Proposal of Tabu DNA Computing Algorithm to Solve N-Queens Problem

Dr. Ahmed Tariq Sadiq
Dr. Hasanen S. Abdullah
Mohmed Natiq Fadhel

Abstract

The N-Queens problem is considered as one of the hard problem to be solved. Many researches have been interested to solve it with different intelligent methods. This paper presents a new hybrid technique to solve N-Queen problem. The proposed method depends on DNA-computing and Tabu search algorithm. Tabu search uses as a tool to increase the performance of the system by reduce the run time and memory capacity as well as the number of the random generated states. The experimental results with proposed hybrid method gives the best results compare with classical DNA-computing. The proposed method gives all solutions with size 11-by-11 as maximum, and multi solutions with size 20-by-20 as maximum. The run time of the proposed system is les than the classical DNA-computing by 50% and the reduction in memory capacity is 75%.

مقترح خوارزمية حسابات الـ DNA المحرمة لحل مشكلة N-Queens

د. أحمد طارق صادق
د. حسنين سمير عبد الله
محمد ناطق فاضل

الخلاصة

تعد مشكلة N-Queens من المشاكل الصعبة الحل. عدة بحوث تناولت حل الموضوع بمختلف الطرق الذكية. في هذا البحث سيتم تقديم تقنية مهجنة جديدة لحل مشكلة N-Queens. الطريقة المقترحة تعتمد على حسابات الحامض النووي والبحث المحرم. البحث المحرم يستخدم كأداة لزيادة اداء النظام من خلال تقليل وقت التنفيذ وسعة الذاكرة وبالتالي تقليل الحالات العشوائية المولدة. نتائج التجارب للنظام المقترح اثبتت انها افضل من حسابات الحامض النووي التقليدية. نتائج النظام المقترح اعطت كل الحلول الممكنة لمصفوفة حدها الاقصى 11*11 وعدة حلول لمصفوفة حدها الاقصى 20*20. وقت التنفيذ للنظام المقترح كان اقل من الخوارزمية التقليدية بنسبة 50% وسعة الذاكرة أقل بنسبة 75%.

1. Introduction

The eight queens is a well known NP-complete problem proposed by C. F. Gaus in 1850. The Problem was investigated by several 19th 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. A general N-Queen problem is defined by the following constraints on an *N*N* grid **[1]**:

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.

Because queens can move any number of squares vertically in a single turn, such placement is not cost efficient. Rather than viewing the board as consisting of n x n squares, it can be seen as comprised of n columns, each with n rows. Because we are placing n queens in n columns, we can determine that there has to be one and only one queen in each column. Applying this knowledge to the recursive algorithm makes placing the queens more efficient. This is also true for the horizontal direction, as we can rotate the board. As for the last, the diagonal direction must be accounted for as well **[2].**

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 **[1].**

In this paper, tabu DNA computing algorithm has been proposed to solve N-Queens problem and compare with classical DNA computing algorithm.

2. DNA Computing

DNA is the basic medium of information storage for all living cells. It contains and transmits the data of life for billions of years. Roughly 10 trillion DNA molecules could fit into a space the size of a marble. Since all these molecules can process data simultaneously, you could theoretically have 10 trillion calculations going on in a small space at once **[3]**. DNA computing began in 1994 when Leonard Adleman has first shown that computing can be done using DNA also, without using usual machine but using test tubes etc. in biological laboratory **[4]**. In the language of computer scientists, 8 binary bits correspond to 1 byte. As the DNA bases are 4 (C, G, A or T instead of 0 or 1), DNA requires half the amount of base pairs, i.e. 4, instead of 8 binary bits, to make one "genetic byte". In DNA computing, also known as molecular computing, a DNA computer is basically a collection of specially selected DNA strands whose combinations will result in the solution to some problem, depending on the problem at hand, technology is currently available both to select the initial strands and to filter the final solution **[5]**.

The fundamental schema of a DNA algorithm, for solving an instance of a combinatorial problem, is the following **[6]**:

i) Generation of a pool DNA strands encoding all possible solutions (the solution space).

ii) Extraction of those that are the true solutions of the given instance.

The second step is performed by a sequence of elementary extraction sub-steps, where at each sub-step all the strands where a specific sub-strands occurs are selected from the pool and constitute the input for the next extraction sub-step. These two steps are usually of complexity that is linear in time with respect to the size of the given instance. This is the conceptual strength of DNA computing **[5].**

There are some available steps (materials and bio-laboratory techniques) that have been used to build the DNA based computing models. These steps are essentially the following **[5, 7]**:

1. **Watson-Crick pairing.** Every strand of DNA has its Watson-Crick complement. As it happens, if a molecule of DNA in solution meets its complement, then the two strands will anneal.
2. **Polymerases.** Polymerases copy information from one molecule into another.
3. **Ligase.** Ligases bind molecules together. For example, DNA ligaes will take two strands of DNA in proximity and covalently bond them into a single strand.
4. **Restriction Enzymes (Nucleases).** Nucleases cut nucleic acids; any double stranded DNA that contains the restriction site with in its sequence is cut by the enzyme at that point.
5. **Polymerase Chain Reaction (PCR)**
   - Polymerase Chain Reaction
   - Amplifies (produces identical copies of) selected DNA molecules.
   - Makes 2n copies (n: number of iteration)
   - Solution filtering or amplification step.
6. **Gel Electrophoresis.**
   -Separates RNA, DNA and oligonucleotides by length.
   - Nucleic acids are mixed with porous gel.
   - Molecules can be seen through staining or other methods.
   - Electrophoresis purifies molecules.
7. **DNA synthesis.** It is now possible to write a DNA sequence on a piece of paper, send it to commercial synthesis facility and receive a test tube containing the written sequence of DNA.

3. Tabu Search

The tabu search algorithm combines a few simple ideas into a remarkably efficient framework for heuristic optimization. Among the main elements of this framework are:

- A (usually) *greedy local* search; the next solution is usually the best not-yet visited
- solution in the current neighborhood.
- A mechanism (*the tabu list*) discouraging returns to recently visited solutions.
- A mechanism that changes the solution path (perhaps by a random move) when no progress has been made for a long time.

In addition, many tabu search algorithms incorporate other features such as flexible memory structures and dynamic aspiration criteria **[8]**. A rough overview of a conceptual tabu search algorithm is given figure 1. For difficult searches, techniques such as influential diversification **[9]** may be used to extend the duration and scope of the search.

Although a tabu search is conceptually simple, any implementation of an efficient tabu search algorithm is problem specific, and no generic tabu search software is available at this time. Among the issues faced by designers of tabu search algorithms are **[10]**:

- The nature of the information included in the tabu list.
- The way the tabu list is organized.
- The lengths of the tabu lists.
- The types of moves used to create new solutions.
- The implementation of the local greedy search algorithm.

- Strategies for diversifying the search when no progress has been made for a while.

```
Initialize
Identify initial Solution
Create empty TabuList
Set BestSolutionDSolution
Define TerminationConditions
Done=FALSE
Repeat
    if value of Solution > value of BestSolution then
        BestSolutionDSolution
    if no TerminationConditions have been met then
    begin
        add Solution to TabuList
        if TabuList is full then
            delete oldest entry from TabuList
        find NewSolution by some transformation on Solution
        if no NewSolution was found or
        if no improved NewSolution was found for a long time then
            generate NewSolution at random
        if NewSolution not on TabuList then
        Solution = NewSolution
    end
    else
    done=TRUE
until done=TRUE
```

Figure 1. A Rough Overview Of A Conceptual Tabu Search Algorithm.

In addition, many of the elements used in a tabu search are themselves heuristic algorithms, most of which rely on some carefully selected parameter value for optimal performance. Few generalizable guidelines are available for the choice of such parameters. These concerns are discussed later in the paper. But first, let us review the main elements of the tabu search algorithm **[10]**.

4. The Proposed Technique

The DNA computing algorithm for the N-Queen problem required several procedures to achieve and perform the potential solutions. Although the N-queen problem has multi and different solutions, according to the number of the queen in the application (Grid dimension of the chess board), as well as it consume different time and memory capacity to implement each case.

Thus in this research there is a new idea to implement the N-queen problem through DNA computing process, this idea presents an easy approach to include the Tabu search inside the DNA processes (stages) to be one of the major stages as an alternative process with the GEL electrophorses and the synthesis stages.

4.1 The Tabu DNA Computing Approach

This approach consists of the following major stages

1. Knowledge Representation: each queen has its own pair codes (the Watson Crick pairing), the first one represent the DNA name of each one while the second one represent the inverse of the first name, each DNA name has ten characters (combination of the A, C,G, and T DNA bases) as an assumed name length.
2. Primary Process: in this process (stage) there are two steps: **Ligase** and **Restriction enzyme (Nucleases)**, the Ligase step is for deciding the permit number of locations in the 2D matrix to each queen, while the Restriction enzyme is to convert each location as a resulting place in the 2D matrix to the base location in each row in that matrix. As an example: The 4-queen problem manipulate as follow:
   The Watson Crick Pairing is
   P0: ACATCTCTCT P0c: TGTAGAGAGA
   P1: ATCTATCAGA P1c: TAGATAGTCT
   P2: TACTCTATCA P2c: ATGAGATAGT
   P3: CTCTCTCTGA P3c: GAGAGAGACT
   The primary process for an instance is as follow
   -The matrix is 4*4 two dimensions.
   -One of the location distributions is 6,9,16,4 this is the ligase step
   - The Restriction enzyme for this instance is the first row in the 4*4 matrix has the first queen in the last location (4), the second row has the second queen in the second location (6), the third row has the third queen in the first location (9), and the fourth row has the fourth queen in the last location (16).
   The final representation of the four-queen in the 4*4 matrix is 3,1,0,3 (since each row in the matrix has four locations 0,1,2,and3).
3. The Generating Random Solutions stage: in this step the computing system is responsible to generate the random solutions which include the right and the wrong solutions, according to the N-queen rules, this is the very important stage because the potential right solutions finding is depend this process in other words, if the generating process is strong, the right solutions will be exist as much as it possible.
4. The PCR stage to eliminate the results that contain repetitions in location with different queens according to the grid (the chess board).
5. The tabu search for the right solutions as the following rules:
   -if there is at most one queen in each row and one queen in each column, then continue to process, other wise ignore the result.
   - according to the previous rule; if there is at most one queen in each of two diagonal, then the result is true (the solution is existed), otherwise ignore the result.
   - for the next result, if the result is true, check it with previous true solutions to prevent the repetitions in the solutions list.
   -for each grid (chess board) there are a limited number of solutions, therefore when the number of the right solutions reach to that limited number, the search process must be stopped at that time.

4.2 The Tabu DNA Computing System Design

The DNA computing system has its own design method that can be describe the overall system stages to solve each problem (application), thus the Tabu DNA approach has a special way to design, since the N-queen problem is considered a hard problem to be solved.

The designing of the proposed system can be described through the following architecture that contains the major stages of the tabu DNA computing approach as in the figures (2) and (3) bellow.

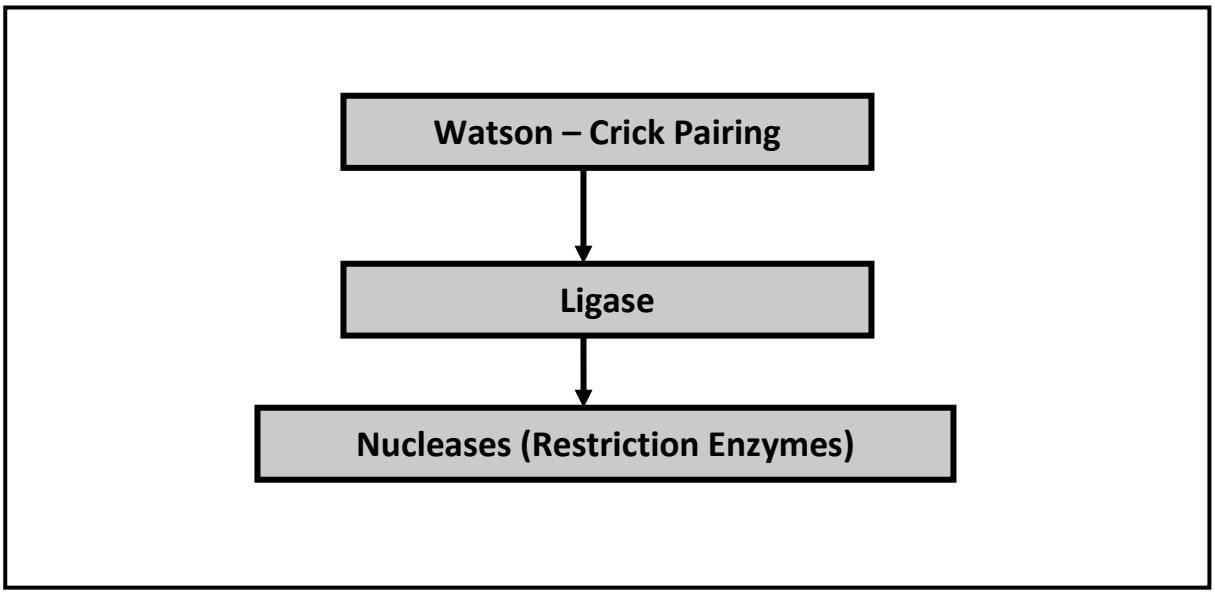

Figure (2) The Knowledge Representation And The Primary Process
Of The Tabu DNA Computing Approach

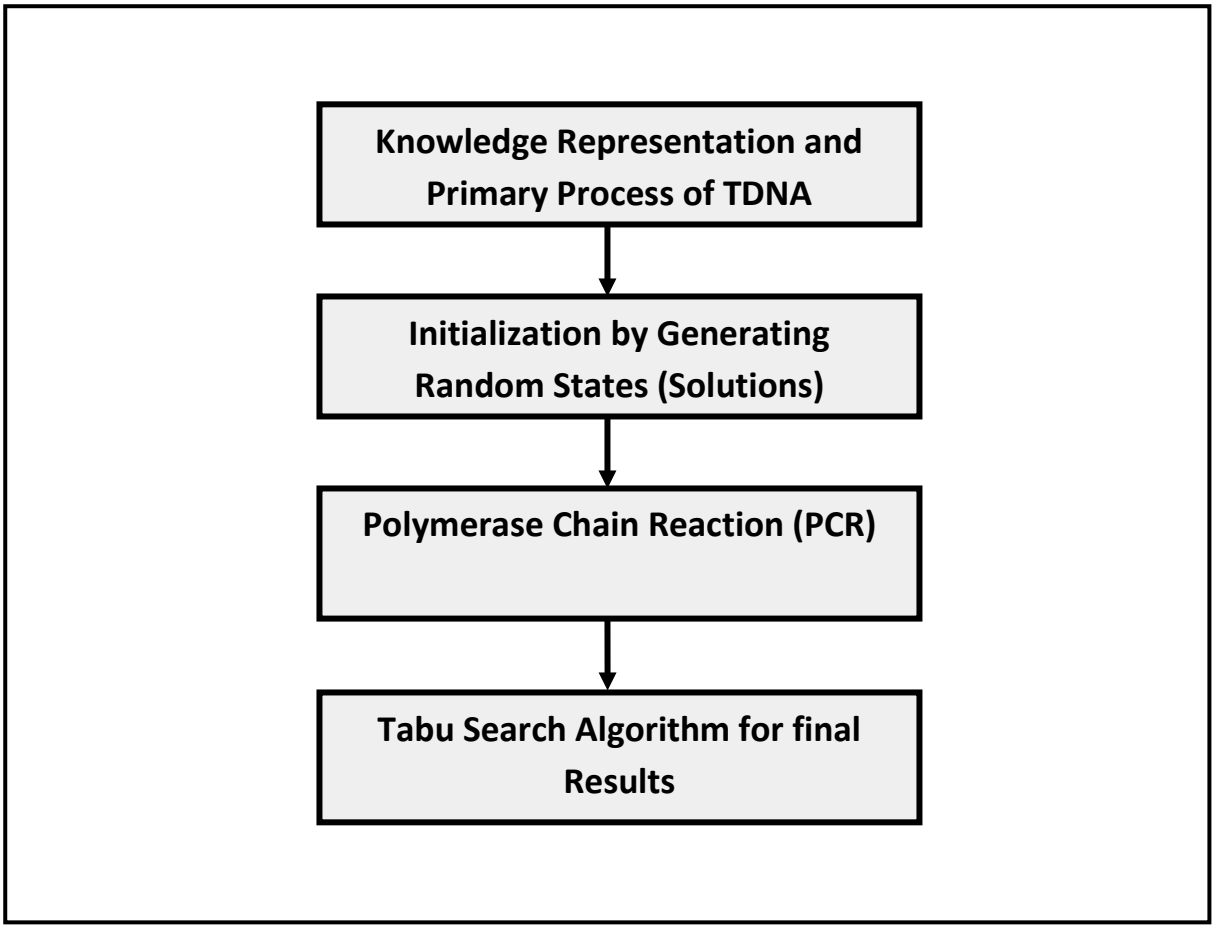

Figure (3) The Architecture Design Of The Tabu DNA Computing Approach

4.3 The Tabu DNA Computing Algorithm

This section represents detail description of the algorithm steps for the Tabu DNA computing system as an approach to improve the mechanism process of the DNA computing for solutions searching, the part bellow represents a complete search algorithm to decide the Tabu DNA computing approach to solve the N-queen problem with different chess board starting with 4*4 grid to 20*20 and more. Figure (4) illustrates the proposed algorithm.

Procedure: Generate Random Results (States)
Input: Grid G with N*N chess board, M */* M= maximum length of each strand*/*
Output: Rgrr[i] Generating Results (States)
Begin
Assign each queen its suitable coding accoeding to the grid dimensions
For each queen q=1,2,3,…..n in turn */* Initialization*/*
I=0
Begin
Select the random queen qr
For each unvisited neighbor qn of qr */* Iteration */*
Begin
Compute the number of unvisited neighbors of qn

Select qr+1
Iterate until the last selected queen has no unvisited neighbors or length
of the strand = M
End;
Put the result in Rgrr[i]
i = i +1
End;
End.

/ The output: results (states) that contain queens such that each strand in Rgrr has no unvisited neighbors or length = M */*

Procedure: PCR (Polymerase Chain Reaction)
Input: Rgrr[i] Generating results (States), M
Output: Rpcr[j] Results (states)
Begin
J = 0
For i = 1 to N
Begin
If Rgrr[i] has length less than M then ignore the result
If Rgrr[i] contains repetition in location of different queens then ignore the result
Else Rpcr[j] = Rgrr[i]
j = j + 1
End;
End.

/ The output: results (states) that have exactly M queens and have not location repetitions */*

Procedure: Tabu
Input: Rpcr[j] PCR results (States)
Output: TabuList[k] Solutions
Begin
Identify initial conditions */* grid dimensions and locations */* **Initialization**
Create empty **TabuList** **Initialization**
Set Right **H** Solutions */* the limited number of solutions of each case */* **Initialization**
Define Termination Conditions k */* k reach to the limited number of solutions of each case H */* **Initialization**
k = 0
Done Rpcr[j] **FALSE**
Repeat
if value of Rpcr[j] = value of Right Solution **then**
Right Solution = Solution
if no Termination Conditions have been met **then**
begin
add Solution to TabuList[k]
k = k+1
j = j+1
if TabuList is full **then**
delete oldest entry from TabuList
find New Solution by some transformation on Solution
if no New Solution was found **or**
if no improved New Solution was found for a long time **then**

```
      /* Solutions that have not similarities in some locations */
       generate New Solution at random
       if New Solution not on TabuList then
       Solution Rpcr[j] New Solution
      end;
       else
 done Rpcr[j] TRUE
   until done Rpcr[j] TRUE
end.
/* The result: final solutions or target solutions TabuList[k] */
Main: Tabu DNA Algorithm
Input: Grid G with N*N chess board, M
Output: TabuList[k] Solutions
Begin
    Knowledge representation (Watson Crick Pairing)
    Primary Process
    Generate Random Results (States)
    PCR
    Tabu
End.
```

Figure (4) The Proposed Tabu DNA Algorithm.

5. Test Results

The following figures (5, 6 and 7) represent real results of implementing the TDNA proposed system in two ways; the first one is the results of solving N-queen problem through the normal (standard) DNA computing approach while the second one is the results of solving N-queen problem through the Tabu DNA computing approach.

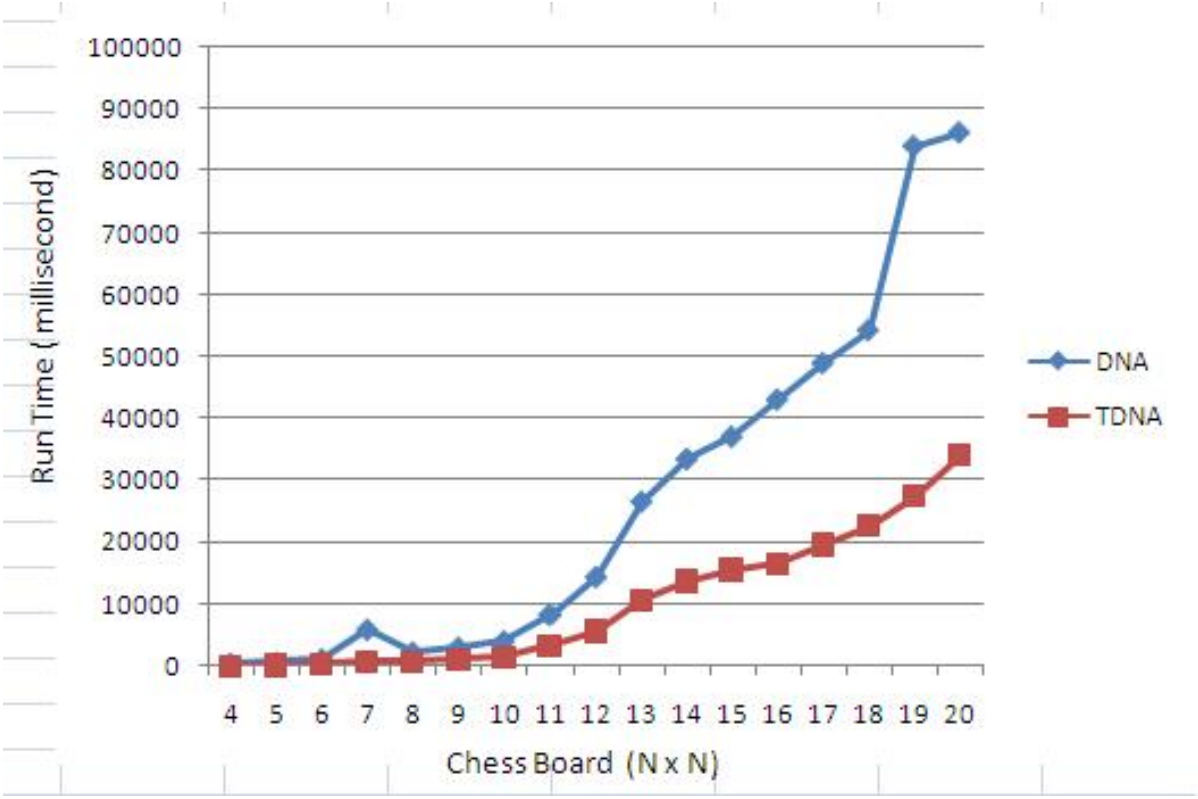

Figure (5) Run Time Curve

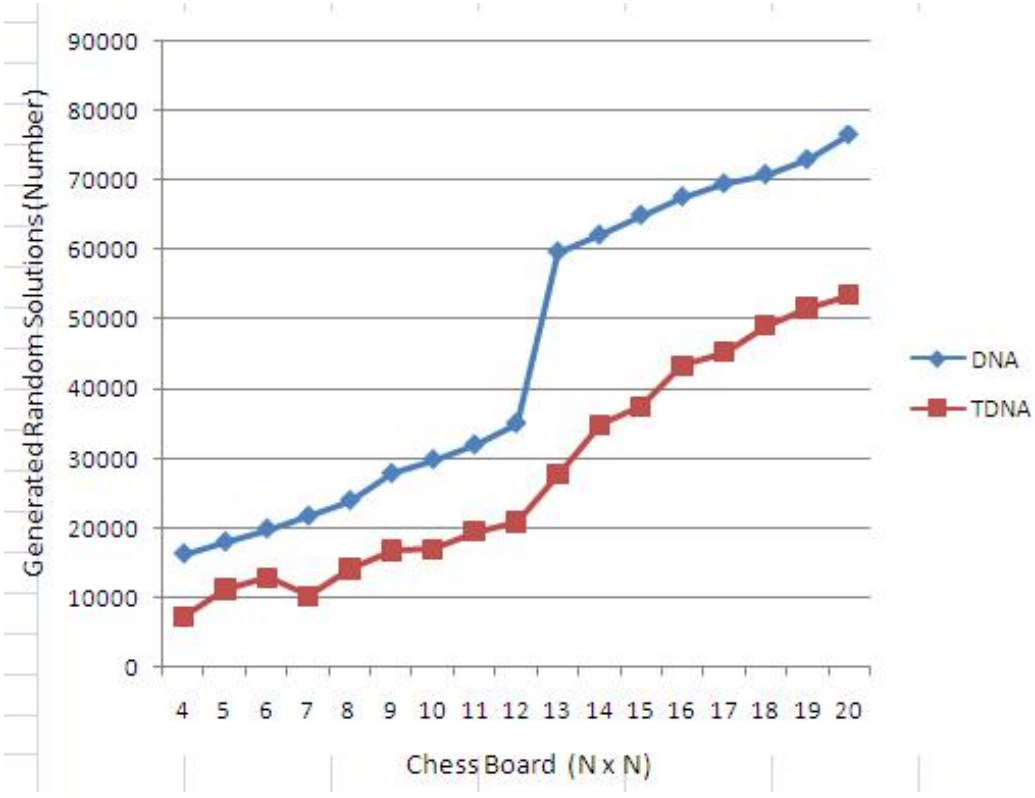

Figure (6) Generated Random Solution Curve

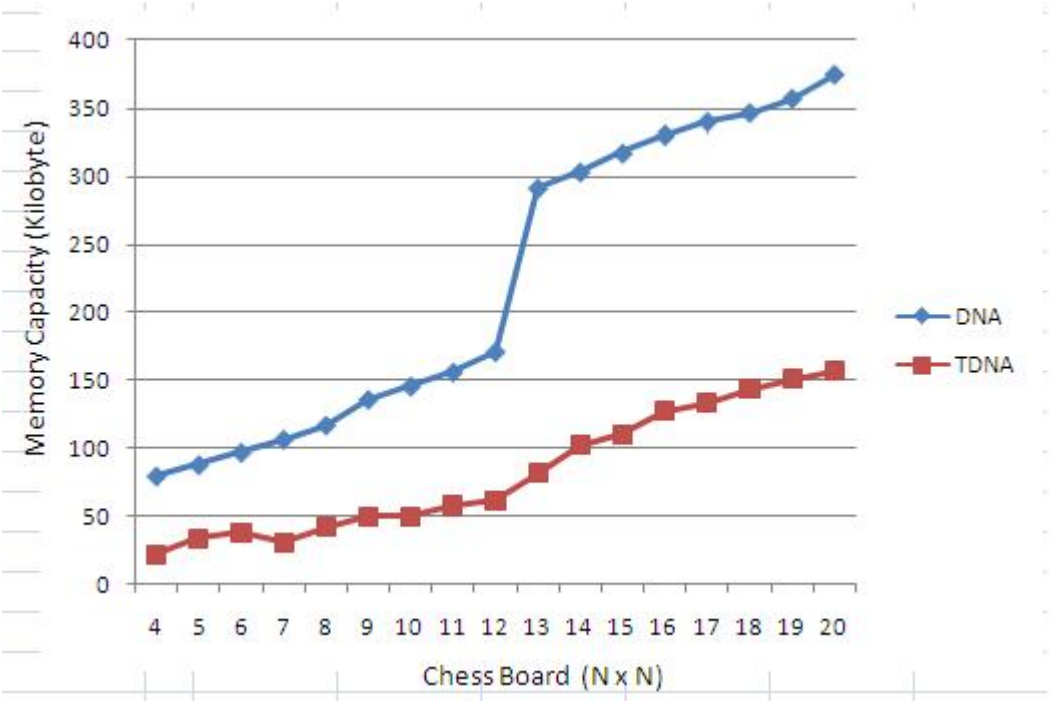

Figure (7) Memory Capacity Curve

6. Conclusions

The tabu DNA computing system has several properties that make it more preferred to be used to solve the N-queen problem than the normal DNA computing system which are as follow:

- The random generating process is more efficient and limited than it in the normal DNA computing system. It is efficient because it contains all the possible solutions in each grid from 4*4 to 11*11 grid, while it is limited because it contains less number of random generating results (the generating process becomes conditionally with the size of the grid or the chess board).
- The run time of the system to perform each case of the N-queen problem is reduced at least to the half amount than in the normal DNA computing system of the same case.
- The memory capacity that is needed to store the instances and the intermediate and final operation and results of each case of the N-queen problem is reduced at least to the quarter amount than in the using (implementing) the normal DNA computing system of the same case.
- The Tabu DNA approach is able to find right multi solutions of the N-queen problem for more complex cases reach to 20*20 grid (chess board) and more.
- The number of solutions for the complex cases (from 12*12 grid and more) is not reach to maximal solution list this is because of the tabulist is limited; a recommended way is to make the tabulist dynamic.

References

1. H. Ahrabian, A. Merzaei and A. Nowzari-Dalini, ***"A DNA Sticker Algorithm for Solving N-Queen Problem***", International Journal for Computer Science and Applications, vol. 5, No. 2, pp 12-22.
2. Jesper Goos, ***"Using Genetic Programming to Solve the n Queens Problem***", Datalogi Roskilde Universitetscenter, November 2005.
3. ***"DNA Computing, State of the Art"***, CPSC 601.73 http://www.cpsc.ucalgary.ca/~omair/cpsc60173/presentation, 28-1-2003.
4. G. P. Raja Sekhar, ***"DNA Computing-Graph Algorithms"***, The Indian Programmer, Supported by Com MaC-KOSEF, Korea, 2003.
5. T. Zingel, ***"Formal Models of DNA Computing: A survey"***, Proc. Estonia Acad. Sci. Phys. Math., 49, 2, 90-99, 2000.
6. G. Franco, C. Ciagulli, C. Laudanna and V. Manea, ***"DNA Extraction by Cross Pairing PCR"***, Springer, 2005.
7. L. M. Adleman, ***"Computing with DNA"***, Magazine: Scientific American, 1998.
8. F. Glover, ***"Tabu Search Fundamentals and Uses, Revised and Expanded"***, Technical Report, April 1995.
9. Hubscher, Roland and F. Glover., ***"Applying Tabu Search with Influential Diversification to Multiprocessor Scheduling",*** Computers in Operations Research 21, 877–844.
10. Arne Thesen, *"**Design and Evaluation of Tabu Search Algorithms for Multiprocessor Scheduling**"*, Journal of Heuristics, Vol. 4, pp.141-160, Kluwer Academic Publishers, 1998.