Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 407816, 7 pages
http://dx.doi.org/10.1155/2013/407816

Research Article

Solving Vertex Cover Problem Using DNA Tile Assembly Model

Zhihua Chen, Tao Song, Yufang Huang, and Xiaolong Shi

Key Laboratory of Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China

Correspondence should be addressed to Tao Song; songtao0608@hotmail.com

Received 13 October 2013; Accepted 15 November 2013

Academic Editor: Sabri Arik

DNA tile assembly models are a class of mathematically distributed and parallel biocomputing models in DNA tiles. In previous works, tile assembly models have been proved be Turing-universal; that is, the system can do what Turing machine can do. In this paper, we use tile systems to solve computational hard problem. Mathematically, we construct three tile subsystems, which can be combined together to solve vertex cover problem. As a result, each of the proposed tile subsystems consists of $\Theta(1)$ types of tiles, and the assembly process is executed in a parallel way (like DNA's biological function in cells); thus the systems can generate the solution of the problem in linear time with respect to the size of the graph.

1. Introduction

In the history of human designing computing devices, nature has provided brilliant ideas and frameworks to design intelligent computing models. Recently, due to the highly distributed and parallel features, bioinspired computing, including DNA computing [1–4], membrane computing [5–9], and neural-like computing [10–12], have received a lot of research attention. One of the famous results of biocomputing is Adleman's DNA computing model, which uses DNA molecule to solve a Hamiltonian path problem in tubes [1]. It demonstrated the feasibility of solving combinatorial problems by molecular computing, which used the vast parallelism of DNA molecule to do the combinatorial search among a large number of possible solutions. Till now, DNA computing has been heavily developed and used in different research fields, such as breaking cryptosystems [2, 13], hiding messages [4, 13], and computing functions [14]. However, most of the proposed DNA computing models are not autonomous, which require human intervention in each step, and the scale of the problems solved by DNA computing is restricted by the large numbers of laboratory procedures, error-prone reactions, and the time consumed. In the present work, we focus on a kind of DNA computing models, named DNA tile assembly model.

DNA tile assembly model can automatically organize the components into defined aggregates with stable and robust structures [15]. Following the research line, it was theoretically proved that the tile assembly model of DNA computing is Turing-universal [2], while the experimental demonstration of computation using DNA tile was initialled in [16]. For general introduction of DNA tile assembly models, one can refer to the respective chapter of [2]. Recently, DNA assembly systems were also used in some other fields such as binary counters [17], computing Sierpinski triangles [18], and theoretically performing sum and product of two numbers [19].

In this work, we use DNA tile assembly model to solve the famous NP-hard problem—vertex cover problem. Specifically, three tile self-assembly subsystems are constructed, which can constitute subsets of vertices, calculate the types of vertices in each subset, and count the number of vertices covered by the vertex in a subset, respectively. We combine the three subsystems to construct an integrated tile system that can solve vertex cover problems in linear time. As a result, we obtain that each subsystem consists of $\Theta(1)$ types of tiles, and the system can generate the solution of the problem in linear time with respect to the size of the graph (due to the fact that each tile in the systems can do the assembly operation in a parallel way).

2. DNA Tile Self-Assembly Model

This section is started by recalling some useful notions and formal definition of DNA tile self-assembly model proposed in [19].

Let Σ be a finite alphabet of binding domains, with null $\in \Sigma$. A tile over Σ is a unit square with 4-tuple $\langle \sigma_N, \sigma_E, \sigma_S, \sigma_W \rangle \in \Sigma^4$ indicating, respectively, the glues on the north, east, south, and west sides of the tile. A *position* is an element of $\mathbb{Z}^2$. The set of directions $D = \{N, E, S, W\}$ is a set of four functions from position to position, such that, for all positions (x, y), $(x, y) \in \mathbb{Z}^2$, then $N(x, y) = (x, y + 1)$, $E(x, y) = (x + 1, y)$, $S(x, y) = (x, y - 1)$, and $W(x, y) = (x - 1, y)$. The positions (x, y) and (x', y') are neighbors if and only if $\exists d \in D$ such that $d(x, y) = (x', y')$. For a tile t, for $d \in D$, we refer to $bd_d(t)$ as the binding domain of tile t on d's side. According to this definition, tiles may not be rotated. A special tile, $empty = \langle null, null, null, null \rangle$, represents the absence of all other tiles. The binding domains determine the interaction between tiles.

A function $g : \Sigma \times \Sigma \to \mathbb{R}$ is a *strength function* denoting the strength of the binding domains. The function g can be calculated as

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

The value may be 0, 1, or 2 (resp., called *null, weak*, and *strong* bonds).

Let T be a set of tiles containing the empty tile. One configuration of T is a function $A: \mathbb{Z} \times \mathbb{Z} \to T$, and $(x, y) \in A$ if and only if $A(x, y) \neq empty$. One configuration is finite if and only if there is only a finite number of distinct positions $(x, y) \in A$. One tile system SYS is a triple $\langle T, g, \tau \rangle$, with T being a finite set of tiles containing empty, g is a strength function, and $\tau \in \mathbb{N}$ is the temperature. If A is a configuration, then a tile t can attach to A at position (x, y) and produce a new configuration A' if and only if

(1) $(x, y) \notin A$,

(2) $\sum_{d \in D} g(bd_d(t), bd_{d^{-1}}(A(d(x, y)))) \geq \tau$,

(3) for all $(u, v) \in Z^2$, $(u, v) \neq (x, y) \Rightarrow A'(u, v) = A(u, v)$,

(4) $A'(x, y) = t$.

For a tile system SYS $= \langle T, g, \tau \rangle$, a set of tiles Γ, and a seed configuration $S : \mathbb{Z}^2 \to \Gamma$, if the above conditions are satisfied, one may attach tiles of T to S. Configurations produced by repeated attachments of tiles from T are said to be produced by SYS on S. If this process terminates, the configuration is called the final configuration F.

Let SYS $= \langle T, g, \tau \rangle$, and let S_0 be a seed configuration such that SYS produces a unique final configuration F on S_0, $W_0 \subseteq 2^{T \times \mathbb{Z}^2}$ the set of all tile position pairs $\langle t, (x, y) \rangle$ such that t can attach to S_0 at (x, y), and S_i the configuration produced by adding all the elements of W_{i-1} to S_{i-1} in one time step, where $i = 1, 2, \ldots$. Number n is the smallest natural number such that $S_n \equiv F$, where n indicates the assembly time of S on S_0 to produce F.

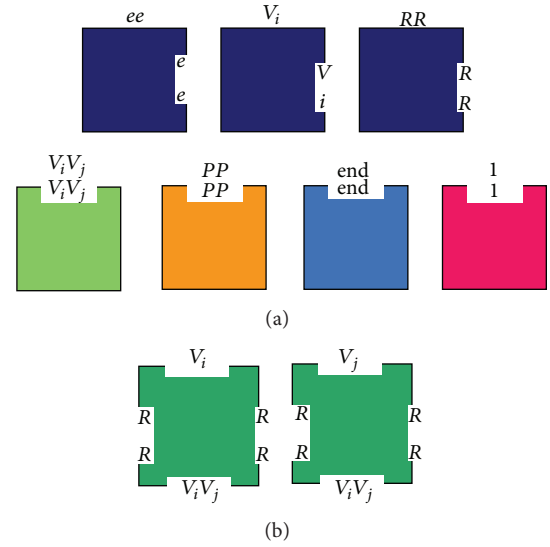

FIGURE 1: (a) Tiles in Γ_{seed} with the name written on the top; (b) the meaning of the tiles in T_v to encode edges V_iV_j and vertex V_i or V_j.

3. Solving Vertex Cover Problem by DNA Tile Assembly Model

The vertex cover tile system contains a set of tiles denoted by T_{vc}, which can be divided into three disjoint subsets of tiles T_v, T_t, and T_k. (Mathematically, we have $T_{vc} = T_v \cup T_t \cup T_k$.) The functions of the tiles in distinguished subsets are different. Specifically, the tiles in T_v are nondeterministically chosen to constitute a vertices subset covering all the edges; the tiles in T_t calculate the types in the constituted vertices subset; the tiles in T_k can be used to count the number of vertices that are not in the constituted vertices subset and determine whether the result meets the requirement of the problem.

The tile systems described above will use n_V types of tiles to encode the vertices, as well as n_E types of tiles to encode the edges in the graph. The set of the tiles used to encode the vertices and edges in the graph are denoted by

$$\begin{aligned} \Gamma_{seed} &= \{ee = \langle null, ee, null, null \rangle, \\ V_i &= \langle null, V_i, null, null \rangle, \\ RR &= \langle null, RR, null, null \rangle, \\ V_iV_j &= \langle V_iV_j, null, null, null \rangle, \\ PP &= \langle PP, null, null, null \rangle, \\ end &= \langle end, null, null, null \rangle, \\ 1 &= \langle 1, null, null, null \rangle \mid 1 \leq i, j \leq n_V \}. \end{aligned}$$

The seeds are graphically shown in Figure 1(a).

3.1. Constituting One Subset. For a given seed configuration consisting of a given graph, we start by describing a tile system that can use tiles from T_v to nondeterministically constitute a vertices subset covering all the edges and to encode the vertices and edges with these tiles. The strategy

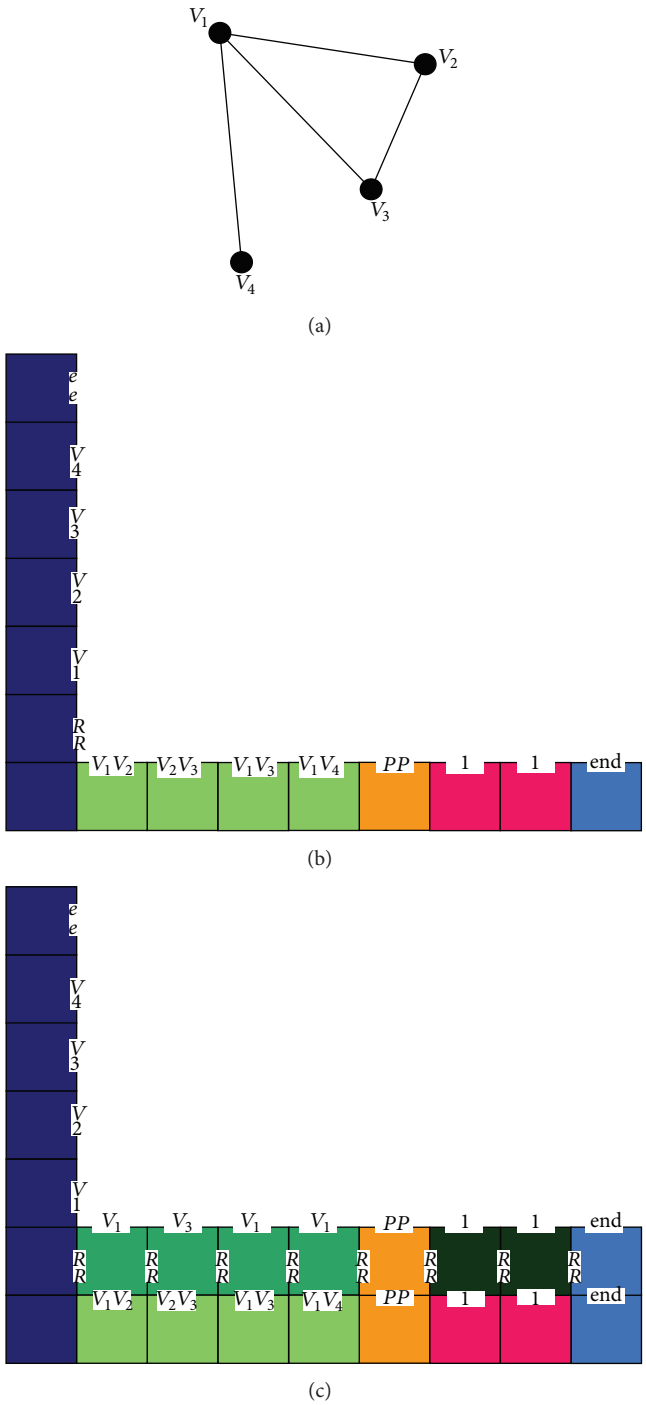

FIGURE 2: SYS_v produces a final configuration F on S_v such that F produces one subset of vertices. (a) An example graph; (b) the example seed configuration; (c) in this example, F constitutes one subset V_1, V_3, V_1, V_1.

used to encode an edge $(V_i, V_j) \in E$ by tiles can be found from Figure 1(b), where the symbols on the sides of the tiles indicate the edge or vertex.

In the following, we will give an example of a directed graph $\gamma = (V, E)$ with 4 vertices (denoted by V_1, V_2, V_3, and V_4; see Figure 2(a)) to explain the process of constituting one subset. The seed configuration S_v is designed by the following binding domains: $bd_S(S(0,0)) = V_1V_2$, $bd_S(S(1,0)) = V_2V_3$, $bd_S(S(2,0)) = V_1V_3$, $bd_S(S(3,0)) = V_1V_4$, $bd_S(S(4,0)) = PP$, $bd_S(S(5,0)) = 1$, $bd_S(S(6,0)) = 1$, and $bd_S(S(7,0)) = end$; see Figure 2(b). And the SYS_v produces a set of final configurations F on S_v such that $F(0,1) = V_1$, $F(1,1) = V_3$, $F(2,1) = V_1$, $F(3,0) = V_1$, $F(4,0) = PP$, $F(5,0) = 1$; $F(6,0) = 1$, and $F(7,0) = end$; see Figure 2(c).

Note that SYS_v can produce all the final configurations in F on S_v, where F corresponds to the vertices subset covering all the edges.

The tiles that are possible to attach to S_v can produce a final configuration F that corresponds to the subset covering all the edges.

(1) F and S_v should agree at the first line with the corresponding binding domains.

(2) Note that $bd_N(F(0,0)) = V_1V_2$, and the only tile with $bd_S(t) = V_1V_2$ might attach here. There are two types of tiles such that $bd_S(t) = V_1V_2$, so only these types of tiles can attach above. Furthermore, $bd_N(F(0,1)) = V_1$ or V_2, which indicates that the subset of vertices contains V_1 or V_2 (in this case, it is V_1).

(3) It is easy to check that $bd_N(F(1,0)) = V_2V_3$, so $F(1,1) = V_2$ or V_3 (in this case, the contained vertex is V_3).

(4) It is easy to check that $bd_N(F(2,0)) = V_1V_3$, so $F(2,1) = V_1$ or V_3 (in this case, the contained vertex is V_1).

(5) It is easy to check that $bd_N(F(3,0)) = V_1V_4$, so $F(3,1) = V_1$ or V_3 (in this case, the contained vertex is V_1).

(6) The tiles $bd_N(F(4,0)) = PP$ and $bd_S(t) = PP$ can attach here, where the tile $\{PP, RR\}$ is attached indicating the boundary of subset is constitutes.

(7) , (8) The same as (6), it can be easily obtained that $bd_S(F(5,0)) = 1, bd_S(F(6,0)) = 1$, so the tile $\{1, RR\}$ is attached here.

(9) We have $bd_S(F(7,0)) = end$, and the only tile with $bd_S(t) = end$ may attach here, which show the boundary of the final configuration F.

It is worth pointing out that no more tiles may attach to F and one of subsets which satisfied the requests of the problem is $\{V_1, V_3, V_1, V_1\}$. Therefore, for all choices of subsets covering all edges, there exists a final configuration F produced by SYS_v on S_v that constitutes a vertex subset. For different final configurations F' produced by SYS_v on S_v will constitute some random vertex subsets covering all edges.

Lemma 1 (subsets assembly time lemma). *The assembly time of the final configuration F that constitutes vertex subset is $\Theta(n_E)$.*

Proof. For each tile in F to attach, a tile in a specific location can have attached before (either to the north or to the east). Hence, there is no parallelism in the assembly, and the

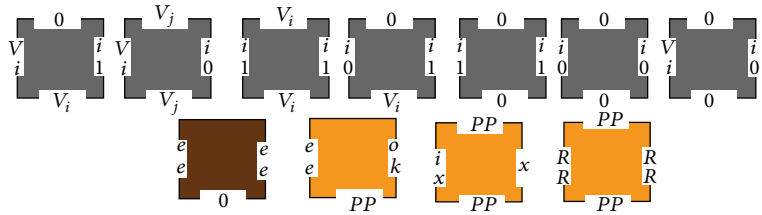

Figure 3: The meaning of tiles in T_t, and V_i indicates the vertex V_i in the graph; V_i can be used to check the presentation of vertex V_i in the subset: if so, the tile outputs $x = 1$, otherwise $x = 0$.

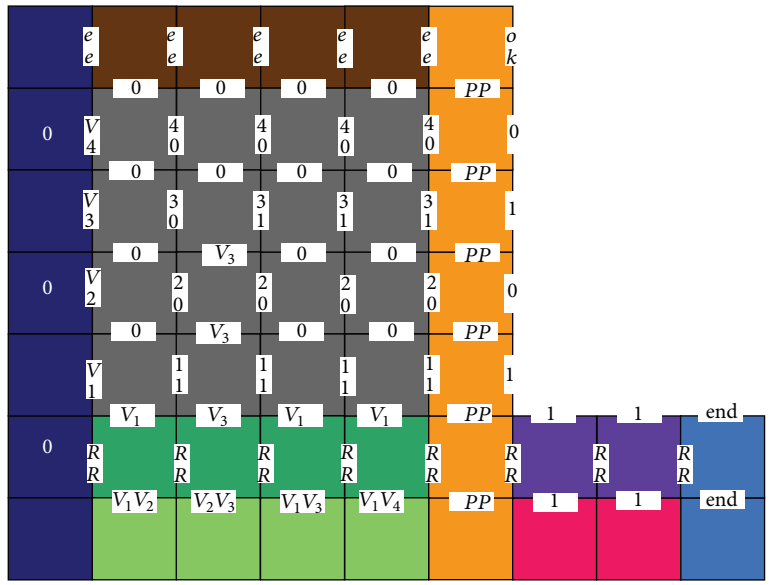

Figure 4: An example of the execution of SYS_t; the subset of vertices is $\{V_1, V_3, V_1, V_1\}$, and the checking result is $x = 1010$.

assembly time equals the total number of tiles that attached, which is $\Theta(n_E)$. □

3.2. Type Calculation. After the tile system have nondeterministically assembled the representation of one vertices subset, we add tiles to implement the vertex type calculation. The meaning of tiles in T_t is given in Figure 3.

Each tile $V_i \in T_t$ is used to check the presentation of vertex V_i whether it is in the constituted vertices subset: if so, then output is 1, otherwise output is 0. The tile system SYS_t can check the types of the vertices in the constituted subset. The example of the function of SYS_t is shown in Figure 4, where the constituted subset is $\{V_1, V_3, V_1, V_1\}$. There are four gray tiles which check the vertex V_i whether it is present in the subset, and the brown tiles complete the checking. The orange tiles are used to output the result: if the vertex is in the subset, the output is $x = 1$, otherwise $x = 0$.

Theorem 2. *Let* $\Sigma_t = \{V_i, i1, i0, 0, PP, RR, ee, ix, x \mid i \in \{1, 2, \ldots, n\}, x \in \{0, 1\}\}$, $\tau_v = 2$, *and* $g_{divisor} = (\sigma, \sigma) = 1$ *for* $\sigma \in \Sigma_t$, T_t *a set of tiles,* S_t *the final configuration produced by* SYS_v *on* S_v *that constitutes the subset of vertices,* F_v *unique final configuration (by which one can get the computing result),* S_t *the configuration such that*

$$\forall x, y \in T_t, \quad F_v(x, y) \neq empty \Longrightarrow S_t(x, y) = F_v(x, y), \tag{2}$$

and $SYS_t = \langle T_t, g_t, \tau_t \rangle$.

Proof. Let *Sub* be the constituted subset and V_i any vertex in the graph with $i \in \{1, 2, \ldots, n\}$. If $V_i \in Sub$, then $x_i = 1$, otherwise $x_i = 0$. The tile system SYS_t computes the final configuration F_v produced by SYS_v, and SYS_t will produce a unique final configuration F_t on S_t. The orange tiles (similar to those in Figure 4) can output the checking result and cooperate with other tiles. Therefore, the tile system SYS_t is able to check the type of the tiles and output the result during the self-assembly progress. □

Corollary 3 (checking assembly time). *The assembly time for* SYS_t *starting from one seed that encodes vertices and edges is* $\Theta(n_V + n_E)$.

Proof. The tiles in F_t will attach in a strict order; that is, only the corner position may be attached to one certain

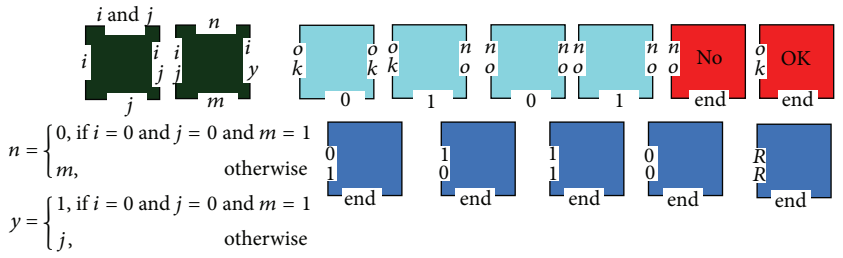

Figure 5: The meaning of tiles in T_k with $i, j, m, n, y \in \{0, 1\}$.

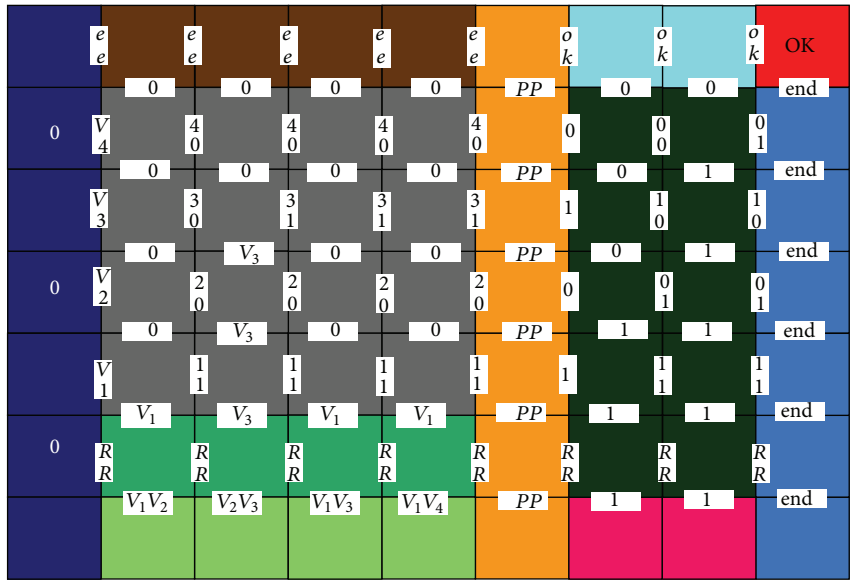

Figure 6: The final configuration F_k produced by the tile system SYS_k for the example of Figure 4.

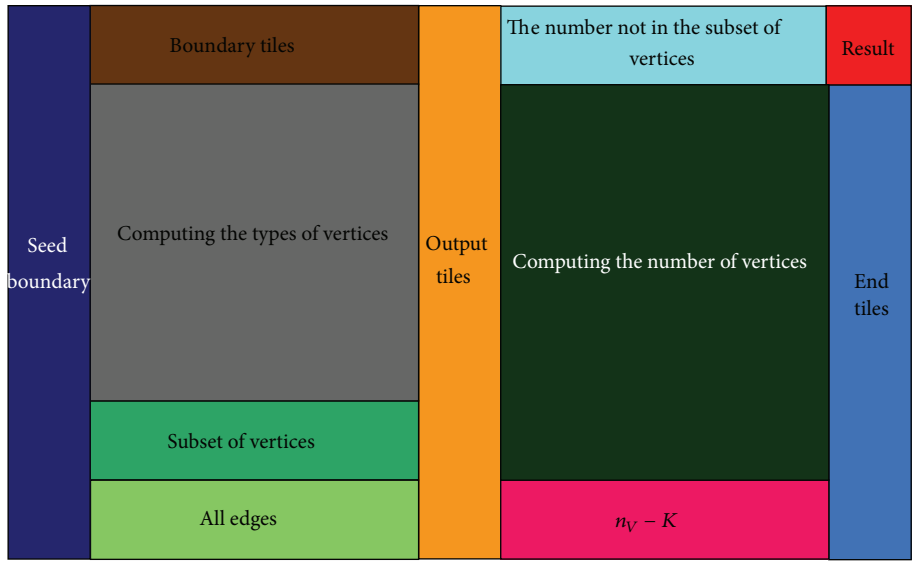

Figure 7: The construction of system SYS_C and the location of subsystems SYS_v, SYS_t, and SYS_k.

type of tiles. So the tiles in the diagonal positions perform the processes of computing in parallel. With the parallelism, the resulting configuration is the $n_V \times n_E$ rectangle, and the assembly time is $n_V + n_E$ steps. □

3.3. Checking the Number of Vertices in the Subset. With the two tile subsystems described above (one generates a random vertices subset, and the other one checks whether the vertex type is contained in the subset or not), we design one more tile subsystem to compute the number of vertices in the subset and to determine whether the number of vertices is less than K or not. The meaning of tiles in T_k is shown in Figure 5, where the dark-green tiles with variables i, j, m, n, and y are used to finish the vertex types counting in the subset; the light-blue tiles are used to check whether the number of vertex types in the subset is less than K or not; the red tiles with "NO" or "OK" serve as the mark to show the result of the problem, and the blue tiles will terminate the checking and mark the final configuration F produced.

Theorem 4. *Let* $\Sigma_k = \{0, 1, 00, 01, 10, 11, ok, no, end, RR\}$, $g_k(\sigma, \sigma) = 1$ *for all* $\sigma \in \Sigma_k$, $\tau_k = 1$, T_k *a set of tiles of the form shown in Figure 5,* S_k *the final configuration produced by the tile system* SYS_t *on* S_t *that outputs each vertex present in the subset or not,* F_k *the unique final configuration (by which one can get the computing result),* S_k *the configuration such that*

$$\forall x, y \in T_t, \quad F_v(x, y) \neq empty \Longrightarrow S_k(x, y) = F_v(x, y), \tag{3}$$

and $SYS_k = \langle T_k, g_k, \tau_k \rangle$. *It holds that* SYS_k *produces a unique final configuration* F_k, *and the number of vertex types in the subset is less than K if and only if* $F_k(n_E + (n_V - K + 1, 1 + n_V + 1)) = OK$.

Proof. If the "OK" tile is ever to attach, then it should attach in position $(n_E + (n_V - K) + 1, 1 + n_V + 1)$. Since the sum of the g values of the binding domains is exactly 2, it can only match its neighbors on all sides with nonnull binding domains to attach at temperature 2. Furthermore, the tile with a south binding domain "end" and a west binding domain "ok," matching the assembly's north binding domain and the east binding domain, is the "OK" tile, so it can attach only when the light-blue tiles "ok" have attached in position from $(n_E + 1, 1 + n_V + 1)$ to $(n_E + (n_V - K), 1 + n_V + 1)$. Once there is one position attaching "no," then the end position will attach "NO" tile.

Let us consider the unique final configuration F_k. The "ok" tile can attach at position $(n_E + 1, 1 + n_V + 1)$, and at west of the tile there is an east binding domain "ok." Therefore, the tiles at positions $(n_E + i, 1 + n_V + 1)$ with $i = 1, 2, \ldots (n_V - K)$ should have the output of east binding domain "ok." It indicates that at least $(n_V - K)$ vertices are not in the constituted subset. If the binding domain "no" occurs, it means that the number of vertex types in the constituted subset is more than K. In this case, the "No" tile will attach at position $(n_E + (n_V - K) + 1, 1 + n_V + 1)$. For the produced subset, an "OK" tile can attach at position $(n_E + (n_V - K) + 1, 1 + n_V + 1)$ to F_k if and only if the number of vertex types in the constituted subset is not more than K. □

Following the example given above, the final configuration F_k produced by the tile system SYS_k is shown in Figure 6.

Corollary 5 (calculating assembly time). *The assembly time of the final configuration F produced by* SYS_k *on* S_k *that draws the conclusion of the number of vertex types in the subset whether more than K or not is* $\Theta((n_V - K) + n_V)$.

3.4. Solving Vertex Cover Problem Using DNA Tile Assembly System. Till now, we have defined three tile subsystems that perform the necessary pieces of solving the vertex cover problem. We now combine them into one single system SYS_{VC} and argue that SYS_{VC} is a tile system solving the vertex cover problem.

If a tile system is the combination of three distinct tile subsystems, the behavior of the combined systems contains the behaviors of the subsystems. Although the tiles from different subsystems can interfere with each other, according to the design of the subsystems SYS_v, SYS_t, and SYS_k, they can work together without interfering. For the most part, each subsystem uses a disjoint set of binding domains, sharing binding domains only when tiles from the different subsystems are designed to interact. As a result, the tiles from each subsystem have a particular set of positions where they can attach. Specifically, tiles from T_v can only attach in row 1, tiles from T_t can only attach in the rectangle defined by the same color rectangle, and tiles from T_k can only attach in the same color rectangle too. Therefore, the tiles from different subsystems do not interfere with each other; see Figure 7. As explained above, we can easily obtain the following theorem.

Theorem 6 (vertex cover theorem). *Let* $\Sigma_{VC} = \Sigma_v \cup \Sigma_t \cup \Sigma_k$, $T_{VC} = T_v \cup T_t \cup T_k$; g_{VC} *agrees with* g_v, g_t, *and* g_k *on their respective domains, and* $\tau_{VC} = 2$. *It holds that the system* $SYS_{VC} = \langle T_C, g_C, \tau_C \rangle$ *is a tile system solving the vertex cover problem.*

Proof. Consider a directed graph $\gamma = (V, E)$ with n vertices denoted by $V_1, V_2, \ldots, V_n$, and a positive integer $K \leq card(V)$. The number of vertices is denoted by n_V, and the number of edges is n_E. We assume that $0 \leq a < n_E$, $S_{VC}(a, 0) = V_i V_j$, for all $(V_i, V_j) \in E$; $1 \leq i, j \leq n_V$, $S_{VC}(n_E, 0) = PP$, $1 \leq b < n_V - K$, $S_{VC}(n_E + b, 0) = 1$, and $S_{VC}(n_E + (n_V - K) + 1, 0) = end$.

For all edges in the graph, tiles from T_C will attach to S_C as follows: the tiles from T_v nondeterministically attach to constitute one vertex subset covering all edges in row 1, as described in Section 3.1. According to Theorem 2, the tiles from T_t attach, in the gray rectangle denoted in Figure 7, to check which vertex is obtained in the subset and output the results. Finally, tiles from T_k attach, in dark-green rectangle denoted in the Figure 7, such that, by Theorem 4, the "OK" tile attaches only when at least $(n_V - K)$ vertices are not present in the constituted subset; that is, the vertex number of the subset is no more than K. Therefore, SYS_{VC} can nondeterministically and identifiably solve the vertex cover problem. □

Corollary 7 (vertex cover assembly time). *The assembly time for* S_{VC} *to produce a final configuration F that draws the*

conclusion of the number of vertex types in the subset whether more than K or not is $\Theta(2n_V + n_E - K)$.

4. Conclusion

In this work, the vertex cover problem is considered to be solved by tile assembly system. Function package is a common technique usually used in programming by general digital computers. Herein, we have "packaged" the proposed three tile subsystems in tile self-assembly model: SYS_v, SYS_t, and SYS_k, and combine them together to form SYS_{VC} with such functions of generating subsets, computing functions, and checking results. For future work, it is still possible to decrease the number of tile types used in the tile systems.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61370105, 61179032, 61100055, 61373066, and 61272071), Natural Science Foundation of Hubei Province (Grant no. 2013CFB159), and the Fundamental Research Funds for the Central Universities (HUST2013TS124).

References

[1] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, no. 5187, pp. 1021–1024, 1994.

[2] E. Winfree, *Algorithmic self-assembly of DNA [Ph.D. thesis]*, California Institute of Technology, Pasadena, Calif, USA, 1998.

[3] J. H. Reif, S. Sahu, and P. Yin, "Compact error-resilient computational DNA tiling assemblies," in *Proceedings of the 10th International Workshop on DNA Computing*, pp. 293–307, June 2004.

[4] P. W. K. Rothemund, N. Papadakis, and E. Winfree, "Algorithmic self-assembly of DNA sierpinski triangles," *PLoS Biology*, vol. 2, no. 12, 2004.

[5] G. Păun, *Membrane Computing: an Introduction*, Springer, Berlin, Germany, 2002.

[6] G. Păun, G. Rozenberg, and A. Salomaa, Eds., *The Oxford Handbook of Membrane Computing*, Oxford University Press, 2010.

[7] G. Păun, "Computing with membranes," *Journal of Computer and System Sciences*, vol. 61, no. 1, pp. 108–143, 2000.

[8] T. Song, L. Pan, K. Jiang, B. Song, and W. Chen, "Normal forms for some classes of sequential spiking neural P systems," *IEEE Transactions on Nanobioscience*, vol. 12, no. 3, pp. 255–264, 2013.

[9] J. Wang, H. J. Hoogeboom, L. Pan, G. Păun, and M. J. Pérez-Jiménez, "Spiking neural P systems with weights," *Neural Computation*, vol. 22, no. 10, pp. 2615–2646, 2010.

[10] T. Song, L. Pan, J. Wang, I. Venkat, K. G. Subramanian, and R. Abdullah, "Normal forms of spiking neural P systems with anti-spikes," *IEEE Transactions on NanoBioscience*, vol. 11, no. 4, pp. 352–359, 2012.

[11] L. Pan and G. Păun, "Spiking neural P systems: an improved normal form," *Theoretical Computer Science*, vol. 411, no. 6, pp. 906–918, 2010.

[12] T. Song, L. Pan, and G. Păun, "Asynchronous spiking neural P systems with local synchronization," *Information Sciences*, vol. 219, pp. 197–207, 2012.

[13] L. Adleman, P. K. M. Rothemund, S. Roweis, and E. Winfree, "On applying molecular computation to the data encryption standard," in *Proceedings of the 2nd Annual Meeting on DNA Computers*, pp. 10–12, Princeton, NJ, USA, 1996.

[14] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman, "Solution of a 20-variable 3-SAT problem on a DNA computer," *Science*, vol. 296, no. 5567, pp. 499–502, 2002.

[15] J.-M. Lehn, "Supramolecular chemistry," *Science*, vol. 260, no. 5115, pp. 1762–1763, 1993.

[16] C. Mao, T. H. LaBean, J. H. Reif, and N. C. Seeman, "Logical computation using algorithmic self-assembly of DNA triple-crossover molecules," *Nature*, vol. 407, no. 6803, pp. 493–496, 2000.

[17] R. D. Barish, P. W. K. Rothemund, and E. Winfree, "Two computational primitives for algorithmic self-assembly: copying and counting," *Nano Letters*, vol. 5, no. 12, pp. 2586–2592, 2005.

[18] P. Rothemund and E. Winfree, "The program-size complexity of self-assembled squares," in *Proceedings of ACM Symposium on Theory of Computing (STOC '02)*, pp. 459–468, Quebec, Canad, May 2000.

[19] Y. Brun, "Arithmetic computation in the tile assembly model: addition and multiplication," *Theoretical Computer Science*, vol. 378, no. 1, pp. 17–31, 2007.