International Journal of Computer Science and Applications,
Vol. 5, No. 2, pp 12 - 22

A DNA STICKER ALGORITHM FOR SOLVING N-QUEEN PROBLEM *

H. AHRABIAN

Center of Excellence in Biomathematics, School of Mathematics, statistics and computer Science, University of Tehran, Tehran, Iran
ahrabian@ut.ac.ir

A. MIRZAEI

Department of Computer Engineering,
University of Amirkabir, Tehran, Iran

A. NOWZARI-DALINI†

Center of Excellence in Biomathematics, School of Mathematics, statistics and computer Science, University of Tehran, Tehran, Iran
nowzari@ut.ac.ir

Over the past few decades numerous attempts have been made to solve combinatorial optimization problems that are NP-complete or NP-hard. It has been evidenced that DNA computing can solve those problems which are currently intractable on even fastest electronic computers. This paper proposes a new DNA algorithm for solving N-Queen problem, a complex optimization problem. The algorithm not only shows whether or not a solution exists, but provides all possible solutions by massively parallel computations. The proposed algorithm can be easily extended to solve other optimization problems.

Keywords: Combinatorial optimization; N-Queen problem; Molecular computing; Sticker model.

1. Introduction

In 1994, Adleman [Adleman (1994)] described a laboratory experiment involving DNA in which an example of Directed Hamiltonian Path Problem was solved in a polynomial time. His paper opened the field of practical DNA computing. Later, Lipton [Lipton (1995)] demonstrated that a large class of NP-complete problems also could be solved in a polynomial time by encoding the problem in DNA molecules. Shortly after, DNA computing has been used in solving many other problems such as finding maximal clique [Ouyang *et al.* (1997)], satisfiability (SAT) problem [Braich *et al.* (2002); Lipton (1995)], calculation of multiplication of Boolean matrix [Oliver (1999)], breaking DES

* This research was partially supported by University of Tehran.

† To whom correspondence should be addressed.

codes [Boneh *et al.* (1996); Leier *et al.* (2000)], chess problem [Faulhammer *et al.* (2000)], simulation of Boolean circuits [Ahrabian *et al.* (2005); Ahrabian and Nowzari (2004); Ogihara and Ray (1999)], arithmetic and logic operation [Barua and Misra (2002); Frisco (2000); Gupta *et al.* (1999)], simulating Turing machine and automata [Beaver (1995); Benenson *et al.* (2003); Gao *et al.* (1999); Rothemund (1996)], and molecular expert system [Wasiewicz *et al.* (2000)].

DNA molecules are composed of single or double DNA fragments or often called oligonucleotids or strands or oligo in short. A single stranded fragment has a phospho-sugar backbone and four kinds of bases denoted by the symbols A, T, G, and C for the bases Adenine, Thymine, Guanine, and Cytosine respectively. These four nucleic acids, which can occur in any order, combined in the Watson-Crick complementary pairs to form the double stranded duplex of DNA. Due to hybridization reaction, A is complementary with T and C is complementary with G and vice versa. As an example, any oligu, such as 5'-ACCTG-3' has the complementary sequence 3'-TGGAC-5'. Digits 5' and 3' denote orientation of oligu.

There are a number of feasible operations in DNA computing. Based on the utilized operations, several DNA computer based models for computation have been studied. Among them Adleman-Lipton model [Adleman (1994); Gupta *et al.* (1999)], surface-based model [Liu *et al.* (2000); Su and Smith (2004); Wang *et al.* (1999)], sticker-based model [Kari *et al.* (1998); Ouyang *et al.* (1997); Paun and Rozenberg (1998); Roweis *et al.* (1998)], splicing systems [Ferretti *et al.* (2000); Freund *et al.* (1999); Mateescu *et al.* (1998)], and self-assembly model [Winfree (2003); Winfree *et al.* (1998)] are becoming highly standardized. Comparing these models, it appears to be easier writing programs for the sticker model [Zimmermann (2002)].

Computing with DNA molecules has many advantages over conventional computing methods. Though DNA computing performs individual operations slowly, it can execute billions of operations simultaneously. This is contrasted with electronic digital computers where individual operations are very fast; however, the operations are executed basically sequentially. The massive parallelism of DNA computing comes from the huge number of molecules which chemically interact in a small volume. Based on the massive parallelism, many researchers tried to solve variety of difficult problems.

The N-Queen problem as a well-known combinatorial optimization problem is NP-Complete and no efficient solution algorithm for this type of problem has yet been found [Yaglom and Yaglom (1964)]. The N-Queen problem consists in placing N queens in the chess board such that no one offends the others. The number of possible arrangements of N queens in the board is extensively high. Thus, an enumeration method is not realistic. Traditional approaches to the N-Queen problems are based on backtracking. Backtracking search techniques can systematically generate all possible solutions, but backtracking searching is exponential in time and is not able to solve a large-scale N-Queen problem. Therefore, lots of new searching methods are proposed. Earlier attempt was by Crawford [Crawford (1992)]. He gave an evolutionary search technique based on genetic algorithm. An efficient local search algorithm for this problem is given in [Sosic and Gu (1994)]. They claim that their algorithm works faster than any backtracking

search algorithm. Also, this problem has been solved with a variety of artificial neural networks [Mandziuk (2002); Ohta (2002); Smith (1999); Xu *et al.* (2005)]. Ohta [Ohta (2002)] introduced the chaotic neural networks with reinforced self-feedback for escaping from the local minimums during the search, and applied it to N-Queen problem. Later in [Xu *et al.* (2005)], a method for improving the transiently chaotic neural network was introduced. This work enables the neural network to have rich search ability initially and to use less CPU time to reach a stable state. A multi-agent evolutionary algorithm for N-Queen problem proposed by Zhong *et al.* [Zhong *et al.* (2005)]. Their results show that their method even for large number of N can find the exact solution very fast.

This paper proposes the use of massive parallelism inherent in DNA computation to solve this problem. To do so, we introduced a general framework to code the constraints of the problem and using this framework, in a large set of possible solutions, the DNA strands which satisfy the constraints of the problem are extracted. These strands are the solution of the problem. It is worth mentioning that many constraints satisfaction problem such as quadratic assignment, maintenance scheduling, resource allocation, and the graph matching problems can be solved in a similar manner.

This paper is organized as follows: Section 2 provides a brief technical introduction to the Sticker model as stated by Zimmermann [Zimmermann (2002)]. In Section 3 we introduce the well known combinatorial optimization application, the N-Queen problem and Section 4 describe how DNA computing can be used to solve it. The conclusion is presented in Section 5.

2. A Model of DNA Computation

The sticker model is a model of molecular computation introduced in [Roweis *et al.* (1998)]. The memory of the sticker model consists of memory complexes. A memory complex is a DNA strand that is partially double and can be viewed an encoding of an *n*-bit binary number. Each memory complex is formed by two basic types of single stranded molecules referred to as memory strands and sticker strands. A memory strand is a single stranded DNA molecule containing *n* none overlapping substrands. Sticker strand is a single-stranded DNA molecule which is complementary to exactly one of the *n* substrands in the memory strand. If a sticker strand is annealed to its matched substrand on a memory strand, the particular substrand is *on*; otherwise, it is *off*.

In sticker model a test tube is a multi-set of memory complexes in which several copies of the same strand may be contained in it. The operators of the sticker model are [Roweis *et al.* (1998); Zimmermann (2002)]:

- **Merge (T_1, T_2, T)**: In this operation, the memory complexes of two test tubes T_1 and T_2 are combined into the test tube T and is returned.
- **Separate (T, *i*, T_1, T_2)**: The operation *separate* produces for a test tube T and an integer *i*, two new test tubes, the test tube T_1 and the test tube T_2. The test tube T_1 consists of all memory complexes of test tube T in which the *i*th substrand is *on*, and the test tube T_2 consists of all memory complexes of test tube T in which the *i*th substrand is *off*.

- **Set (T, *i*)**: The operation turns *on* the *i*th substrand of each memory complex of test tube T.
- **Clear (T, *i*)**: The operation turns *off* the *i*th substrand of each memory complex of test tube T.

The input of a sticker algorithm is a test tube (*initial test tub*). This test tube contains memory strands that called library. In general an *[n,k]* library is a commonly proposed input test tube that consists of memory complexes with *n* substrands such that the first *k* of which are turned *on* or *off* and the last *n-k* of which are turned *off* [Roweis *et al.* (1998)]. Zimmermann [Zimmermann (2002)] proposed a different kind of input test tubes, $\left[m, \binom{n}{k}\right]$ libraries. Such a library consists of $\binom{n}{k}$ memory complexes each of which having *m* substrands of the form $w0^{m-n}$, where *w* is a binary word of length *n* with *k* substrands turned *on* and *n-k* substrands turned *off*. So this library provides an encoding of all the subsets of *k* elements (*k*-subsets) of an *n*-set.

Computation through sticker model is carried out using a finite sequence of the operation merge, separate, set, and clear on an input library.

The output of a sticker computation is a sequence of test tubes called *final test tubes*. The output of a final test tube is read or it is reported that it contains no memory complexes.

3. The N-Queen Problem

The eight queens is a well known NP-complete problem proposed by C. F. Gaus in 1850 [Wirth (1976)]. The Problem was investigated by several 19-th century mathematicians. The characteristic property of this problem is that it requires large amount of computations.

The general N-Queen problem was explored in 1950's by Yaglom and Yaglom [Yaglom and Yaglom (1964)]. A general N-Queen problem is defined by the following constraints on an *N*N* grid:

1. Only one queen can be placed in any row.
2. Only one queen can be placed in any column.
3. Only one queen can be placed on any diagonal.
4. Exactly *N* queens must be placed on the grid.

There have been several approaches taken in the study of this problem (as diverse as algorithmic design, program development, parallel and distributed computing, and artificial intelligence). This widespread interest in the N-Queen problem is in part due to the property that characterizes difficult problems, viz., satisfying a set of global constraints. In the next section we introduce our molecular algorithm for solving this problem.

4. DNA sticker algorithm for N-Queen problem

The first step of any molecular algorithm is introducing an encoding system which specifies how the problem is encoded in DNA strings. For this purpose, any arrangement of the problem with N queens in the grid $m=N*N$ is encoded as a DNA sequence of length $m+N+2(2N-1)$. In other words each is considered as a memory complex of size $m+N+2(2N-1)$ bits. The first m bits of each memory complex encode the grid $m=N*N$, and the kth bit, $k=(i-1)*N+j$ for $1 \le k \le N^2$, in this part is set *on* if the queen is located in the ith row and jth column. The next N bits are used for recording the column position of each queen, and the other two consecutive $2N-1$ bits are used for recording the main and the secondary diagonal positions of the queens. Figure 1 shows the structure of the designed DNA strand. In this figure the part "a" encodes the presence or absence of a queen in the corresponding position. Parts "b" to "d" can be used for counting the number of queens in the columns, and in the main and secondary diagonals respectively. Clearly, as it is illustrated in the Figure 2, by knowing the value of k, then i and j (row and column position) and even the diagonal positions $d1$ and $d2$ of the queen can be easily evaluated mathematically as the function **Findpos** given in Figure 3.

Therefore, the initial input of the algorithm, is a library $\left[m+N+2(2N-1), \binom{m}{N} \right]$ which is assigned in the initial test tube T_0. This library contains $\binom{m}{N}$ strands of length $m+N+2(2N-1)$ that are amplified in a fairly enormous rate. Note that for feasibility of DNA operations in the test tube, we should have many copies of any strand. For this reason, before the execution of any DNA operation, the strands in the test tubes are amplified in fairly enormous rate (nearly 100000 copies). As it is defined, all the arrangements of queens are considered by different combination of $0-1$ of the first m bits of the strands in this library. Thus our molecular algorithm requires initially all possible arrangements of N queens in the grid encoded as DNA sequences.

With regard to the above encoding the molecular solution of the N-Queen problem is given below. In the first step we extract memory complexes that have exactly one queen placed in each row. The procedure **Count** given in Figure 4 is designed for this purpose. A test tube T_0, an index L, and an integer n are the three parameters of this procedure. The procedure **Count** selects all the strands in the T_0 that have exactly one single *on* bit in the positions from $L+1$ to $L+n$ of the strand. In this procedure the test tubes $T_{\#=0}$, $T_{\#=1}$, and $T_{\#>1}$ contain memory strands with no *on* substrand, just one *on* substrand and more than one *on* substrands respectively.

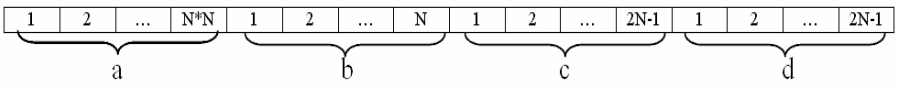

Fig. 1. The DNA strand for encoding the problem.

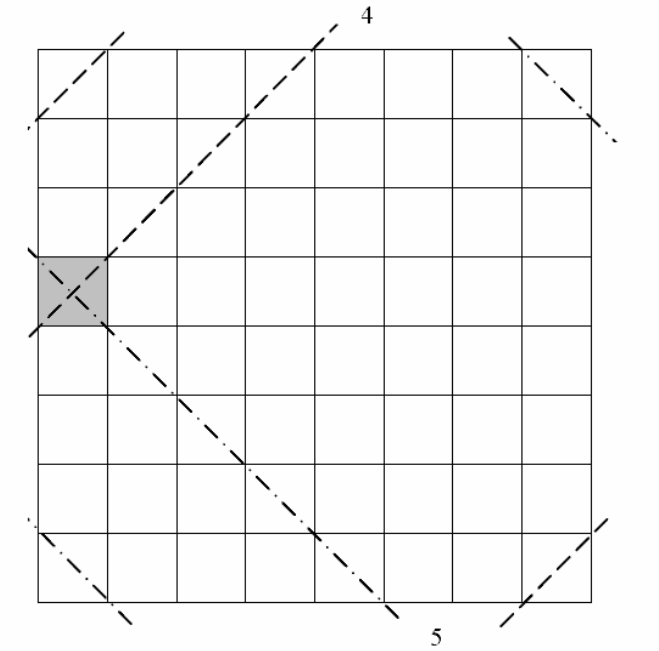

Fig. 2. The shaded cell *k=25* is in the row *i=4*, column *j=1*, main diagonal *d1=5* (right-skewed line), and secondary diagonal *d2=4* (left-skewed line).

1. Findpos (*k, i, j, d1, d2*)
2. Begin
3. $i = \left\lfloor \frac{(k-1)}{N} \right\rfloor + 1$;
4. $j = (k-1) \bmod \mathrm{N}$;
5. $d1 = (j-i) + N$;
6. $d2 = (j+i) - 1$;
7. End Findpos .

Fig. 3. Algorithm **Findpos**.

```
1.  Count (T0, L, n)
2.  Begin
3.     T#=0 =T0 ;
4.     T#=1 =empty ;
5.     T#>1 =empty ;
6.     For p=0 to n-1 Do Begin
7.           separate (T#=1, L+p+1, T1, T2) ;
8.           T#>1 =merge (T1, T#>1) ;
9.           T#=1 = T2 ;
10.          separate (T#=0, L+p+1, T1, T2) ;
11.          T#=1 =merge (T1, T#=1) ;
12.          T#=0 = T2 ;
13.    End for ;
14.    Return T#=1 ;
15. End Count .
```

Fig. 4. Algorithm **Count**.

Using the **Count** algorithm we can write the algorithm to extract the memory complexes in which one queen is placed in each row. The procedure **RowConstraint** shown in Figure 5 takes a test tube T_0 as the input parameter. T_0 contains all the memory complexes corresponding to the defined arrangements of queens in the grid that have exactly one queen in each row. In this procedure the position of each queen in each column is recorded in the positions from m to $m+n$ of the strands.

Later the memory complexes with a single queen in each column are selected by the procedure **ColumnConstraint** given in Figure 6. Suppose that a queen is to be assigned to column j. If no queen was assigned to this column previously, the column count is set to one. On the other hand, if currently there is another queen in that column (the column count is one) the memory complex is dropped because it has two or more queens in this column.

```
1.  RowConstraint (T0)
2.  Begin
3.     For p=0 to N-1 Do Begin
4.        T0 = count (T0, p*N, N) ;
5.     End for ;
6.  Return T0 ;
7.  End Rowconstraint .
```

Fig. 5. Algorithm **RowConstraint**.

```
1.  ColumnConstraint (T0)
2.  Begin
3.      For k=1 to m Do Begin
4.          Findpos (k, i, j, d1, d2) ;
5.          separate (T0, k, T1, T2) ;
6.          separate (T1, m+j, T3, T4)  ;
7.          set (T4, m + j) ;
8.          T0= merge (T2, T4) ;
9.      End for ;
10.     Return T0 ;
11. End ColumnConstraint .
```

Fig. 6. Algorithm **ColumnConstraint**.

Consequently, the **ColumnConstraint** procedure omits the memory complexes with *2* or more queens in each column.. Similar algorithm must be employed for diagonals. The **DiagonalConstraint** Procedure shown in Figure 7 does this.

With regard to above discussions our N-Queen sticker algorithm is shown in Figure 8. This algorithm takes a test tube T_0 as the input parameter. This test tube T_0 contains a library $\left[m + N + 2(2N-1), \binom{m}{N} \right]$ as defined earlier. Finally the algorithm returns all valid arrangements of N-Queen problem as DNA strands in test tube T_0.

```
1.  DiagonalConstraint (T0)
2.  Begin
3.    For k=1 to m Do Begin
4.          Findpos (k, i, j, d1, d2) ;
5.          separate (T0, k, T1, T2) ;
6.          separate (T1, m+N+d1, T3, T4)  ;
7.          separate (T4, m+N+2(N-1)+ d2,  T5, T6) ;
8.          set (T6, m+N+d1)  ;
9.          set (T6, m+N+2(N-1)+d2) ;
10.         T0= merge (T2, T6) ;
11.     End for ;
12.     Return T0 ;
13. End DiagonalConstraint .
```

Fig. 7. Algorithm **DiagonalConstraint**.

```
1.  N-Queen (T0)
2.  Begin
3.      T0= RowConstraint (T0) ;
4.      T0= ColumnConstraint (T0) ;
5.      T0= DiagonalConstraint (T0) ;
6.      If T0 is not empty Then
7.          Return T0 ;
8.      Else
9.          Teport “No Response Found” ;
10. End N-Queen .
```

Fig. 8. Algorithm **N-Queen**.

Now the time analysis of the above algorithms is discussed. As we can see the **for** loop in the **Count** algorithm iterates n times (and we call it with n equal to N). Every loop iteration requires constant time. Hence the complexity of the **Count** algorithm is $O(N)$. The total time taken to check the row constraint is $O(N^2)$ because it requires N Counting steps. The **ColumnConstraint** and **DiagonalConstraint** algorithms require m ($=N^2$) time steps. Hence their complexity are $O(N^2)$. Considering all the operations in total, we see that the N-Queen problem can be solved in $O(N^2)$ time.

Due to massive parallel inherent in DNA operations, our molecular algorithm does an exhaustive search in the solution space, and all the solutions are obtained in a polynomial time. The other presented algorithms in the literature [Crawford (1992); Mandziuk (2002); Ohta (2002); Smith (1999); Sosic and Gu (1994); Xu *et al.* (2005); Zhong *et al.* (2005)] use heuristic and machine learning methods for this problem and they can find an optimal or near-optimal solutions.

5. Conclusion

In this paper the N-Queen problem is formulated by DNA sequences and a polynomial time molecular algorithm is given for its solution. The employed model for our algorithm is a sticker model. The proposed algorithm can be easily extended to solve quadratic assignment, maintenance scheduling, resource allocation, and the graph matching problem.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful suggestions.

References

Adleman, L.M. (1994): Molecular computation of solutions to combinatorial problems. Science, **266**, pp. 1021-1029.

Ahrabian, H.; Ganjtabesh, M.; Nowzari-Dalini, A. (2005): DNA algorithm for an Unbounded Fan-in Boolean circuit. Biosytems, **82**, pp. 52-60.

Ahrabian, H.; Nowzari-Dalini, A. (2004): DNA simulation of NAND Boolean circuits. Adv. Model. Optimiz., **6,** pp. 33-41.

Barua, R.; Misra, J. (2002): Binary arithmetic for DNA computers. Lect. Notes in Comput. Sc., **2568** , pp. 124-132.

Beaver D. (1995): Computing with DNA. J. Comput. Biol., **2** , pp. 1-13.

Benenson, Y.; *et al.* (2003): DNA molecule provides a computing machine with both data and fuel. Proc. Natl. Acad. Sci., **100** , pp. 2191-2196.

Boneh, D.; Dunworth, C.; Lipton, R.J. (1996): Breaking DES using a molecular computer. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **27**, pp. 37-66.

Braich, R.S.; *et al.* (2002): Solution of a 20-variable 3-SAT problem on a DNA computer. Science, **296**, pp. 499-502.

Crawford, K.D. (1992): Solving the N-Queens Problem Using Genetic Algorithms. In Proceedings ACM/SIGAPP Symposium on Applied Computing, Kansas City, pp. 1039-1047.

Faulhammer, D.; *et al.* (2000): Molecular computation: RNA solutions to chess problems. Porc. Natl. Acad. Sci., **97**, pp. 1385-1389.

Ferretti, C.; *et al.* (2000): On the universality of post and splicing systems. Theoret. Comput. Sci., **231**, pp. 171-180.

Freund, R.; Kari, L.; Paun, G. (1999): DNA computation based on splicing: The existence of universal computers. Theory Comput. Syst., **32**, pp. 69-112.

Frisco, P. (2000): Parallel arithmetic with splicing Romanian J. of Inform. Sci. and Tech., **3**, pp. 113-128.

Gao, Y.; *et al.* (1999): DNA implementation of nondeterminism. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **48**, pp. 137-148.

Gupta, V.; Parthasarathy, S.; Zaki, M.J. (1999): Arithmetic and logic operations with DNA. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **48**, pp. 149-160.

Kari, L.; *et al.* (1998): DNA computing, sticker systems, and universality. Acta Inform., **35**, pp. 401-420.

Leier, A.; *et al.* (2000): Cryptography with DNA binary strands. Biosystems, **57**, pp. 13-22.

Lipton, R.J. (1995): DNA solution of hard computational problem. Science, **268**, 542-545.

Liu, Q.; *et al.* (2000): DNA computing on surfaces. Nature, **403**, pp. 175-179.

Mandziuk, J. (2002): Neural networks for the N-Queens problem: A review. Control and Cybern., **31**, pp. 217–248.

Mateescu, A.; *et al.* (1998): Simple splicing systems. Discrete Appl. Math., **84**, pp. 145-163.

Ogihara, M.; Ray, A. (1999): Simulating Boolean circuits on DNA computers. Algorithmica, **25**, pp. 239-250.

Oliver, J. (1999): Computation with DNA: Matrix Multiplication. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **44**, pp. 113-122.

Ohta, M. (2002): Chaotic neural networks with reinforced self-feedbacks and its application to N-Queen problem. Math. Comput. Simulation, **59**, pp. 305–317.

Ouyang, Q.; *et al.* (1997): DNA solution of the maximal clique problem. Science, **278**, pp. 446-449.

Paun G.; Rozenberg, G. (1998): Sticker systems Theoret. Comput. Sci., **204**, pp. 183-203.

Rothemund, P. (1996): A DNA and restriction enzyme implementation of Turing machines. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, **27**, pp. 75-119.

Roweis, S.; *et al.* (1998): A sticker-based model for DNA computation. J. comput. Biol., **5**, pp. 615-629.

Smith, K.A. (1999): Neural networks for combinatorial optimization: A review of more than a decade of research. Informs J. Comput., **11**, pp. 15–34.

Sosic, R.; Gu, J. (1994): Efficient local search with conflict minimization: A case study of the N-Queen problem. IEEE Trans. Knowl. Data Engineering **6**, pp. 661-668.

Su, X.; Smith, L.M. (2004): Demonstration of a universal surface DNA computer. Nucleic Acids Res., **32**, pp. 3115-3123.

Wang, L.; *et al.* (1999): Surface-based DNA computing operations: Destroy and readout. Biosystems, **52**, pp. 189-191.

Wasiewicz, P.; *et al.* (2000): The inference based on molecular computing Int. J. Cybern. Syst., **3**, pp. 283-315.

Winfree, E. (2003): DNA computing by self-assembly. The Bridge, **33**, pp. 31-38.

Winfree, E.; *et al.* (1998): Design and self-assembly of two-dimensional DNA crystals. Nature, **394**, pp. 539-545.

Wirth, N. (1976): *Algorithms + Data Structures = Programs*, Prentice Hall, Englewood Cliffs.

Xu, X.; Tang, Z.; Wang, J. (2005): A method to improve the transiently chaotic neural network. Neurocomputing, **67**, pp. 456–463.

Yaglom A.M.; Yaglom, I.M. (1964): *Challenging Mathematical Problems with Elementary Solutions*, Holden-Day, San Francisco.

Zhong, W.; Liu, J.; Jiao, L. (2005): Evolutionary Agents for N-Queen Problems. Lect. Notes in Comput. Sc., **3612**, pp. 366 – 373.

Zimmermann, K. (2002): Efficient DNA sticker algorithms for graph theoretic problems. Comput. Phys. Comm., **144**, pp. 297-309.