A Polynomial-Time DNA Computing Solution for the N-Queens Problem

Ramin Maazallah*, Aliakbar Niknafs, Paria Arabkhedri

Department of Computer Engineering, Shahid Bahonar University of Kerman, Kerman, Iran

Abstract

The N-queens problem is a classic combinatorial problem that there is no polynomial time solution for it in silicon based computers. It belongs to the set of NP-Complete problems and needs a plenty of calculations. On the other hand, it has been evidenced that DNA computing is able to solve such complex problems efficiently. In this paper we propose a method based on Adleman-Lipton model, a model of DNA computing, which is able to solve the N-queens problem in a polynomial time complexity. It provides all the solutions and runs in O(N^2).

Keywords: DNA computing, Adleman-Lipton Model, N-queens problem, NP-Complete;

1. Introduction

DeoxyriboNucleic Acid (DNA) computing is a novel method of computing that uses DNA molecules for computation. It was initially developed by Leonard Adleman in 1994. Adleman (Adleman, 1994) explained an experiment for solving the Hamiltonian Path Problem (HPP) in a polynomial time using DNA molecules. The HPP is an NP-Complete problem and there is no polynomial time solution for it in silicon based computers. Adleman’s work showed the efficiency of using DNA molecules in solving such complex problems. Following Adleman’s success, polynomial time algorithms for other NP-Complete problems such as graph coloring problems (Xu, et al., 2011), (Yeh & Wu, 2009), SATisfiability (SAT) problem (Braich, Chelyapov, Johnson, Rothemund, & Adleman, 2002), maximal clique problem (Ouyang, Kaplan, Liu, & Libchaber, 1997), knapsack problem (Darehmiraki & Nehi, 2007), (Henkel, Back, Kok, Rozenberg, & Spaink, 2007) and bin packing problem (Alonso Sanches & Soma, 2009) has been proposed. In fact the main reason of using DNA molecules for computation is their vast parallelism power. This feature of DNA molecules let us try many different possibilities at once. So the computation is carried out in parallel. Higher information density and lower consumption of energy per operation are two more benefits of DNA computing (Adleman, 1994).

The N-queens problem is the general form of eight-queens puzzle that looks for the ways of placing N queens on an N*N board such that no two queens attack each other. This problem can be quite computationally expensive

* Corresponding author. Ramin Maazallah Tel: +13 3243415
E-mail address: ramin.mz66@gmail.com
because there are a lot of possible arrangements for $N$ queens. Brute-force algorithms try to find all the solutions by checking all these arrangements, but this is unpractical for large values of $N$. Thus several different approaches have been proposed in literatures to solve the $N$-queens problem without enumeration. Some of these are as follows: using of backtracking search (Bitner & Reingold, 1975), local search and conflict minimization techniques (Sosic & Gu, Efficient local search with conflict minimization: A case study of the nqueens problem, 1994), neural networks (Onomi, Maenami, & Nakajima, 2011), (Bharitkar & Mendel, 2000), search heuristic methods (Kale, 1990), (Martinjak & Golub, 2007), probabilistic local search algorithms (Sosic & Gu, Fast search algorithms for the n-queens problem, 1991) and integer programming (Foulds & Johnston, 1984).

In this paper we propose a method based on Adleman-Lipton model (Adleman, 1994), (Lipton, 1995), which is able to solve the $N$-queens problem in a polynomial time complexity. It provides all the solutions to this problem and runs in $O(N^2)$. This paper is organized as follows: Section 2 briefly introduces DNA computing concepts and the Adleman-Lipton model. Section 3 describes the $N$-queens problem. Our proposed method is given in Section 4 and is evaluated in Section 5. Finally Section 6 concludes the paper.

2. DNA Computing

DNA is a polymer composed of monomers called nucleotides. Each nucleotide contains three components: a sugar, a phosphate group, and a base. Figure 1 illustrates the structure of a nucleotide. The sugar has five carbon atoms which are numbered from 1’ to 5’. The Phosphate group is attached to the 5’ carbon and the base is attached to the 1’ carbon. There are four different types of bases: Adenine, Guanine, Cytosine and Thymine, abbreviated as A, G, C and T, respectively. What makes a nucleotide distinct from another is the base portion. So we can refer to every nucleotide as A, G, C or T nucleotides, depending on the type of base they have. Therefore it is possible to consider a DNA strand as a string over the alphabet \{A, G, C, T\}. For example ATTGCATGG is a DNA strand composed of 9 nucleotides.

In every DNA computing experiment, there is a (test) tube containing lots of DNA molecules (strands). Each of the DNA molecules can be a potential solution to the problem. Also there are some biological operations available, which are performed on the DNA strands of the tube to perform computations. These operations differ according to the DNA computing model used.

![Figure 1. Nucleotide structure. Each nucleotide is composed of a sugar, a phosphate group and a base.](image)

The Adleman-Lipton is a DNA computing model constructed upon the operations proposed by Adleman and Lipton in their papers (Adleman, 1994), (Lipton, 1995). The most important operations of this model that we have used are as follows:

- **Extract**\((T, S, T^+, T^-)\): Given a tube $T$ and a string $S$, this operation produces two tubes $T^+$ and $T^-$ as follows: All the DNA strands of $T$ having $S$ in their sequence are placed in $T^+$ and the remained DNA strands are placed in $T^-$.  

- **Copy**\((T, T_1, T_2, ..., T_n)\): Given a tube $T$, this operation creates tubes $T_1$ to $T_n$ which are identical copies of $T$. ($T_1$ to $T_n$ will have the same DNA strands as $T$.)

- **Merge**\((T, T_1, T_2, ..., T_n)\): Given tubes $T_1$ to $T_n$, this operation pours the contents of $T_1$ to $T_n$ into tube $T$, so the tube $T$ contains all the DNA strands of $T_1$ to $T_n$.

- **Append**\((T, S)\): Given a tube $T$ and a string $S$, this operation appends $S$ to the end of all the DNA strands of $T$.  


3. The N-Queens Problem

The eight-queens problem is a classical puzzle that was originally proposed by the chess player Max Bezzel. The problem is to place eight queens on an 8*8 chessboard such that no two queens threaten each other. Therefore no two queens can be placed on the same row, column or diagonal. The eight-queens problem is a special case of the generalized N-queens problem for N=8. The N-queens problem is defined as placing N queens on an N*N board having these constraints:

F There must be exactly one queen on each column.
F There must be exactly one queen on each row.
F No more than one queen can be placed on any diagonal.

A solution to this problem can be represented as a permutation of N rows, because there must be exactly one queen on each column. So we can find solutions to the N-queens problem by enumerating through all N^N permutations, but this process takes a long time for large values of N, even on the fastest silicon based computers.

4. Proposed Method

In this section we present our method named DNA-NQUEENS which is able to find all the solutions to the N-queens problem in a polynomial time. Figure 2 illustrates the structure of an N*N sample board. The board has N rows, represented by R1 to RN, (2N-1) main diagonals, labeled M1 to M2N-1, and (2N-1) secondary diagonals, S1 to S2N-1. As it was noted in Section 3, the goal is to place N queens on the board such a way that no two queens threaten each other. Let Qi,j denote the queen at ith row and jth column, so there are totally N^2 number of different queens.

To solve the problem by DNA computing, we must encode each of the queens as a specific DNA strand, constructed as follows: Let Ri be a DNA strand encoding ith row for i=1,2,…,N, and Mj a DNA strand encoding jth main diagonal for j=1,2,…,2N-1 and finally Sk a DNA strand which encodes kth secondary diagonal of the board for k=1,2,…,2N-1. For each of the queens, Qi,j, i=1,2,…,N, j=1,2,…,N, we construct a DNA strand that its string is the concatenation of the string of these DNA strands: Rr,i,j, Mr,i,j and Ss,i,j in which r,i,j, m,i,j and s,i,j are the row, main diagonal and secondary diagonal of Qi,j which are calculated as:

\[ r_{i,j} = i, \]
\[ m_{i,j} = j + |i - N|, \]
\[ s_{i,j} = i + j - 1. \]
Figure 3 shows matrix $Q$ that encodes all $N^2$ number of queens.

$$
\begin{bmatrix}
R_1M_{N}S_1 & R_1M_{N+1}S_2 & \ldots & R_1M_{2N-1}S_N \\
R_2M_{N-1}S_2 & R_2M_{N}S_3 & \ldots & R_2M_{2N-2}S_{N+1} \\
\vdots & \vdots & \ddots & \vdots \\
R_{N}M_{1}S_N & R_{N}M_{2}S_{N+1} & \ldots & R_{N}M_{N}S_{2N-1}
\end{bmatrix}
$$

Figure 3. Matrix $Q$ which encodes all $N^2$ number of queens as a specific DNA strand. $Q_{i,j}$ encodes the queen placed on $i$th row and $j$th column.

Our proposed method has two major steps. First we create all permutations of $N$ queens on the board such that there is exactly one queen on each column (totally $N!$ permutations exist). Second we extract solutions among all these permutations. A permutation is said to be a solution, if it satisfies these three constraints:

1. There is exactly one queen on each row.
2. The $N$ queens are placed on $N$ distinct major diagonals.
3. The $N$ queens are placed on $N$ distinct secondary diagonals.

The pseudocode of our proposed method ($\text{DNA-NQUEENS}$) is shown in Figure 4. It’s output is a tube named $T_0$ containing all the solutions to the problem. The first procedure which is executed is $\text{GeneratePermutations}$. It generates all possible arrangements of $N$ queens on the board. Figure 5 shows the pseudocode of this procedure. The $\text{Init} (T_0)$ command creates an empty tube named $T_0$. After execution of $\text{GeneratePermutations}$ procedure, the tube $T_0$ contains DNA strands of this form: $Q_{r_1}Q_{r_2}Q_{r_3}\ldots Q_{r_N}$, $r_i \in \{1,2,\ldots,N\}$, $i = 1,2,\ldots,N$. Each DNA strand of $T_0$ encodes a way of putting $N$ queens on the board but not all these strands encode a solution. For example, the DNA strand $R_1M_4S_1R_2M_4S_3R_4M_3S_6R_2M_6S_5$ encodes the arrangement of four queens shown in Figure 6 but is not a solution to the four-queens problem because it violates constraint 1 and 3. ($Q_{2,2}$ and $Q_{2,4}$ are placed on the same row and also the queens are on three distinct major diagonals rather than four.)

![Figure 4. DNA-NQUEENS procedure.](image-url)

![Figure 5. GeneratePermutations procedure.](image-url)
Figure 6. A permutation of four queens encoded by the DNA strand: \(R_1M_4S_2R_4M_2S_5R_3S_4M_3S_3\). This permutation is not a solution to the four-queens problem.

The **CheckRows** procedure gets a tube of DNA strands and checks to see which of them satisfies constraint 1 and extracts them. The pseudocode of this procedure is presented in Figure 7.

The **CheckMainDiagonals** procedure which is shown in Figure 8, checks the second constraint of solutions. It extracts all the DNA strands which have the queens on \(N\) distinct major diagonals. To accomplish this, one more DNA strand is required. This strand is represented by \(M\) and is appended to the end of the DNA strands as follows: First the DNA strands having one or more queens placed on the first major diagonal are extracted and are appended one \(M\). The same process is repeated for all other major diagonals. The strands having the queens on \(N\) distinct major diagonals, are appended the strand \(M\), exactly \(N\) times. To extract these strands a new DNA strand is constructed by repeating DNA strand \(M\), exactly \(N\) times. We refer to this strand as \(M^N\) (See Figure 9).

Finally the **CheckSecondaryDiagonals** procedure checks the last constraint of solutions. It extracts all the DNA strands having \(N\) queens on \(N\) distinct secondary diagonals. Like in **CheckMainDiagonals** procedure, we need one more DNA strand represented by \(S\). Also \(S^N\) is the DNA strand created by repeating the strand \(S\), exactly \(N\) times. The pseudocode of this procedure is illustrated in Figure 10.
Figure 9. The DNA strand $M^N$. This strand is used for extracting the permutations having $N$ queens on $N$ distinct major diagonals.

```
PROCEDURE CheckSecondaryDiagonals
INPUT: Tube T₀
OUTPUT: Tube T₀
BEGIN
FOR i=1 to 2N-1
    EXTRACT(T₀, S_i, T₁, T₀)
    Append(T₁, S)
    Merge(T₀, T₀, T₁)
END FOR
Extract(T₀, S₀, T₀, T₁)
RETURN T₀
END
```

Figure 10. CheckSecondaryDiagonals procedure.

After execution of the DNA-NQUEENS procedure, each of the DNA strands remained in $T₀$ encodes a solution to the $N$-queens problem. We have simulated our method on silicon based computers. For example running the DNA-NQUEENS for $N=4$, yields the following DNA strands:

$R_2M_3S_2R_4M_2S_5R_1M_6S_3R_3M_5S_6MMMSSSS$

$R_3M_2S_3R_1M_5S_2R_4M_3S_6R_2M_6S_5MMMSSSS$

These strands encode two solutions of the four-queens problem as shown in Figure 11.

![Figure 11. Two solutions of the four-queens problem obtained by running DNA-NQUEENS for N=4. These solutions are encoded by these DNA strands:](image)

$R_2M_3S_2R_4M_2S_5R_1M_6S_3R_3M_5S_6MMMSSSS$

$R_3M_2S_3R_1M_5S_2R_4M_3S_6R_2M_6S_5MMMSSSS$

5. Evaluation

One measure for evaluation of an algorithm is the time complexity. It specifies the execution time of the algorithm as a function of the size of the input to the problem. The time complexity of an algorithm is calculated by counting the number of elementary operations of the algorithm and is expressed using “big O” notation. The time
complexity for each procedure of our proposed method is given in Table 1. Note that in calculation of these time complexities, the number of execution of DNA operations (Append, Merge, Extract and Copy) is counted.

Table 1. The time complexity for each procedure of the proposed method

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneratePermutations</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>CheckRows</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>CheckMainDiagonals</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>CheckSecondaryDiagonals</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>

Therefore the time complexity of our proposed method is $O(N^2)$, meaning it gives polynomial time solutions to the $N$-queens problem. Note that there is no polynomial time solution to this problem using silicon based computers.

6. Conclusion

The vast parallelism power of DNA molecules makes this possible to create all potential solutions to a problem and then extract feasible ones with the help of biological operations available. In this paper we propose a method based on Adleman-Lipton DNA computing model which finds all the solutions to the $N$-queens problem. The time complexity of our proposed method is $O(N^2)$. This paper shows the efficiency of DNA computing in solving NP-Complete problems which are quite computationally expensive using silicon based computers.

References