Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 4, pp. 490-505

Implementation of the Timetable Problem Using Self-assembly of DNA Tiles

Z. Cheng, Z. Chen, Y. Huang, X. Zhang, J. Xu

Zhen Cheng
College of Computer Science and Technology
Zhejiang University of Technology
288 Liuhe Road, Hangzhou, P.R. China
Email: chengzhen0716@163.com

Zhihua Chen, Yufang Huang, Xuncai Zhang
Department of Control Science and Engineering
Huazhong University of Science and Technology
1037 Luoyu Road, Wuhan, P.R.China

Jin Xu
School of Electronics Engineering and Computer Science
Peking University
No.5 Yiheyuan Road Haidian District, Beijing, P.R.China
E-mail: jxu@pku.edu.cn

Abstract: DNA self-assembly is a promising paradigm for nanotechnology. Recently, many researches demonstrate that computation by self-assembly of DNA tiles may be scalable. In this paper, we show how the tile self-assembly process can be used for implementing the timetable problem. First the timetable problem can be converted into the graph edge coloring problem with some constraints, then we give the tile self-assembly model by constructing three small systems including nondeterministic assigning system, copy system and detection system to perform the graph edge coloring problem, thus the algorithm is proposed which can be successfully solved the timetable problem with the computation time complexity of $\Theta(mn)$, parallely and at very low cost.
Keywords: timetable, self-assembly, graph edge coloring, DNA tiles

1 Introduction

Since Adleman [1] demonstrated the use of recombinant DNA techniques for solving a small combinational search problem, the field of DNA-based computing has experienced a flowering growth and leaves us with a rich legacy. DNA computing [2, 3] potentially provides a degree of parallelism and high density storage far beyond that of conventional silicon-based computers.

DNA tile self-assembly is an important method of molecular computation and it is also a crucial process by which objects autonomously assemble into complexes [4]. This phenomenon is common in nature and yet is poorly understood from mathematical and programming perspectives. It is believed that self-assembly technology will ultimately permit the precise fabrication of complex nanostructures. The DNA nanotechnology was initiated by Seeman [5] who proposed self-assembled nanostructures made of DNA molecules, and the key of this technology is immobilization of Holliday junction (crossover) to make well-defined DNA structures. Seeman [6] also utilized one of such structures called DX (double crossover) tile to realize a patterned lattice made of these tiles, which is used to construct not only simple pattern such as periodic stripes or barcodes, but also the complex algorithmic pattern. Winfree [7] proposed 2D self-assembly process and showed that computation by self-assembly is Turing-universal. Eng [8] demonstrated that self-assembly of linear, hairpin, and branched DNA molecules can generate

regular, bilinear, and context-free languages, respectively. Researchers have used DNA tile algorithmic self-assembly to create crystals with patterns of binary counters [9, 10] and Sierpinski triangles [11], which can be used to implement arbitrary circuit [12]. But those crystals are deterministic, generating nondeterministic crystals may hold the power to solve complex problems quickly.

Because of the complex and special structure of DNA tiles, tile self-assembly is theoretically an efficient method of executing parallel computation where information is encoded in DNA tiles and a large number of tiles can be self-assembled via sticky-end associations. Mao et al. [13] experimentally implemented the first algorithmic DNA tile self-assembly which performed a logical computation (cumulative XOR), however that study only executed two computations on fixed inputs. For the application in arithmetic, Brun [14] proposed and studied theoretically the systems that computed the sums and products of two numbers using the DNA tile self-assembly model, which enough revealed that DNA tile self-assembly had the basic computational ability; For the complex application in combinational problems, tile self-assembly has been proposed as a way to cope with huge combinational NP-complete problems, such as solving the satisfiability problem [15] by using 2D DNA self-assembly tiles, nondeterministically factoring numbers [16], deciding a system of subset sum problem [17]. But generally, the scale is limited to only moderate size problem at best, which further explores the power of computing using DNA tile self-assembly. Furthermore, this model can also be used in the cryptography. XOR computation on pairs of bits can be used for executing a one-time pad cryptosystem that provides theoretically unbreakable security [18].

It is well known that timetable problems [19] are very difficult and time consuming to solve, especially when dealing with large instances. The timetable problem is a combinatorial problem [20] consisting in finding an assignment of a fixed number of teachers to a fixed number of hours in a week, in such a way that a large number of given constraints are satisfied. And it is also known in general to be NP-complete [21]. For the most important, the timetable problems are subject to many strict constraints that are usually divided into two categories: “hard" and “soft" [22]. Hard constraints are rigidly enforced and have to be satisfied for the timetable problem. Soft constraints are those that are desirable but not absolutely essential. So it is difficult to generate a satisfactory solution within a short time. In order to avoid the disadvantage of their exponential computation complexity, here we mainly focus on the timetable problem based on DNA tile self-assembly, which is a kind of better technique and the model can successfully perform the problem with the operation time complexity of $\Theta(mn)$, parallely and at very low cost.

The rest of this paper is structured as follows: Section 2 describes the mechanism of self-assembly based on the DNA tiles in detail. Section 3 shows the process of performing the timetable problem by self-assembling. The conclusion will summarize the contribution of our work.

2 Algorithmic DNA tile self-assembly

Algorithmic DNA self-assembly is both a form of nanotechnology and a model of DNA computing. As a nanotechnology, the aim of algorithmic DNA self-assembly is to design tiles with carefully choosing glue types on their sides. Two tiles are said to be of different types if their sides have different glue types. Useful tile types are nontrivial to design but relatively easy to duplicate in large quantity. A key design challenge for algorithmic DNA tile self-assembly is to use only a small number of different tile types to assemble a target nanostructure to complete the corresponding computation.

2.1 Models for algorithmic DNA tile self-assembly

The tile assembly model extends the theory of Wang tilings [23] of the plane by adding a natural mechanism for growth. As a computational model, algorithmic DNA self-assembly encodes the input of a computational problem into DNA patterns and then manipulates these patterns to produce new DNA

patterns that encode the desired output of the computational problem. Informally, the model consists of a set of four sided Wang tiles whose sides are each associated with a type of glue. The bonding strength between any two glues is determined by a glue function. A special tile in the tile set is denoted as the seed tile. Assembly takes place by starting with the seed tile and attaching copies of tiles from the tile set one by one to the growing seed configuration whenever the total strength of attraction from the glue function meets or exceeds a fixed parameter called the temperature. Generally, the tile set and the seed configuration should be constructed before the biological operations together with the suitable temperature.

In addition, the tile assembly model [24] is a formal model of crystal growth. It was designed to model self-assembly of molecules such as DNA. Rothemund and Winfree [25] defined the abstract tile assembly model, which provides a rigorous framework for analyzing algorithmic self-assembly. Here, we mainly use the abstract tile assembly model to solve the timetable problem. Intuitively, the model has tiles or squares that stick or don't stick together based on various binding domain on their four sides. Figure 1 gives the structures of DNA tiles, mainly including the TAO and TAE tiles. Figure 1(a) describes the structure of TAO tile. Figure 1(b) shows the three TAO tiles joining diagonally. The TAE tiles and the corresponding abstract tiles can be seen in (c), (d) and (e). Figure 1(f) gives the structures which are assembled to form a compact lattice.

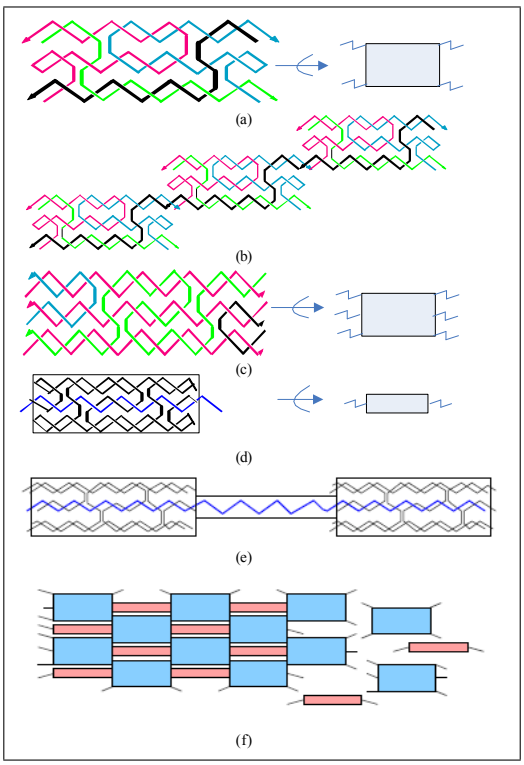

Figure 1: DNA Tiles. (a)Structure of TAO tile. (b)Three TAO tiles join diagonally. (c)Structure of TAE tile. (d)TX tile. (e)Two TAE tiles join. (f)The two types of tiles can assemble to form a compact lattice structure.

2.2 Computation by DNA tile self-assembly

Computation by self-assembly is the spontaneous self-ordering of substructures into superstructures driven by annealing of Watson-Crick base-pairing DNA sequences. Computation by DNA tile self-assembly entails the building up of superstructures from starting units such that the assembly process

itself performs the actual computation.

DNA tile self-assembly is also a highly parallel process, where many copies of different molecules bind simultaneously to form intermediate complexes. One might be seeking to construct many copies of the same complexes at the same time, as in the assembly of periodic 1D or 2D arrays; Alternatively, one might wish to assemble in parallel different molecules, as in DNA-based computation, where different assemblies are sought to test out the combinatorics of the problem.

A sequential or deterministic process of DNA tile self-assembly has three highly parallel instruction steps [4]. The first one is molecular recognition: elementary molecules selectively bind to others. The second is growth: elementary molecules or intermediate assemblies are the building blocks that bind to each other following a sequential or hierarchical assembly. The cooperativity and non-linear behavior often characterize this process. The third way is termination: a built-in halting feature is required to specify the completion of the assembly. In practice, their growth is interrupted by physical and/or environmental constraints. DNA tile self-assembly is a time-dependent process and because of this, temporal information and kinetic control may play a role in the process before thermodynamic stability is reached.

3 Implementing the timetable problem based on DNA tile self-assembly

In this section, we first give the definition of the timetable problem, then we mainly show the algorithm for solving the timetable problem based on DNA tile self-assembly, and concretely introduce the process how they can perform this problem. Finally, examples of success and failure in the tile attachments are given to demonstrate the reasonability and validity of the algorithm.

3.1 The timetable problem

The typical timetable problem consists in assigning a set of activities/actions/events (e.g. work shifts, duties, classes) to a set of resources (e.g. physicians, teachers, rooms) and time periods, fulfilling a set of constraints of various types. Constraints stem from both nature of timetable problems and specificity of the institution involved. In other words, timetable or planning is a process of putting in a sequence or partial order a set of events to satisfy temporal and resource constraints required to achieve a certain goal, and is sometimes confused with scheduling, which is the process of assigning events to resources over time to fulfill certain performance constraints. However, many scientists consider scheduling as a special case of timetable and vice versa [26].

In this paper, we solve a special kind of timetable problem which is the coursetable problem [27]. The problem consists in scheduling courses for a set of courses in a university, taught by available teachers in a given period composing a number of weeks, and in available classrooms. Although the constraints of timetable problem vary from case to case, one can classify all constraints into hard constraints and soft constraints. Hard constraints must be strictly satisfied because any timetable that violates just one will become useless. A timetable that violates some soft constraints can still be usable although it may cause some inconvenience to the users. It is often very difficult to satisfy all the soft constraints in a real life. Some concrete definitions of hard constraints and soft constraints in a coursetable problem will be given as follows [28]. Some examples of hard constraints are:

HC0 - A teacher can only teach in a single place at a time.

HC1 - A teacher can only give one course at a time.

HC2 - A room can only host one course at a time.

HC3 - A student can only attend one course at a time.

HC4 - Room capacities must be respected.

HC5 - No more than a teacher is scheduled to teach in a room each time.

HC6 - Each subject is scheduled in a proper room (for example, a laboratory needs a proper equipment).

HC7 - Every teacher must have scheduled all his hours.

HC8 - Every student must have scheduled all his hours.

We denote the fact that some conditions can be deduced from other constraints. For example, HC0, HC2→ HC1. Some examples of soft constraints are:

SC0 - The courses should be scheduled in the morning and the seminaries and laboratories in the afternoon.

SC1- Some courses are scheduled with a prior consideration.

SC2 - As much as possible the preferences of the teachers and the ones of the students should be respected.

3.2 Solving the timetable problem based on DNA tile self-assembly

Here, we mainly introduce the algorithm for implementing the timetable problem based on DNA tile self-assembly. First, the timetable problem can be converted into the graph edge coloring problem, then the tile self-assembly model is used to solve the graph edge coloring problem with some constraints including mainly constructing three small systems which are nondeterministic assigning system, copy system and detection system, thus the timetable problem can be successfully carried out. Examples can be given to indicate how the tile self-assembly model performs in this problem.

The graph edge coloring problem

Let G be an undirected graph where V is the set of vertices and E is the set of edges. Mathematically, an assignment of colors to the edges of a graph G(one color to each edge so that adjacent edges are assigned different colors) is called a coloring of G. Edges with a same color define a color class. A k-coloring of G is proper if incident edges have different colors; that is, if each color class is a matching, otherwise conflicts happen. A coloring with at least one conflict is called an infeasible coloring. A graph is k-edge-colorable if it has a proper k-edge-coloring. For the given coloring of a graph G, a set consisting of all those edges assigned the same color is referred to as a color class.

In this study, the timetable problem can be converted into the graph edge coloring problem. First, a complete bipartite graph, denoted as $K_{m,n}$, is a graph consisting of two sets of vertices, one with m vertices and the other with n vertices. There is exactly one edge from each vertex in the one set to each vertex in the other set. There are no edges between vertices within a set. Then we give the bipartite graph from the arrangement matrix of the timetable problem. Second, the hard and soft constraints can be considered as the constraints of the graph edge coloring problem. According to the graph theory, the feasible solutions of the edge colorings are the arrangements of courses in the timetable problem. Here, we mainly propose non-deterministic algorithm to solve the graph edge coloring problem by using the massive parallelism possible in DNA tile self- assembly, thus the timetable problem with some given constrains can be successfully solved.

In the process of implementing the tile self-assembly systems, many assemblies happen in parallel by creating billions of billions of copies of the participating DNA tiles, so this is simulated by an exponential number of DNA assemblies which can be converted into the space occupied by the DNA molecules, thus we expect that the procedure will run in parallel on all possible colorings. In this case, there are many possible valid tilings, any or all of which may be produced. When tiles are implemented by real molecules, one would expect a set of tiles to nondeterministically generate a combinatorial library of input assemblies, and then a deterministic set of rule tiles could evaluate each input assembly to determine whether it represents the desired answer.

The nondeterministic assigning system

Non-determinism implies that at some steps the algorithm makes a non-deterministic choice. Of course, there are many differences between the deterministic computation and nondeterministic computation. In terms of deterministic computation, it can be defined as a tile system to produce a unique final seed configuration if for all sequences of tile attachments, all possible final configurations are identical. Comparing to the deterministic computation, the nondeterministic computation is a system in which different sequences of tile attachments can attach different tiles in the same position. Intuitively, a system nondeterministically computes a function if at least one of the possible sequences of tile attachments produces a final configuration, which contains the computation results. Furthermore, in many implementations of the tile assembly model that would simulate all the nondeterministic executions at once, it is useful to be able to identify which executions succeed and fail in a way that allows selecting only the successful ones.

The nondeterministic assigning system can give a color set to the edges of the graph. First, the edges of the given graph can be labeled as "$e_1, e_2, \cdots, e_m$", the vertices can be noted as $X_i (1 \leq i \leq n)$. Here, m, n is the number of edges and vertices of the graph respectively. Each edge in the graph can be nondeterministeically obtained one color. The same edge connecting different vertices should share the same color. If there is only one edge which is adjacent to the vertex, the information about the vertex and the edge needn't be arranged on the rightmost column in the seed configuration, but should be assigned with one color at the bottom of the seed configuration of the nondeterministic assigning system.

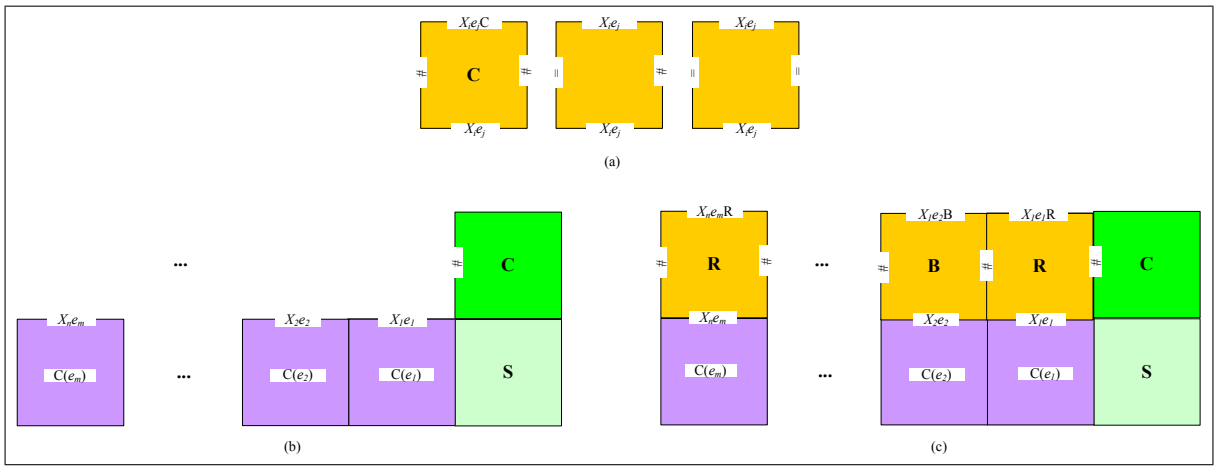

Figure 2: The framework of the nondeterministic assigning system. (a) The basic tile types of this system. (b) The seed configuration of the system. (c) An example of the nondeterministic assigning system.

The color set of the edges can be nonderministically generated by the tile self-assembly configuration. Here, suppose the edge e_j is on the vertex X_i which is labeled as X_ie_j. $C(e_j)$ is denoted as the edge e_j with the color "C". The same edge on the remainder vertices can pass the information from the bottom to the upper in the tile and can obtain the same color. The basic tile types of this system can be shown in Figure 2(a). Figure 2(b) shows the seed configuration of the system. Figure 2(c) is an example of the nondeterministic assigning system which can assign a color set to the edges.

The copy system

The copy system mainly carries out three functions by designing three basic tile types which can be shown as follows in Figure 3. Here, X_ie_j denotes the edge e_j is adjacent to the vertex $X_i (1 \leq i \leq n)$. "C" is the color of the edge e_j on the vertex X_i.

The first function is to pass the color of the edge given by the nondeterministic assigning system to the same edge on different vertices, so it can make sure that the same edge adjoining different vertices has the same color. The tile type is labeled with blue. If the edge is adjacent to different vertices, "$X_i e_j C*$" and $X_i e_j$ should be passed to the left and the upper of the tile respectively. Otherwise, the color "C" can be passed to the edge $X_i e_j$. At the same time, if the input at the bottom of the tile includes the information of the colors, the outputs are only needed to copy the information of the inputs from left to right, and synchronously from bottom to upper in the tile.

The second is to copy the possible colorings generated by the nondeterministic assigning system from the bottom of the seed configuration to the uppermost of the self-assembly complexes. After the edges sharing the common vertex have been checked the colorings by the detection system, there would be a condition that the edges adjoining different vertices $(k \neq i)$ and with different colors $(C_1 \neq C_2)$ will meet together, but they have no need to be checked the coloring and also should be passed to the left tiles which is shown in the second tile type with the color turquoise.

The third is to copy the information with the edges which are adjacent to the vertices on the rightmost column to the left tiles, so the detection system can check up whether colorings of the edges sharing the common vertex are feasible, and synchronously, they also should be passed to the upper in the tile. Here, $X_i e_j$ is the edge which has at least two adjacent edges sharing a common vertex. Once a vertex has l adjacent edges, $(l-1)$ edges with the labels smaller then should be arranged on the rightmost column in the seed configuration of the problem, but all the edges should be at the bottom of the seed configuration. The tile type can be shown as follows with the color rosiness.

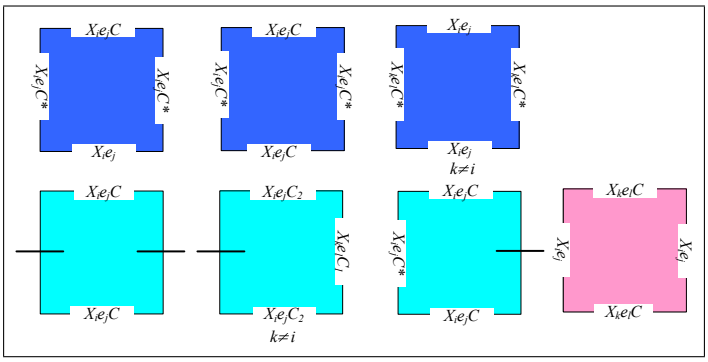

Figure 3: The basic tile types of the copy system

The detection system

The key to the detection system is to make one system implement the checking operations. If the adjacent edges sharing the common vertex have different colors, the feasible coloring of the edges for the vertex has been completed with the symbol "Ok". Once the edges sharing one common vertex have at least two same colors, the self-assembly complexes will stop to grow with the information "No tile can match" and the coloring is not feasible. Here, the coloring of the adjacent edges for each vertex should be satisfied with the same constraints. For the timetable problem, the hard and soft constraints can be described by the constraints of the edges coloring for the corresponding graph.

When the edges which are adjacent to the vertices on the rightmost column and the coloring of each edge from the copy system are passed to the detection system, it can check up whether the coloring is feasible or not. If the comparison result at this step is "Ok", it should be passed to the left tiles to continuously make the next color checking until the edges don't share the same vertex, then it doesn't check the coloring with the left tiles any more, and synchronously the colors of the edges at the bottom of the seed configuration in the nondeterministic assigning system should be passed to the higher layers.

Here, e_k and $e_j(k \neq j)$ are the adjacent edges on the common vertex X_i, and they have different colors, so the comparison result is "Ok". For the second tile type, the edge e_j which is adjacent to the vertex labeled as X_i meets the color "C", then it can pass "$X_i e_j C$" to the right in the tile, and the value of the tile is the color "C" of the edge e_j.

If the result is "No tile can match", the self-assembly complexes can't grow any more and the input colors of the edges are not the feasible solutions of the graph edge coloring problem. The formula of the detection system can be described as follows:

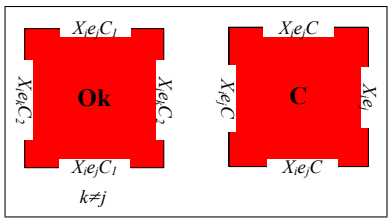

Figure 4: The basic tile types of the detection system

Here, this system will use the L-configuration to encode inputs, and produce its output on the top row of an almost complete rectangle. Therefore, systems could chain results together. The input structure encodes the edges colorings of the graph on the bottom row and encodes the edges and their adjacent vertices on the rightmost column. The output tiles needs three different kinds of tiles as follows in Figure 5. The pink tile shows the edge e_j with the color "C" adjoining the vertex X_i which is the feasible coloring for the graph. Only if all the edges adjoining different vertices have feasible colorings, the result is "Success", and the feasible solution can't be obtained otherwise.

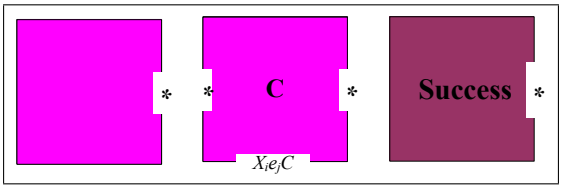

Figure 5: The output tiles of the timetable problem

Here, we design the algorithm to implement the timetable problem as following steps:

Step 1: Convert the timetable problem into the graph edge coloring.

Step 2: Generate all possible input combinational colors of the edges to the given graph for the timetable problem with some constraints.

Step 3: According to the rules for dealing with the constraints using the massive parallelism of DNA self-assembly to check up whether all the possible inputs are the feasible colorings for the edges in the graph. In this process, the copy system can pass the color of each edge in the graph from the bottom of the seed configuration to the higher layers, and synchronously copy the information of edges which are adjacent to the vertices, then the detection system can judge whether the colorings of the edges sharing the common vertices are feasible or not.

Step 4: Reject all infeasible solutions according to constraints of the edge coloring and reserve all feasible solutions, therefore we can obtain feasible solutions of the graph edge coloring problem which are also the solutions of the timetable problem.

Step 5: Reading the operation result is done by the reporter strand method. Under certain biological operations, we can obtain all the result strands which run through all the results of the feasible colors of the edges for the given graph. Each report strand records the result of one feasible input. The strands can be amplified by polymerase chain reaction using the primers to ligate each end of the long reporter strand. Then through gel electrophoresis and DNA sequencing, we can read out the result strands of

different lengths representing the information with the feasible coloring results. Finally, we can easily get the feasible solutions of the timetable problem.

The actual implementation detail is not discussed here since they fall outside of the scope of this paper. However, we believe that we make no arbitrary hypotheses. In fact, our work is based on the achievements that come with DNA tiling computation in general.

Examples of the timetable problem

Here, we take an example of timetable problem to verify the validity of our method. Suppose there are three teachers X_1, X_2, X_3, and four classes Y_1, Y_2, Y_3, Y_4. The arrangement matrix of the courses is shown as follows:

$$P = \begin{array}{c} \\ X_1 \\ X_2 \\ X_3 \end{array} \begin{array}{c} \begin{array}{cccc} Y_1 & Y_2 & Y_3 & Y_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

The timetable problem has the hard constraints are:

HC0 - A teacher can only teach in a single place at a time.

HC1 - A teacher can only give one course at a time.

HC2 - A room can only host one course at a time.

HC3 - A student can only attend one course at a time.

HC4 - No more than a teacher is scheduled to teach in a room each time.

HC5 - The teacher X_3 should give a course to the class Y_2, which is arranged in the second period in the morning time.

HC6 - There are enough rooms for the courses where the students attend. Some soft constraints are:

SC0 - Some courses are scheduled with a prior consideration.

SC1 - As much as possible the preferences of the teachers and the ones of the students should be respected.

SC2 - If possible, the order of the courses classes taken Y_j are more earlier than Y_{j+1}.

First, we should convert the timetable problem into the graph edge coloring problem with some constraints. The bipartite graph from the arrangement matrix of the timetable problem can be shown in Figure 6. All the edges of the graph are "e_1, e_2, e_3, e_4, e_5, e_6, e_7" and the vertices are "X_1, X_2, X_3, Y_1, Y_2, Y_3, Y_4".

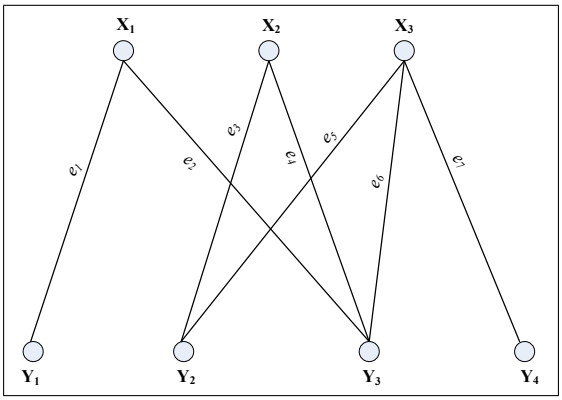

Figure 6: The bipartite graph from the arrangement matrix of the timetable problem

Second, according to the method introduced above, we need construct the basic tile types in each of the three small systems and they are the same as the tiles described above and the seed configuration, which can be shown in Figure 7. When all the tiles and the seed configuration are prepared, we put them

together into the reaction buffer. According to the DNA tiles prepared and the mechanism of algorithmic DNA tile self-assembly through Watson-Crick base pairing, the self-assemble process starts at the same time with the connector tiles, so the final stage can be seen in Figure 8.

We also can see that the process of the three small systems performing. The nondeterministic assigning system can give a color set "RBRYBRY" to all the edges of the graph "$e_1, e_2, e_3, e_4, e_5, e_6, e_7$" which are adjacent to the vertices "X_1, X_2, X_3, Y_1, Y_2, Y_3, Y_4" respectively. All the edges on different vertices should be arranged at the bottom of the seed configuration no matter whether the vertices adjoin only one edge or not. At the same time, the edges "$X_1e_1, X_3e_5, Y_2e_3, Y_3e_2, Y_3e_4$" are on the rightmost column of the seed configuration. The copy system can pass the colors of the edges from the bottom of the seed configuration to the upper layers, and pass the edges and their adjacent vertices on the rightmost column to the left tiles, so that the detection system can check whether the colorings of the edges sharing one common vertex are feasible. The vertex X_1 which has two adjacent edges e_1 and e_2 with different colors "R" and "B" respectively is checked up the feasibility of the colorings by the detection system and the result is "Ok". For the vertex X_3, it has three adjacent edges e_5, e_6 and e_7 which are at the bottom of the seed configuration. One of the two edges e_5, e_6 with the smaller subscripts than e_7 are on the rightmost of the column. The detection system only need verify the colors of e_5 and e_6, e_5 and e_7, e_6 and e_7, and the three comparison results are all "Ok". The method of checking the colorings of other vertices is also the same.

Finally, to output the computation result, we would implement a modification of the standard sequence-reading operation that uses a combination of PCR and gel electrophoresis. On adding these tiles, and allowing them to anneal, then we get the final tile assembly. On adding ligase to seal the bonds, we will have a single strand of DNA passing through the tiles in the final output layer, which encodes the colorings of the edges. This single strand begins with the unique nucleotide sequence labeled "Success". Therefore, the feasible assignment of the edge colorings can be obtained if and only if the symbol "Success" appears in the result DNA strand. Through using the operations, we can extract the strands of different lengths representing the output tiles in the result strands. In this example, we can obtain the feasible solution of the "RBRYBRY". The color sets "R", "B" and "Y" have the corresponding relationship with the edge sets "e_1, e_3, e_6" , "e_2, e_5" and "e_4, e_7" which are also "X_1Y_1, X_2Y_2, X_3Y_3", "X_1Y_3, X_3Y_2" and "X_2Y_3, X_3Y_4". Thus the feasible solution of the timetable problem which is also the arrangement of the courses can be described as: "X_1Y_1, X_2Y_2, X_3Y_3" are arranged in the first period, "X_1Y_3, X_3Y_2" and "X_2Y_3, X_3Y_4" in the second and third period respectively which are satisfied with the constraints in the problem.

For the nondeterministic algorithm, we give the same example to show the failure in attaching tiles in Figure 9 and don't get the right results. If the nondeterministic assigning system gives a color set "RBRYBBY" to the edges "$e_1, e_2, e_3, e_4, e_5, e_6, e_7$" respectively, there will be some conflicts in the process of the growth for the assembly complexes. When the detection system checks up the coloring of the vertex X_3, the colorings of the edges e_5 and e_6 are the same which are both "B", so the conflict generates and the result is "No tile can match", thus the self-assembly complexes can't grow any more, therefore, the coloring of the edges assigned is the infeasible solution of the problem. It means that "X_1Y_1, X_2Y_2", "X_1Y_3, X_3Y_2,X_3Y_3" and "X_2Y_3, X_3Y_4" are not the feasible arrangements of the courses, here there is a conflict that the teacher X_3 can't give a course in two different classes Y_2 and Y_3 at the same period.

Complexity analysis

The complexity of the design is considered in terms of computation time, computation space and the number of distinct tiles required. Generally, suppose there are m teachers, and n classes for the timetable problem.

It is obvious from the given examples that the upper bound of the computation time T is $T = m(n-1)+n(m-1)+mn+4+mn+mn+2 = \Theta(mn)$.

The upper bound of the computation space S taken for each assembly is the area of the assemble

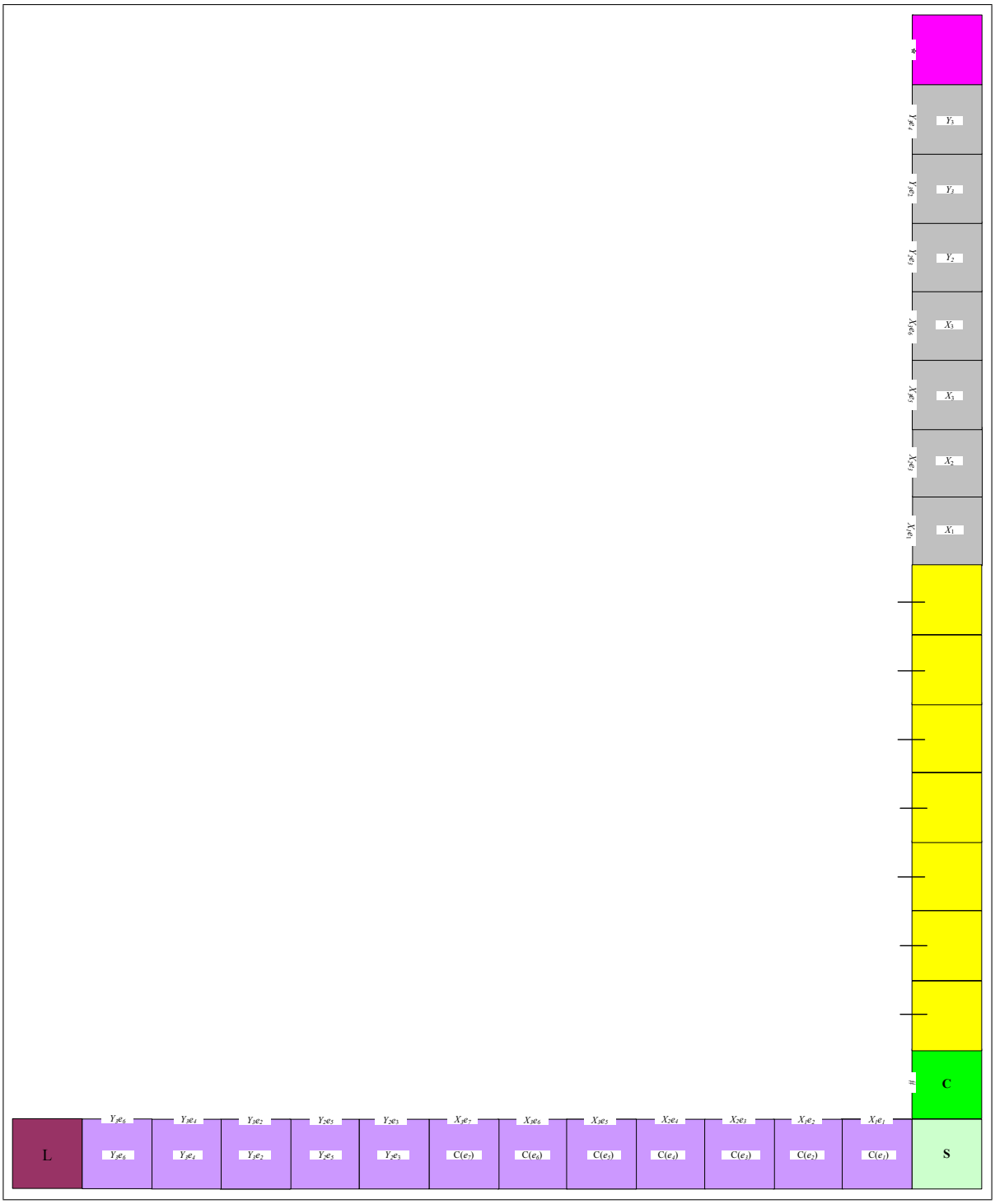

Figure 7: The seed configuration of the timetable problem in the example

Figure 8: The final stage of the successful example for the problem

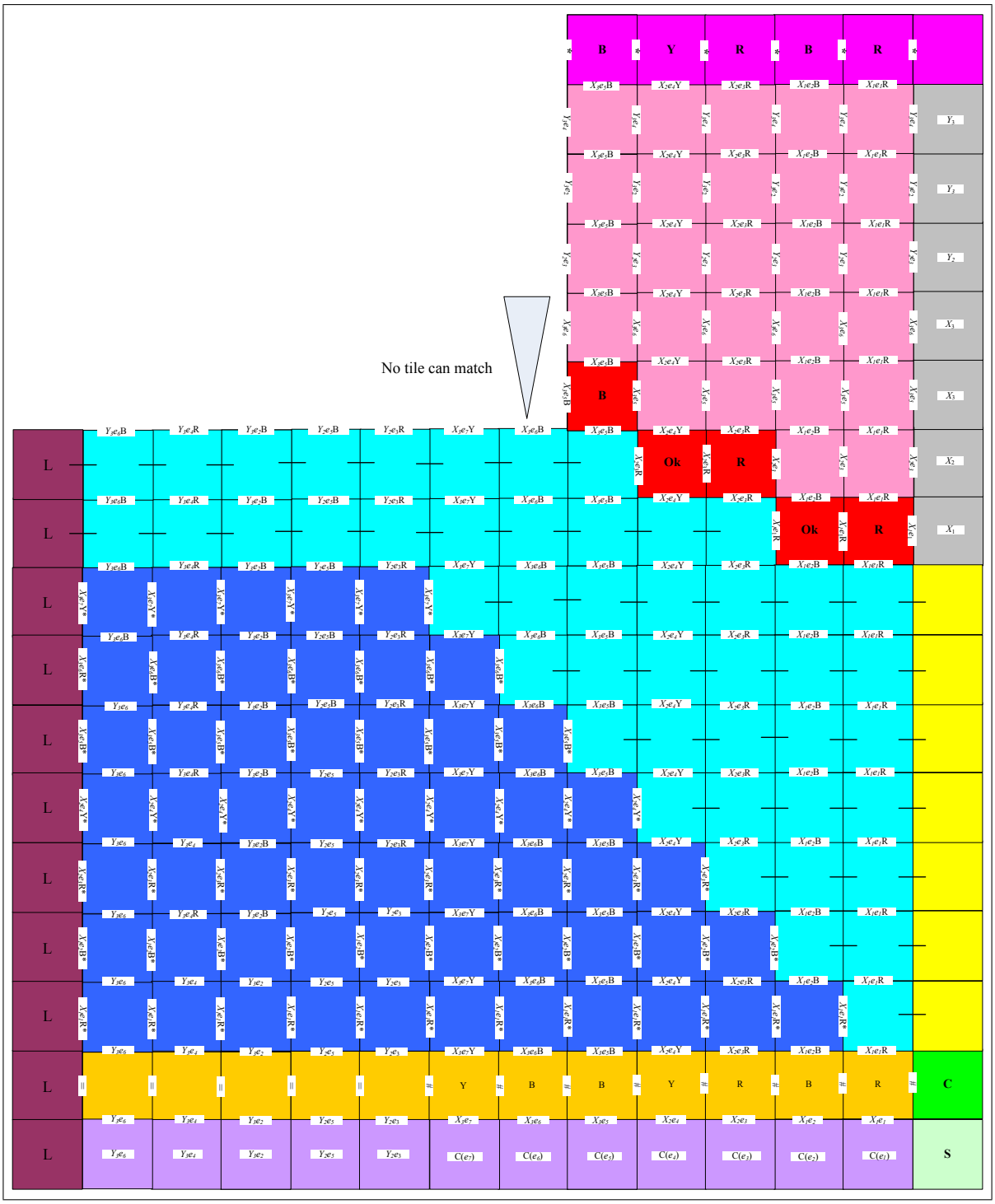

Figure 9: The failure of the example for the problem because of infeasible coloring sets

complexes represented by $S = [m(n-1)+n(m-1)+mn+4]*[mn+mn+4] = \Theta(m^2n^2)$ which is upper-bounded polynomially to the number of variables.

Finally, suppose the graph G which is converted from the given timetable is k-edge-colorable. The upper bound of tiles needed contain the following tiles:

Boundary tiles. These tile types include the input boundary tiles and computation boundary tiles. According to the number of the edges in the graph, there are $2mn$ tiles for the input tiles at the bottom of the seed configuration, and $[m(n-1)+n(m-1)+3]$ on the rightmost column. The computation boundary tiles contain $(kmn+km+3)$ types. So the upper bound of the boundary tiles is $[2mn+m(n-1)+n(m-1)+kmn+km+6]$.

Computation tiles. For the assigning operations, there must be $(kmn+mn)$ tiles with the upper bound, and they are shown in Figure 2. For the copy system which can be seen in Figure 3, the upper bound of the tile types is $[km(n-1)+kn(m-1)+kmn+kmn+kmn]$. For the detection system in Figure 4, there must be $[km(n-1)+kn(m-1)+kmn]$ tiles with the upper bound. Thus the upper bound of the computation tiles is $[9kmn+mn-k(m+n)]$.

Output tiles. Finally, there must be some tiles to output the results. The upper bound is $(2kmn+2)$ and the tiles are shown in Figure 5.

Summing up all the tile types, because the value of k can be determined for the given timetable problem, so we can have the upper bound of the total number of tiles: $[2mn+m(n-1)+n(m-1)+kmn+km+6]+[9kmn+mn-k(m+n)]+(2kmn+2)=\Theta(mn)$.

4 Summary and Conclusions

DNA tile self-assembly is looked forward to many applications in different fields. In this paper, we show how the DNA self-assembly process can be used for solving the timetable problem. The advantage of our method is that once the initial strands are constructed, each operation can compute fast parallelly through the process of DNA self-assembly without any participation of manpower, thus the algorithm is proposed which can be successfully implemented the timetable problem with the operation time complexity of $\Theta(mn)$, parallely and at very low cost. A limitation of the algorithm, which is common for most DNA computations, comes from the fact that the exponential dimension of the problem has been pushed into the physical space (volume) occupied by the DNA molecules. This will eventually become a restrictive factor. The input size and thus the DNA volume can't grow forever. This implies an upper bound to the size of instances that can be solved in practice.

While DNA tile self-assembly suffers from high error rates, the possible sources of errors are, either an error in constructing the tiles, or an erroneous binding of tiles, methods of error control and error correction may be used to decrease the error rates in the computation of DNA tile self-assembly model. Many experimental results in DNA tile self-assembly have not appealed to the advantages of crystal growth; however, these early works on the fundamentals of self-assembly and the physical experimental evidence of actual DNA tile crystals suggest a bright future for DNA tile self-assembly. The field of nanotechnology holds tremendous promise, but many technical hurdles will have to be overcome before algorithmic DNA tile self-assembly can be developed into a practical commercial technology. If the molecules and supramolecules can be controlled at will, then it may be possible to achieve vastly better performance for computers and memories. So we can see that the DNA tile self-assembly model has various applications in many fields and it also might open up a host of other applications in materials science, medicine, biology and other ways.

Acknowledgments

The work was supported by the National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, 60533010 and 60803113), 863 Program of China (2006AA01Z104), and program for New Century Excellent Talents in University (NCET 05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20070001020), Chenguang Program of Wuhan (200750731262), and the open fund of Key Lab. for Image Processing and Intelligent Control (No.200703).

Bibliography

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, vol. 266, pp. 1021-1024, 1994.

[2] L.Q. Pan, J. Xu, Y.C. Liu, A Surface-Based DNA Algorithm for the Minimal Vertex Problem, *Progress in Natural Science*, vol. 13, pp. 81-84, 2003.

[3] L.Q. Pan, G.W. Liu, J. Xu, Solid phase based DNA solution of the coloring problem, *Progress in Natural Science*, vol. 14, pp. 104-107, 2004.

[4] A. Carbone, N.C. Seeman, Molecular Tiling and DNA Self-assembly, *Springer-Verlag Berlin Heidelberg*, LNCS 2950, pp. 61-83, 2004.

[5] N.C. Seeman, DNA nanotechnology: novel DNA constructions, *Annu. Rev. Biophy. Biomol. Struct.*, vol. 27, pp. 225-248, 1998.

[6] C. Mao, W. Sun, N.C. Seeman, Designed two dimensional DNA Holliday junction arrays visualized by atomic force microscopy, *J. Am. Chem. Soc.*, vol. 121, pp. 5437-5443, 1999.

[7] E. Winfree, On the computational power of DNA annealing and ligation, *DNA Based Computers*, pp. 199-221, 1996.

[8] T. Eng. Linear self-assembly with hairpins generates the equivalent of linear context-free grammars. *3rd DIMACS Meeting on DNA Based Computers*, *Univ. of Penn.*,1997.

[9] R. Barish, P. Rothemund, E. Winfree, Two computational primitives for algorithmic self-assembly: Copying and counting, *Nano Letters*, vol. 12, pp. 2586-2592, 2005.

[10] Pablo Moisset de Espane′s, Ashish Goel, Toward minimum size self-assembled counters, *Springer Science Business Media B.V.*, 2008.

[11] P. Rothemund, N. Papadakis, E. Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, *PLoS Biology*, vol. 12, pp. 2041-2053, 2004.

[12] M. Cook, P. Rothemund, E. Winfree, Self assembled circuit patterns, *DNA*, pp. 91-107, 2004.

[13] C. Mao, T.H. LaBean, J.H. Reif, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, *Nature*, vol. 407, pp. 493-496, 2000.

[14] Y. Brun, Arithmetic computation in the tile assembly model: Addition and multiplication, *Theoretical Computer Science*, vol. 378, pp. 17-31, 2006.

[15] G.L. Michail, T.H. LaBean, 2D DNA self-assembly for satisfiability, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. vol. 44, pp. 139-152, 1999.

[16] Y. Brun, Nondeterministic polynomial time factoring in the tile assembly model, *Theoretical Computer Science*, vol. 395, pp. 3-23, 2008.

[17] Y. Brun, Solving NP-complete problems in the tile assembly model, *Theoretical Computer Science*, vol. 395, pp. 31-36, 2008.

[18] A. Gehani, T.H. LaBean, J.H. Reif, In DNA Based Computers: Proceedings of a DIMACS Workshop, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.

[19] D. Werra, An introduction to timetabling. *European Journal of Operations Research*, vol. 19, pp.151-162, 1985.

[20] D. Abramson, Constructing school timetables using simulated annealing: Sequential and parallel algorithms. *Management Science*, vol. 37, pp. 98-113, 1991.

[21] A. Colorni, M. Dorigo, V. Maniezzo, Genetic algorithms and highly constrained problems: the time-table case. *In: H.P.Schwefel and R.Manner(eds). Parallel Problem Solving from Nature. Proceedings of 1st Workshop, PPSN 1, LNCS 496, Dortmund, Germany, 1-3 October. Springer-Verlag,* pp. 55-59, 1991.

[22] L. Gaspero, A. Schaerf, Tabu search techniques for examination timetabling. *In Proceedings of the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT 2000)*, *Springer-Verlag*, LNCS 2079, pp. 104-117, 2001.

[23] H. Wang, Proving theorems by pattern recognition, *I. Bell System Technical Journal*, vol. 40, pp. 1-42, 1961.

[24] E. Winfree, Algorithmic self-assembly of DNA, Ph.D.Thesis, Caltech, Pasadena, CA, June 1998.

[25] P. Rothemund, E. Winfree, The program-size complexity of self-assembled squares, *ACM Symposium on Theory of Computing (STOC)*, pp. 459-468, 2001.

[26] Mihaela Oprea. MAS_UPUCT: A Multi-Agent System for University Course Timetable Scheduling. *International Journal of Computers*, *Communications* & *Control*, Vol. II, pp. 94-102, 2007.

[27] Zhipeng L., Jin-Kao Hao. Adaptive Tabu search for course timetabling. *European Journal of Operational Research*, doi:10.1016/j.ejor.2008.12.007, 2009.

[28] A.R. Mushi, Mathematical programming for mulations for the examinations timetable problem: the case of the university of DAR ES SALAAM. A*frican Journal of Science and Technology (AJST) Science and Engineering Series*, Vol. 5, pp. 34-40, 2004.