6 labels. Non-labeled surfaces indicate non-sticky ends. Figure 3(b) shows a hexahedron tile with labels on all its surfaces.

Formally, the DNA tile can be denoted by a 6-tuple $(\sigma_X, \sigma_{-X}, \sigma_Y, \sigma_{-Y}, \sigma_Z, \sigma_{-Z}) \in \Sigma^6$, in which the six variables indicate the labels on the six surfaces of the hexahedron. Each surface correspond to one of the directions of x, y, z axis and their inverses directions in the Cartesian coordinate system. For instance, the hexahedron in Figure 3(b) is denoted by (b, &, y, r, 1, 0).

It's unnecessary to list all the possible tiles in our design. Instead, we give the value scope of the variables in the 6-tuple. If a variable is not referred, it means that the surface is not labeled. With the two referenced tables: T_a and T_c we get the following tiles.

3.3.1. Adjacent Tiles

($\sigma_Z = \{0, 1\}$), include two subtypes, taking the information whether the vertices are adjacent or nonadjacent.

Figure 4(a) shows the two subtypes of the adjacent tiles, the hexahedron with "0" in its top surface represents the cells with value 0 in T_a, while the hexahedron with "1" in its top surface represents the cells with value 1 in T_a. These two tiles can form into different structures indicating different adjacent information from different graphs.

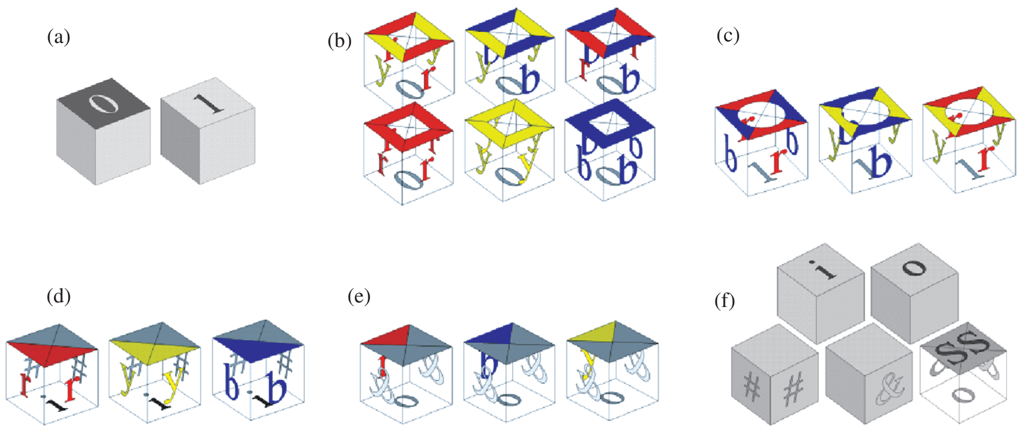

Fig. 4. Design of hexahedron tiles: (a) Adjacent tiles. (b) Coloring pass tiles. (c) Coloring check tiles. (d) Coloring input tiles. (e) Coloring output tiles. (f) Coloring boundary tiles.

3.3.2. Coloring Pass Tiles

($\sigma_X = \sigma_{-X} = \{\mathrm{r, b, y}\}$, $\sigma_Y = \sigma_{-Y} = \{\mathrm{r, b, y}\}$, $\sigma_{-Z} = \{0\}$), include six subtypes, representing the cells in T_c. Cells in the same position in T_a take the value 0.

Figure 4(b) shows the six subtypes of the coloring pass tiles. On their bottom surfaces are all label "0", the labels on the side surfaces represent the color values in T_c. Their top surfaces are painted with colors corresponding to the labels, each with a square blank being left in the center.

3.3.3. Coloring Check Tiles

($\sigma_X = \sigma_{-X} = \{\mathrm{r, b, y}\}$, $\sigma_Y = \sigma_{-Y} = \{\mathrm{r, b, y}\} \wedge \sigma_X \neq \sigma_Y$, $\sigma_{-Z} = \{1\}$), include three subtypes, representing the cells in the T_c. Cell in the same positions in T_a take the value 1.

Figure 4(c) shows the three subtypes of the coloring check tiles. On their bottom surfaces are all label "1", the labels on other surfaces represent the color values in T_c, their top surfaces are painted with colors corresponding to the labels, and each with a round blank being left in the center.

And yet, the tiles are not enough for assembling, some more tiles are required to input the color value to the assembly and output the final coloring scheme.

3.3.4. Coloring Input Tiles

($\sigma_X = \sigma_Y = \{\mathrm{r, b, y}\}$, $\sigma_{-X} = \sigma_{-Y} = \{\#\}$, $\sigma_{-Z} = \{\mathrm{i}\}$), include three subtypes. They can determine the color of each vertex and input them into the assembly.

Figure 4(d) shows the three subtypes of the coloring input tiles. On their bottom surfaces are all label "i", the labels on the back surfaces are "#", and labels on the front surfaces are color values.

3.3.5. Coloring Output Tiles

($\sigma_X = \{\mathrm{r, b, y}\}$, $\sigma_Y = \sigma_{-Y} = \{\&\}$, $\sigma_{-Z} = \{\mathrm{o}\}$), include three subtypes. They can output the final coloring scheme.

Figure 4(e) shows the three subtypes of the coloring output tiles. On their bottom surfaces are all label "o", the labels on the right-back and left-front surfaces are "&", and labels on the left-back surfaces are color values.

3.3.6. Coloring Boundary Tiles

($\sigma_Z = \{\mathrm{i, o}\}$), ($\sigma_X = \{\&\}$), ($\sigma_X = \sigma_Y = \{\#\}$), ($\sigma_X = \sigma_Y = \{\&\}$, $\sigma_{-Z} = \{\mathrm{o}\}$, $\sigma_Z = \{\mathrm{SS}\}$), include five subtypes. They can control the growing direction of the self-assembly.

Figure 4(f) shows the five subtypes of the coloring boundary tiles. The above two tiles are labeled with "i" or "o" on their top surfaces. Below them, the left first tile is labeled with "#" on its two front surfaces, the left second tile is labeled with "&" on its right-front surface, and the left third tile is labeled with "o" on its bottom surface, "&" on its two back surfaces and "SS" on its top surface.

3.4. The Process of Self-Assembly

From the design of the DNA tiles, we can see that there are tiles with one labeled surfaces, two labeled surfaces, four labeled surfaces and five labeled surfaces. In order to make the assembling more organized, we import a mechanism to instruct the assembling.

A strength function g: $\sum \times \sum \to R$ is considered such that mismatched surfaces have no interaction strength and matching surfaces have positive strengths.

$$g(\sigma, \sigma') = \begin{cases} 1 & \text{if } \sigma = \sigma' \\ 0 & \text{otherwise} \end{cases}$$

A tile may be added to an assembly if the summed strength of its interactions with its neighbors exceeds a threshold, called temperature. The variable comes from the anneal temperature of DNA hybridization. In our model, the temperature is a constant 3, thus, only tiles that match up to three surfaces can be assembled in position.

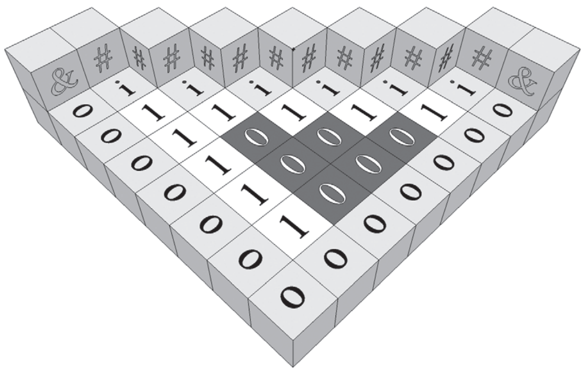

Fig. 5. Seed configuration of 3D self-assembly.

3.4.1. Seed Configuration

The process of self-assembling start from the seed configuration, which is a DNA structure constructed artificially beforehand, carrying the information of the problem to be solved.

In our problem, the seed configuration is made up of the adjacent tiles and the coloring boundary tiles, carrying the adjacent information of the graph to be colored.

Figure 5 shows the seed configuration of the 3D self-assembly model. Through appropriate placement of every tile, we get the adjacent information of G_0. In addition, some coloring boundary tiles are posited around in order to let the growth of self-assembly run smoothly.

Once the seed configuration is constructed, the self-assembly is ready to grow. Before that there is a fact worth to be noticed. While assembling, a tile can be rotated arbitrarily which might result in a situation that a surface in x-axis is attached to a surface in y-axis with identical labels on their surfaces.

3.4.2. Successful Self-Assembly

The growth of self-assembly follows the flow of the non-deterministic algorithm. Figure 6(a) shows the first step of assembling, each vertex of G_0 is assigned with a color non-deterministically, and this process can be achieved in a unit time during assembling.

The details are, the coloring input tile ($\sigma_X = \sigma_Y = \{r\}$, $\sigma_{-X} = \sigma_{-Y} = \{\#\}$, $\sigma_{-Z} = \{i\}$) is attached to the first (left to right) "i" tile which represents V_1 in G_0, with "i," "#" and "#" matched on deferent surfaces. Analogously, the coloring input tile ($\sigma_X = \sigma_Y = \{b\} \wedge \sigma_{-X} = \sigma_{-Y} = \{\#\} \wedge \sigma_{-Z} = \{i\}$) are attached to the second, fourth and sixth coloring entrance tile which represent in V_2, V_4 and V_6 in G_0, the coloring input tile ($\sigma_X = \sigma_Y = \{y\}$, $\sigma_{-X} = \sigma_{-Y} = \{\#\}$, $\sigma_{-Z} = \{i\}$) are attached to the third and fifth coloring entrance tile which represent V_3, V_5 in G_0.

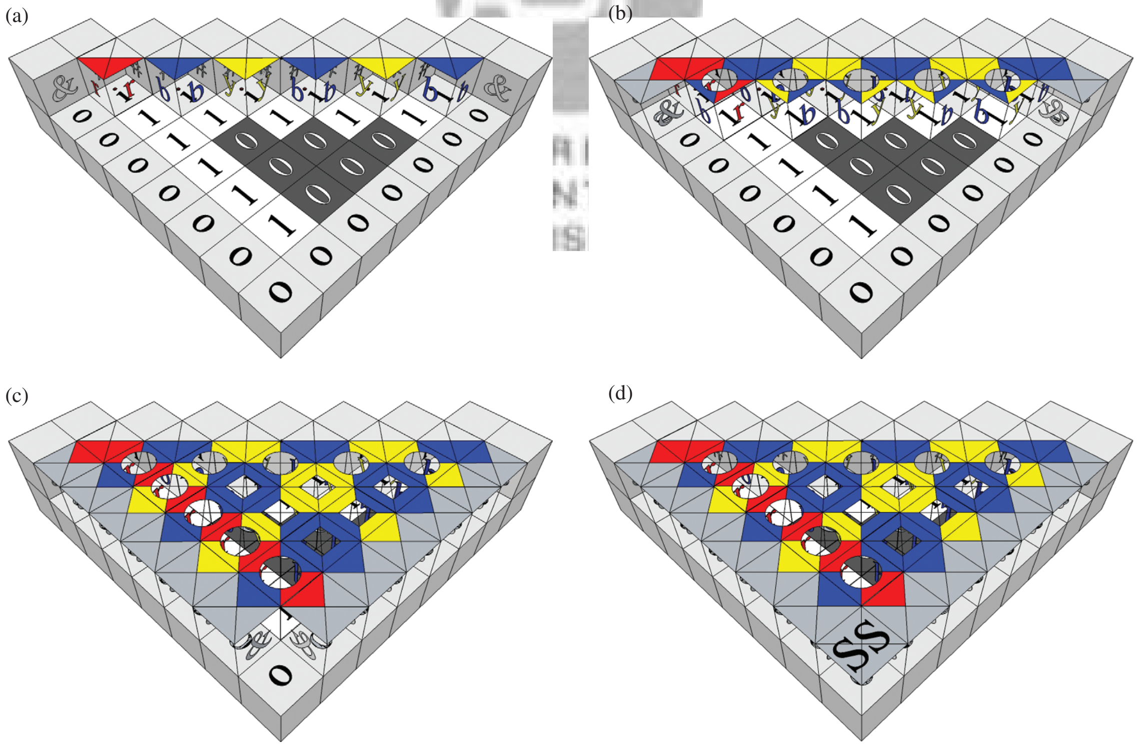

Fig. 6. A successful self-assembly. (a) The first step of assembling, the coloring input tiles are assembled randomly. (b) The second step of assembling, begin to check the coloring scheme. (c) The step before the end of the assembling, when coloring check is finished. (d) The end of the assembling, a successful self-assembly with proper coloring scheme is presented.

And then, the color assignment must be checked whether there exist any vertices that being assigned with the same colors to its adjoining vertices.

Take the position of vertices pair 12 for example, labels on the contiguous surfaces are "1", "r" and "r", thus only the coloring check tile ($\sigma_X = \sigma_{-X} = \{\text{r}\}$, $\sigma_Y = \sigma_{-Y} = \{\text{b}\}$, $\sigma_X \neq \sigma_Y \{\wedge\} \sigma_{-Z} = \{1\}$) can be attached here. Figure 6(b) shows the first step of coloring check, it's noticed that the vertices pairs: {12, 23, 34, 45, 56} are checked simultaneously. In the mean time, two coloring output tiles are assembled in position.

Subsequently, the vertices pairs sets, {13, 24, 35, 46}, {14, 25, 36}, {15, 26}, {16} are processed at different steps each is achieved in a unit time. Figure 6(c) shows the assembly which has finished coloring check in five steps.

Lastly, the "SS" tile ($\sigma_X = \sigma_Y = \{\&\}$, $\sigma_{-Z} = \{\text{o}\}$, $\sigma_Z = \{\text{SS}\}$) is attached to the surfaces labeled with "&", "&" and "o", which means the accomplishment of the entire self-assembling. The terminal state is shown in Figure 6(d), from which an appropriate coloring scheme "rbybyb" is obtained.

3.4.3. *Unsuccessful Self-Assembly*

Not all the possible color assignments satisfies the restriction of vertex coloring, actually, a majority of them result in a failure finally. The improper coloring scheme can be detected at a certain step of the assembling.

Figure 7 presents an instance of unsuccessful self-assembly. As we can see that at the position with a red "×" marked on the top, three are labels "r", "r" and "1" close-by. Seems that a tile ($\sigma_X = \sigma_{-X} = \sigma_Y = \sigma_{-Y} = \{\text{r}\}$, $\sigma_{-Z} = \{1\}$) may fits here. Unfortunately, there are no such tile in our design, hence, the self-assembly is not going to grow any more.

3.5. Complexity Analysis

The complexity of our 3D DNA self-assembly model is considered in terms of computation time, computation space and the number of distinct tiles.[25,26]

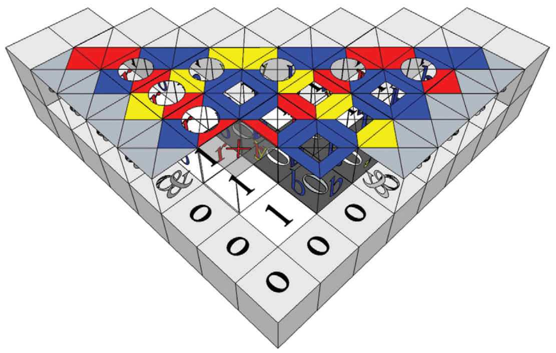

Fig. 7. An unsuccessful self-assembly.

3.5.1. *Computation Time*

According to the growing trend of the assembly, the computation time can be compute as follows, $T = \Theta(n+2) = \Theta(n)$.

3.5.2. *Computation Space*

The space taken for each assembly is the volume of the assembly, which is easy to compute as follows: $S = \Theta((n+1)^2 + 4(n+1) + 2) = \Theta(n^2)$.

3.5.3. *Number of Distinct Tiles*

The types of distinct tiles include all the subtypes in our design. Totally, there are 2 subtypes of adjacent tile, subtypes of coloring pass tile, $C_2^1 \times C_k^1$ subtypes of coloring check tile, C_k^2 subtypes of coloring input tile C_k^1 subtypes of coloring output tile, 5 subtypes of coloring boundary tile. In sum, it is, $N = 2 + C_2^1 \times C_k^1 + C_k^2 + C_k^1 + C_k^1 + 5 = \Theta(k^2)$. In case $k = 3$, $N = 2 + 2 \times 3 + 3 + 3 + 3 + 5 = 22$.

4. CONCLUSIONS

Benefit from the high memory density and massive parallelism, DNA computing is provided with unexampled dominance in solving difficult problems, especially for NP-complete problems.

The 3D DNA self-assembly model in our study can be viewed as a kind of extension from 2D TAM. Concerning the graph vertex 3-coloring problem, we separate the adjacent information and coloring information of vertices in a graph into different tilling plane, so as to reduce the complexity of DNA tiles. As a result, we get the solution with a constant number of tile types: 22, in linear time $\Theta(n)$ and polynomial space $\Theta(n^2)$.

While the 3D self-assembly model augments the computational power, it bring tough challenges to biochemical technique. So far, we cannot valid our model experimentally, that is what we are going to work on in the future.

Acknowledgments: We thank Zuoxin Gan for helpful discussion and acknowledge the support by the National Natural Science Foundation of China (Grant Nos. 60803113).

References

1. L. M. Adleman, *Science* 266, 1021 **(1994)**.
2. L. M. Adleman, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society 122 **(1996)**, p. 1.
3. E. Bach, E. Glaser, A. Condon, and C. Tanguay, *Proceedings, Eleventh Annual IEEE Conference on Computational Complexity*, Philadelphia **(1996)**, p. 290.
4. Y. Liu, J. Xu, L. Pan, and S. Wang, *J. Chem. Inf. Comput. Sci.* 42, 524 **(2002)**.

5. X. Jin and G. Liu, *Computer and Communications* 21, 6 (**2003**).
6. S. Wang, W. Liu, and J. Xu, *Systems Engineering and Electronics* 27, 568 (**2005**).
7. C. Sun, X. Zhu, W. Liu, and J. Xu, *Computer Engineering and Applications* 42, 58 (**2006**).
8. Y. Yang, A. M. Wang, and J. Ma, *Fourth International Conference on Natural Computation*, Washington (**2008**), p. 547.
9. E. Winfree, T. Eng, and G. Rozenberg, *Lecture Notes in Computer Science* 2054, 63 (**2001**).
10. C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, *Nature* 407, 493 (**2000**).
11. H. Wang, *Tech. J* 40, 1 (**1961**).
12. E. Winfree, *DNA Based Computers* 199, 199 (**1996**).
13. M. G. Lagoudakis and T. H. LaBean, *Proceedings, the 5th DIMACS Workshop on DNA Based Computers*, Cambridge 54 (**2000**), p. 141.
14. Y. Brun, *Proceedings of the 4th Foundations of Nanoscience: Self-Assembled Architectures and Devices, FNANO07*, Snowbird, UT, USA (**2007**).
15. Y. Brun, *Theoretical Computer Science* 395, 31 (**2008**).
16. Y. Brun, *Journal of Algorithms* 63, 151 (**2008**).
17. N. Jonoska, S. A. Karl, and M. Saito, *BioSystems* 52, 143 (**1999**).
18. J. H. Reif, *Proceedings, DNA Based Computers III: DIMACS Workshop, Providence* (**1999**), p. 217.
19. E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman, *Nature* 394, 539 (**1998**).
20. P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce, *Nature* 451, 318 (**2008**).
21. J. C. Crocker, *Nature* 451, 528 (**2008**).
22. Y. He, T. Ye, M. Su, C. Zhang, A. E. Ribbe, W. Jiang, and C. Mao, *Nature* 452, 198 (**2008**).
23. E. Winfree, X. Yang, and N. C. Seeman, *Proceedings, DNA Based computers II, Providence* (**1999**), p. 191.
24. J. A. Bondy and U. S. R. Murty, Graph Theory with Applications, Macmillan, London (**1976**).
25. P. W. K. Rothemund and E. Winfree, *Proceedings, Thirty-Second Annual ACM Symposium on Theory of Computing*, Portland (**2000**), p. 459.
26. L. Adleman, Q. Cheng, A. Goel, and M. D. Huang, *Proceedings, Thirty-Second Annual ACM Symposium on Theory of Computing*, Crete (**2001**), p. 740.

Received: 24 March 2009. Accepted: 26 May 2009.