A new approach to advance the DNA computing

Zhiquan Frank Qiu∗, Mi Lu

Department of Electrical Engineering, Texas A&M University, College Station, TX 77843-3128, USA

Received 21 September 2001; received in revised form 17 December 2002; accepted 23 January 2003

Abstract

It has been evidenced that DNA computing can solve those problems which are currently intractable on even the fastest electronic computers. The algorithm design for DNA computing, however, is not straightforward. Both a significant knowledge of the DNA molecule and a strong background in computer engineering are required to develop efficient DNA computing algorithms. The existing models based on which a few DNA computing algorithms were developed are not sufficiently powerful and robust to attract potential users.

In this paper, a new DNA computing model is introduced based on which new algorithms are developed to solve the 3-Coloring problem. These new algorithms are presented as vehicles for demonstrating the advantages of the new model, and they can be expanded to solve other NP-complete problems. Our new algorithms can significantly speed up computation and therefore achieve a better time performance. Furthermore, with the given resource, our algorithms can solve problems of much greater size as compared to existing DNA computing algorithms. In addition, the error rate can be greatly reduced by applying our new algorithms. All of the advantages provided by the new model make DNA computing very efficient and attractive in solving computational intense problems.

© 2003 Elsevier B.V. All rights reserved.

Keywords: DNA computing; Molecular computing; Coloring problem; Parallel processing; Speedup; Error resistant; Larger size problem

1. Introduction

1.1. Motivation

A single strand of DNA is a string consisting of a combination of four different base nucleotides: adenine (A), cytosine (C), guanine (G) and thymine (T). When attached to deoxyribose, these base nucleotides can be strung together to generate long sequences. Each single string can be paired up with a complementary string to form a double helix. This pair-up only occurs under the Watson-Crick (WC) complement rule. That is, A only pairs with T and G only pairs with C. Also the double strand can be separated by heating. The dissociated strands separate from each other without breaking the chemical bonds that hold the nucleotides together along the single strand. Either one of these denatured single strands or both together can be used for further operations because they perfectly complement one another. One good example of this idea is the polymerase chain reaction (PCR) method. It is used to initialize test tubes in many DNA computing algorithms by making numerous identical strands through a repetition of the above procedure.

Since Watson and Crick’s discovery, many ways to manipulate DNA have been developed. Biological techniques include the use of enzymes for cutting and pasting, and the polymerase chain reaction for the...
reproduction of DNA strands. Biotechnological techniques include the selective filter, tagging and DNA sequencing. Together, these developments enable us to use DNA as a modifiable storage medium—a kind of memory. These developments also allow us to use these techniques as operations on that memory in order to implement algorithms.

Because the DNA strands can be used to encode information and DNA bio-operations are completed based on interactions between the strands, each DNA strand can be counted as one processor. Numerous strands are involved in DNA bio-operations and interactions between one another occur simultaneously, which can be viewed as a realization of massive parallel processing.

DNA was first used for computations in 1994. In his groundbreaking Science article, Adleman described his experiment that solved a seven-node instance of the Hamiltonian path problem from graph theory. (The problem involves finding a path containing all nodes only once, through a mathematical graph.) He devised a code for the edges of a graph based on the encodings of their nodes.

As a result, Adleman produced sequences that corresponded to candidate solution paths by randomly gluing together sequences of single nodes. By producing enough of these sequences, all candidate solutions were constructed. This construction was done in parallel. In other words, all strands underwent the reactions simultaneously.

Through a sequence of filtering steps, strands that were of the wrong length or that did not contain all the required nodes were eliminated. Only those strands that corresponded to actual solutions were kept. This filtering was also done concurrently on all strands. The fact that DNA strands remained after this process indicated that solutions to the problem existed; by sequencing the remaining DNA, a single solution was decoded.

Since Adleman [1] solved one 7-vertex instance of the Hamiltonian path problem, a well-known representative of NP-complete problems, the major goal of subsequent research in DNA computing area is to develop new techniques to solve NP-complete problems that cannot be solved by current electronic computers in a reasonable amount of time. NP-complete problems are those problems for which no polynomial-time algorithm has yet been discovered, in contrast to polynomial-time algorithms whose worst-case run time is O(n^k) for some constant k, where n is the size of the problem.

Consider that 1 l of water can hold 10^{22} DNA strands. The potential computing power is significant, and this recognition raises the hope of solving the problems currently intractable on electronic computers. Rather than using electronic computers on which the time needed to solve NP-complete problems grows exponentially with the size of the problem, DNA computing technology can solve these problems within a time proportional to the problem size. An NP-complete problem which might take thousands of years to solve on current electronic computers would only take a few months if the existing DNA computing techniques were adopted.

As indicated in a few articles [9,10,12,13,15,17], most DNA computing algorithms are based on certain developed DNA computing models. These very popular models are: the sticker based model [18,19], the surface based model [16,20] and the self-assembly based model [21,22]. The problem with the sticker based model is that the stickers annealed to the long strand may fall off during bio-operations, causing a very high error rate. The limitation of the surface based model is that the scale of computation is severely restricted by the two-dimensional nature of the surface based computation. The shortcoming of the self-assembly based model is that it makes use of biological operations which are not matured.

In this paper, a new DNA computing model is introduced which can eliminate these problems. Based on this model, algorithms can be designed to solve NP-complete problems in a significantly reduced time. If an electronic computer capable of 10^6 operations per second is used, it would take 10^{19} years to solve the 3-Coloring problem for a graph with 256 vertices and 569 edges, when using the fast Biegel and Eppstein’s algorithm [3]. Nevertheless, using the current model of DNA computing algorithms, the problem can be solved in 2 months assuming that the average DNA operation takes 20 min [2,3]. Inspiringly, based on our newly introduced model, it would take approximately 2 days to finish the process with our new algorithm. In addition, with the available solution space, our new model could also increase the size of the problems solvable by DNA computing algorithms. Even more attractively, our new model can greatly reduce the error
rate which has always burdened the development of DNA computation.

1.2. Our new model

Our new model adopts only mature DNA biological operations [1,15]. The following basic principle operations: synthesis, ligation, separation and detection are selected for building this new model.

Synthesis $I(P, \pi)$: To generate a pool of coded strands, $P$, following criteria $\pi$. Strands are coded differently for different applications using the four base nucleotides: A, G, T and C. A set is defined as a group of strands, and the container holding a set of strands is called a pool. If the criteria is the colors of a node in a graph, then a pool of strands coding all the possible colors for the node is expected after synthesis. In the graph coloring problem, the strand is encoded for the colors of a number of nodes. Here, a few consecutive nucleotides on the strand coded for the color of one node form a region. For example, in Fig. 1, one strand consists of three regions such that $s = \{RBR\}$ where $(CCAAG)$, $(AATTC)$ and $(CCAAG)$ represent the colors for three nodes as Red ($R$), Blue ($B$) and Red ($R$), respectively.

Ligation $L(P_1, P_2)$: To bind strands in pool $P_1$ with strands in pool $P_2$. Each code $s_1$ in $P_1$ is ligated to every other code $s_2$ in $P_2$. If the strands in $P_1$ represent the codes $[s_1] = \{1, 2, \ldots, e\}$, where $s_1 \in P_1$ and those in $P_2$ represent the codes $[s_2] = \{1, 2, \ldots, d\}$, where $s_2 \in P_2$, after the ligation, the ligated strands are stored in $P_1$ and they represent the codes $[s_2] = \{1, 2, \ldots, e \times d\}$, where $s_k = s_1s_2$, for $k = i + (j - 1) \times e$. $P_1$ and $P_2$ are separated by heating in order to make the paired strands apart from each other without breaking the chemical bonds that hold the nucleotides together inside the single strand. The strands in the pool containing a region that complements to the probes will be hybridized to, and captured by the probes, while all those without the region will remain in the pool [18].

A gel-based separation technique for DNA computing [5] has been developed which uses gel-layer probes instead of the bead to capture the strands. The capture layer only retains the strand with a region complementing to the probe when it is cooled down, and will let all strands pass when the layer is heated up. The advantage of using gel-based probes over bead-based probes is that the gel-based method is more accurate when capturing DNA molecules. In Fig. 2 which illustrates the gel-based separation, a set of strands run from the left side buffer to the right. At each capture layer, the temperature is cold in order to capture the desired strands, and all unwanted strands are passed through into one pool. Then the temperature is raised to let all desired strands in the layer pass into another pool. The strands from the left buffer are separated and stored in two different pools.

Detection $D(P)$: To check if there is any strand left in the pool, $P$. If the answer is "yes", the strands in the pool should be decoded.

The rest of this paper is organized as follows: Section 2 gives an introduction to our new algorithm.
based on the new model we proposed, to solve the 3-Coloring problem. The time complexity analysis of our new algorithm is provided in Section 3, showing the algorithm's significant speedup over the existing DNA computing algorithms. Furthermore, how the algorithms can be improved to lower the error rate is discussed. Section 4 concludes this paper.

2. The fundamental new algorithm

Our new algorithms for the 3-Coloring problem are developed based on our new DNA computing model. The basic algorithm which will significantly reduce the computation time is introduced in this section. In Section 2, the algorithm will be advanced to solve larger size problems and to lower the error rate.

2.1. 3-Coloring problem

The 3-Coloring problem, a special case of the $k$-Coloring problem where $k = 3$, is a well-known representative of the class of NP-complete problems. NP-complete problems are those problems that no polynomial-time algorithm has yet been discovered. A new algorithm for solving the 3-Coloring problem will be introduced which will simplify the explanation of our new DNA computing model. The NP-complete class has the surprising property that if any one NP-complete problem can be solved in polynomial-time, then every problem in NP has a polynomial-time solution [8]. Hereby, the algorithms developed here can be expanded to solve the $k$-Coloring problem and be generalized to solve other NP-complete problems.

$k$-Coloring problem: A $k$-Coloring problem considers how to color an undirected graph $G = (V, E)$ in such a way that no two adjacent vertices share the same color [7]. Two nodes connected by an edge are referred to as adjacent vertices. The solution is a function $c: V \rightarrow 1, 2, \ldots, k$ such that $c(u) \neq c(v)$ for every edge $(u, v) \in E$. In other words, the numbers 1, 2, $\ldots$, $k$ represent the $k$ colors, and the adjacent vertices must have different colors. The $k$-Coloring problem determines whether $k$ colors are adequate to color a given graph [8].

A simple example graph with 10 nodes and 10 edges, $G(10,10)$, is given in Fig. 3. It is clearly shown that the graph can be colored if $k \geq 3$.

Some existing DNA computing algorithms for solving the 3-Coloring problem can be found in [2,3]. Basically, all these algorithms first generate a pool of encoded DNA strands representing all the possible color patterns of the $n$-node graph where each color pattern is an assignment of colors to nodes. For example, for nodes $n_1, n_2, n_3, n_4$, "BBRG" is one pattern which assigns Blue to $n_1$, Blue to $n_2$, Red to $n_3$ and Green to $n_4$, while "RGBB" is another pattern which colors $n_1, n_2$ as Red, Green, Blue and Blue, respectively. After the strands are generated and stored in a pool, the strands representing the color patterns with no color conflict need to be separated. Two nodes along an edge are defined as having color conflict when they are sharing the same color. For the color patterns with some color conflict existing along some edges of the graph, the correspond strand should be filtered out from the pool.

![Fig. 2. Separation operation based on gel layers.](image-url)
Our new algorithm is introduced next. Following that, the different variations of the algorithm and the advantages of the new algorithms will be described.

2.2. The new algorithm

Considering a given graph $G = (V, E)$, with $V = \{v_i\}_{i=1}^n$ being a set of nodes and $E = \{e_j\}_{j=1}^m$ being a set of edges, our approach to solving a 3-Coloring problem is divide-and-conquer. We first partition graph $G$ into two subgraphs: $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ such that $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ and $|V_1| \approx |V_2|$ by eliminating all edges $(u, v)$ such that $u \in V_1$ and $v \in V_2$. Henceforth, we will refer this set of edges as the cut-set of $G$, $C$ [6,7]. The partition process can be performed recursively. That is, subgraph $G_i$ can be partitioned into $G_{2i-1}$ and $G_{2i+2}$ until each subgraph contains only one vertex and $n$ subgraphs exist in total (see Fig. 4).

After partitioning the graph $G$ into $n$ subgraphs, the algorithm starts to merge every two subgraphs recursively and in parallel. Before the merge, every subgraph is colored with three colors. During the

**Recursive Dividing Step**

![Diagram](image)

Fig. 3. An example graph $G[10,10]$ that can be colored as Red ($R$), Green ($G$) and Blue ($B$).

![Diagram](image)

Fig. 4. Divide the graph, $G$, with $n = 2^p$ nodes until each subgraph only contains one node.
merge, the color patterns of the two subgraphs can be combined together if no new color conflict is caused. Note, to merge two subgraphs, the edges in the cut-set eliminated earlier in the partition of the two subgraphs will be added back and each addition of such edge will introduce a color conflict if the nodes it links are of the same color. Hence, the color patterns that worked for the subgraphs may not work for the merged graph after they are combined, and the combined color patterns may be eliminated. The merge operation continues until graph $G$ is re-established and the color patterns legitimate for it are found.

Our new algorithm for solving the 3-Coloring problem on a sparse graph is presented in Fig. 5. The first $\text{for loop}$ is used to generate $n$ pools of strands to represent all possible color patterns for $n$ subgraphs while initially each subgraph only contains one node.

The function of the $\text{while loop}$ is to, first, merge the pairs of two subgraphs. The bio-operation needed to merge the two subgraphs is $\text{ligation}$ which ligates strands in two pools to form longer strands. Let the color patterns for subgraph $G_1$ be $s_i$ and those for $G_2$ be $s_j$. For a given $s_i$, all the $s_j$s should be ligated with it, and such operations are performed over all the $s_i$s. That is, the strand for one color pattern of a subgraph is replicated and each duplicated copy is ligated with one of those strands representing the color patterns of the other subgraph. After the merge, all the color patterns of the merged graph will be represented by the ligated long strands.

After the merge, some ligated strands may encode the color patterns that have color conflicts introduced by those edges in the cut-set eliminated in the partition step. Our task is to investigate every edge in the cut-set and detect all the color conflicts caused hereby. This is accomplished by the separation operation, i.e. in all the ligated strands, to filter out strands that contain any color conflict from the pool. For each edge under investigation, two nodes, $i$ and $j$, are connected. We first separate the pool into three pools that contain the strands having node $i$ colored as $R$, $G$, and $B$. In these three pools, the strands having node $j$ colored as $R$, $G$, and $B$ are filtered out, respectively, by using the separation operation.

In the outer $\text{for loop}$, multiple copies of all the strands in all the pools need to be prepared for the next round of ligation. This duplication can be accomplished by using the polymerase chain reaction process [2,14].

If there is any strand left in the final pool, then the 3-Coloring problem has an answer: “yes”. Otherwise, the graph cannot be colored by only three colors and the answer is “no”.

3. The advanced new algorithms and their beauty

3.1. Speeding up the process

Consider a planar graph which is a graph with no two edges cross one another and is drawable on a plane. The size of the cut-set is $O(\sqrt{n})$ in such a graph [11]. Our new algorithm can solve the 3-Coloring problem of the planar graphs within $O(\log(n) + \sqrt{n})$ time. The first term, $O(\log(n))$, is the time needed to merge the subgraphs recursively in order to form the original graph $G$. The second term, $O(\sqrt{n})$, is the time needed to separate the strands representing the legitimate color patterns of the graph from the pool. It has been shown already that in this case, our DNA computing algorithm has a better time performance than the existing algorithm. In what follows, we present an advanced algorithm based on Algorithm 1 which
speeds up the process and improves the time performance. Given a dense graph, the number of edges can be the number of vertices squared. That means that the computation complexity of the existing DNA computing algorithms, $O(m + n)$, becomes $O(n^2)$.

Algorithm 2, the advanced DNA computing algorithm we propose is shown in Fig. 6. It is different from Algorithm 1 in that the color conflict is checked node by node rather than edge by edge.

For a dense graph the algorithm needs to be slightly modified, as shown in Fig. 6 in order to achieve the best performance. The only difference between this algorithm and Algorithm 1 is that instead of checking the color conflict edge by edge, the new algorithm checks the color conflict node by node. All of the strands that represent the color patterns with color conflicts between the node under investigation and all other nodes are isolated from the pool in a single step.

The implementation can be accomplished by using the device shown in Fig. 2 where the probes in the capture layer represent the colors of all the nodes connected to the node under investigation. All such nodes are checked simultaneously for color conflicts. The probes are different from what was introduced before, which previously was used to represent only the color of one node. For a graph with $n$ nodes, $n-1$ devices as given in Fig. 2 are necessary for $n-1$ separation operations. As shown in Algorithm 2, the time complexity of our new algorithm solving the 3-Coloring problem on a dense graph of $n$ nodes is $O((\log n) + n) = O(n)$, where the first term is the complexity of combining the subgraphs to regenerate the original graph, and the second term is the total complexity of checking all the $n$ nodes for color conflicts, one node at a time. Compared to the $O(n^2)$ time complexity of the existing DNA computing algorithms, the time performance of our new algorithm is a significant improvement.

For the 3-Coloring problems of some graphs, the solution may be reached even more quickly when a pool becomes empty in the middle of the process. It means that if three colors are not sufficient to color even a subgraph of a graph, they are certainly not capable to color the entire graph, obviously leading to the final answer of “no”. The last step of the algorithm can be easily performed by the detection operation listed in Section 2.2.

Algorithm 2

```
for i = 1 to n do
    In Parallel( f (P_i, color of node i) )
end

while f ≠ n do
    In Parallel( Make multiple copies of strands in all pools )
    In Parallel( S(P_i, P_j, P_k) )
    In Parallel( Relabel all pools i to 2 )
    for i = 1 to \frac{n}{2} do
        In Parallel( for j = 1 to N_i, N_i = # of nodes in subgraph i do
            S(P_i, P_j, P_k, w_i), w_i is color conflicts along all edges with endpoint j \forall j ∈ V_i
        end)
    end
    In Parallel( Make multiple copies of strands in all pools )
    f = \frac{n}{2}
end
```

Check if the pool is empty to return the “yes” or “no” accordingly.
3.2. Solving larger problems

The existing DNA computing algorithms for the 3-Coloring problems introduced in [2,15] have a solution space of $3^n$, the total number of color patterns for $n$ nodes with three colors, and requires $O(n + m)$ operations [3]. The size of the largest problem that can be solved by these existing algorithms is greatly restricted by the solution space. In the previously published results [2,3], the largest graph that can be solved for the 3-Coloring problem using the existing DNA computing technique is with 46 vertices because the total number of color patterns must be smaller than the number of strands used to represent them within a liter of water. That is

$$3^n < 10^{22}$$

$$n \leq \left\lfloor \log_3(10^{22}) \right\rfloor = 46$$

With our newly developed algorithm, this restriction is greatly loosen and much larger problems can be solved. The size of the largest problem that can be solved by our new algorithm is analyzed next.

The graph that can be colored with any number of colors is a graph with $n$ nodes and no edge. An example of such a graph with no edge is shown in Fig. 7. All the other graphs with $n$ nodes can be generated by adding edges to this disconnected graph.

Let $r$ be the proportion of strands that are retained in the pool after each separation based on the color conflict induced by adding one edge. From the graph with $n$ nodes and no edge, such as the one in Fig. 7, every time an edge is added in toward building the $n$-node, $m$-edge graph under consideration, some color patterns of $n$ nodes may need to be dropped due to the conflicts induced by this newly added edge. In other words, the color patterns that work for the graph without the edge may contain color patterns that have two nodes along the edge being colored the same. Such color patterns that color the two nodes with the same color will not work for the graph with the newly added edge. Based on a graph of $n$ nodes with no edge, in order to reach a graph with $m$ edges, $m$ edges must be added in. Among the strands that represent all $3^n$ different color patterns of the graph, a proportion $r$ of them can be kept after one separation operation is taken for each newly added edge, and $r^m$ of them can be kept in the final pool after $m$ separation operations are performed for $m$ newly added edges. This proportion of strands must be smaller than the total number of strands allowed to be involved in the computation, e.g. $10^{22}$ in 1 l of water. Thus,

$$10^{22} > 3^n \times r^m.$$  

Assume that the implementation checks the color conflict one edge at a time. Without losing generality, we can assume that all the adjacent edges sharing the same endpoint, node $\#1$, are processed first. Those edges with endpoint $\#2$, and so forth, are processed one by one. Suppose that the edge connected to node $\#i$ is under investigation and the other end is connected to node $\#j$, where $1 \leq i \leq n - 1$ and $j > i$. To color one node, let the sample space be $S: \{s_1, s_2, \ldots, s_k\}$ for $k$ colors. The probability $P(S) = 1$ and $P(s_1) = P(s_2) = \cdots = P(s_k) = 1/k$. For the 3-Coloring problem in which $k = 3$ and $S: \{R, G, B\}$, say, $P(R) = P(G) = P(B) = 1/3$. To color multiple nodes, let $c_j$ be the color of node $j$, and $P(c_j = s_i)$ be the probability that nodes $j$ is colored as $s_i$. The average proportion $r$ can be calculated based on the following independent cases that cover all the possibilities.

![Fig. 7. A disconnected graph with no edge.](image-url)
Under this condition, color patterns \{(G,R),(G,B),(B,R),(B,G)\} should be eliminated and those with \{(R,R),(G,G),(B,B)\} should be kept. The proportion of the strands that need to be separated is \(\frac{2}{3}\), which gives the proportion of (2) is
\[
\frac{2}{3} \times \frac{2}{3} = \frac{4}{9}
\]

The strands coding \((c_k,c_i,c_j)\) \(\in\{(R,R,G),(R,R,B),(G,G,B),(B,G,R),(B,G,B)\}\) should be eliminated due to edges \(e(k,i)\) and \(e(b,j)\). At this moment, color patterns only contain those with \(c_a \neq c_i\) and \(c_b \neq c_j\) because those with \(c_k = c_i\) or \(c_k = c_j\) are eliminated due to edges \(e(a,i)\) and \(e(b,j)\). The probability for strands to be kept is
\[
\frac{2}{3} \times \frac{2}{3} = \frac{4}{9}
\]
The color patterns

\[ P(c_i \neq c_j | (c_i \neq c_k) (c_j \neq c_l)) = \frac{P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l)}{P(c_i \neq c_k)(c_j \neq c_l)} \tag{3} \]

There are in total \( \binom{1}{3} \times \binom{1}{3} \times \binom{1}{3} \) possibilities of \( (c_i, c_j, c_k, c_l) \) where each node can choose a color from \( \{R, G, B\} \). In order to meet the criteria \( (c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) \), the color for node ‘a’ may be picked first which has \( \binom{1}{3} \) different possibilities. Then \( \binom{1}{3} \) different color options are left for node \( i \) due to \( c_a \neq c_i, \binom{1}{3} \) color selections are for node \( j \) where \( c_i \neq c_j \) and \( \binom{1}{3} \) are for node \( b \) to satisfy \( c_j \neq c_b \).

Therefore,

\[ P(c_i \neq c_j)(c_j \neq c_k)(c_i \neq c_l) = \binom{3}{1} \times \binom{3}{1} \times \binom{3}{1} = 8 \]

where the numerator is the number of possibilities of \( (c_i, c_j, c_k, c_l) \) that can meet the criteria \( (c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) \) and the denominator is the total possibilities of \( (c_i, c_j, c_k, c_l) \).

\[ P(c_i \neq c_j)(c_j \neq c_k) = P(c_i \neq c_j)|P(c_j \neq c_k) = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \]

Thus, the value of (3) is

\[ \frac{2}{3} \]

and the proportion of strands that should be kept is 2/3.

Cast 4: The last case is when there is at least one node \( k \) where \( k < i \), connecting to both nodes \( i \) and \( j \), like the one shown in Fig. 8(c). The color patterns \( \{n_1 n_2 \ldots n_i \ldots n_j \ldots n_k \ldots X\} = \{XX \ldots c_i \ldots c_j \ldots c_k \ldots X\} \) with \( c_i = c_f \) should be separated. At this moment, strands representing color patterns with \( c_k = c_i \) and \( c_k = c_j \) have already been separated due to edges \( e(k, i) \) and \( e(k, j) \).

Because color patterns \( \{n_1 n_2 \ldots n_k \ldots n_j \ldots n_k \ldots X\} = \{XX \ldots c_i \ldots c_j \ldots c_k \ldots X\} \) with \( (c_i, c_j, c_k) \in \{(R, G, B), (R, B, G), (G, R, B), (G, B, R), (R, R, B), (R, B, R), (B, B, R), (B, B, G), (B, G, R), (G, G, R), (G, G, B), (B, G, G), (B, G, B), (G, B, G), (G, G, G)\} \) should be kept, and those with \( (c_i, c_j, c_k) \in \{(R, R, G), (R, R, B), (G, G, R), (G, G, B), (B, G, G), (B, G, B), (G, B, G), (G, G, G)\} \) should be separated, the proportion \( r \) is:

\[ \frac{P(c_i \neq c_j)(c_j \neq c_k)(c_i \neq c_l)}{P(c_i \neq c_k)(c_j \neq c_l)} = \frac{P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l)}{P(c_i \neq c_k)(c_j \neq c_l)} \tag{4} \]

Because

\[ P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) = P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) \]

and

\[ P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) = P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) \]

an obvious conclusion is

\[ P(c_i \neq c_j)(c_i \neq c_k)(c_j \neq c_l) = \frac{1}{4} \]

Hence, the value of (4) is:

\[ \frac{2}{3} \times \frac{1}{4} = \frac{1}{2} \]

when nodes \( i \) and \( j \) are connected to one common node \( k \) where \( k < i \).

We can define a tight boundary for keeping the strands in the pool at the time an edge is under consideration. After each edge is considered, at least one-third of the strands are separated from the current pool. At most two-thirds of the strands are retained in the pool to represent color patterns of the graph with no color conflict.

For example, for a planar graph with 256 vertices, in order to generate the answer for the 3-Coloring problem with \( 10^{12} \) strands, the number of edges must be more than 569. This is because

\[ 10^{12} > 3^{256} \times 2^m \Rightarrow m > 569 \]

Because 569/256 \( \approx 2.2 \) is the average minimum number of degrees required for each node to have the 3-Coloring problem solved by 11 of strands. Meanwhile, 256 vertices are significantly more than the
number of vertices in the graph that are solvable by the existing 3-Coloring DNA computing algorithms, within the same given solution space. If an electronic computer that can perform $10^9$ operations per second is used, then $10^{19}$ years will be required to solve the 3-Coloring problem for the graphs with 256 vertices even if the fast Biegel and Eppstein’s algorithm [3] is used. By introducing our new model, it will take approximately 2 days to finish the implementation of our new algorithm, assuming that the average DNA operation takes 20 min [2,3]. Our new algorithm can be used to solve the 3-Coloring problems for graphs containing significantly higher number of vertices and, as compared to the existing DNA computing algorithms or the algorithms designed for electronic computers, is also significantly faster.

### 3.3. Error resistance

At the time that DNA computing was introduced, a question was raised about how errors may affect the computing results. Although mature biological operations have very low error rates, errors may still accumulate and thus generate incorrect answers. An introduction to our new algorithm with error resistance follows.

Most of the errors in DNA computing occur during the separation operation. After each separation operation, one pool is separated into two pools. Let the pool containing all the strands coding the color patterns that may be able to color the graph be defined as the positive pool, $P_t$. The pool containing all the strands representing the color patterns with no color conflict kept in the graph coloring problem, at the time that the final pool is generated, strands will be decoded from the pool. False positive errors are easy to handle because in the graph coloring problem, at the time that the final pool is generated, strands will be decoded from the pool. Unwanted color patterns will be quickly dropped after checking whether they work for the graph by using electronic computers. On the other hand, false negative errors are more difficult to detect and they are usually more expensive to correct.

Presented below is our new and improved DNA computing algorithm for solving the 3-Coloring problem that can reduce the false negative error to a minute rate, $\epsilon$. Assume that each separation operation has an average false negative rate of $q$, that is

$$q = \frac{a}{a + \beta}$$

where $a$ is supposed to be the number of strands representing the color patterns with no color conflict left in the negative pool, and $\beta$ is the number of strands representing the color patterns with no color conflict left in the positive pool. The proportion of strands that represent the color patterns with no color conflict kept successfully in the positive pool is $p$.

$$p = \frac{\beta}{a + \beta}$$

where

$$p + q = 1.$$

The most straightforward method of reducing the false negative error rate is to repeat the same process a number of times. Suppose that $d$ times are repeated and the false negative error which results is $E$, when $E = d^\epsilon$. To assure that $E \leq \epsilon$, we have $d^\epsilon \leq \epsilon$ and

$$d = \left\lceil \frac{\log\epsilon}{\log q} \right\rceil.$$  

That is, after repeating the separation operation $\left\lceil \frac{\log\epsilon}{\log q} \right\rceil$ times, the false negative error rate will be smaller than $\epsilon$. However, this method is not only inefficient but may also possibly increase the false positive, i.e. leading strands with color conflicts into the positive pool.

Our previously described new algorithm can be advanced to reduced the false negative error rate. As we discussed, the negative pool, $P_f$, may have some strands that represent color patterns without color conflicts. Instead of being discarded, the $P_f$ pool will be further processed by the next operation in order to retain those strands that may represent the final answer to the problem. After the second separation operation, $P_f$ will be divided into the positive pool, $P_h$, and the negative pool, $P_j$. Pool $P_l$ is separated into the positive pool, $P_h$, and the negative pool, $P_j$, where $P_h$ contains all the strands currently representing color patterns with no color conflict. $P_j$ contains the strands
that represent those color patterns with color conflicts along the new edge under consideration. Pools $P_t$, $P_{t-1}$, and $P_{t-2}$ will be combined together and labeled as $P_t$. $P_{t-1}$ is relabeled as $P_{t-1}$ and $P_{t-2}$ is labeled as $P_{t-2}$. After subsequent separation operations at different levels, the corresponding processes have been listed in Fig. 9. Pool $P_t$ has all the strands representing color patterns having color conflict along at least one edge, while pool $P_{t-1}$ contains those having conflicts along at least two different edges. $P_t$ contains the strands that represent those color patterns capable of coloring the graph. The possibility that pool $P_t$ has strands that should be in $P_1$ is $q^2$. The same operations should continue until $d+1$ different pools, which are $P_1, P_2, \ldots, P_d$ and $P_t$ are generated where $q^2 \leq \epsilon$. The false negative rate for strands left in pool $P_t$ that represent color patterns with no color conflict is smaller than $\epsilon$. The extra expense required to achieve this low error rate with our new algorithm is very small. With a 1% false negative error rate in a single separation operation, it is very easy to reduce the overall false negative rate to 0.0001%, with $d$ as small as 3.

4. Conclusion

In this paper, a new model for DNA computing is introduced. Based on the new model, our new algorithms for the 3-Coloring problems have been presented. The new algorithms have the advantage of faster speed as compared to the existing DNA computing algorithms. The new algorithms also represent a significant speed improvement over the existing algorithms.

Our algorithms are obtained by parallelizing the separation operation on multiple edges and by parallelizing other DNA computing processes. This has provided the opportunity to solve very large problems which cannot currently be solved by electronic computers in reasonable amount of time. The solution space of $10^{22}$ strands in a 11 pool can be efficiently used. With the given solution space, problems of a large size that cannot be solved using the existing DNA computing techniques are now solvable. The introduction of our new algorithms makes DNA computing a very attractive option to potential users who want to solve problems that are currently unsolvable.

Finally, our new algorithm for error resistance is also presented. The DNA computing technique has been greatly improved by reducing the error rate to a very small percentage. This improvement will make DNA computing significantly more reliable.

Based on the new model, other algorithms can be developed to solve different NP-complete problems and those problems that are computationally intense. All the algorithms that will be developed based on our new model have the advantages described above. This new model is able to expedite the development of DNA computing techniques.
References