Solving N-Queen Problem with Stimulated Annealing

Akila Panditha
Department of Computer Science and Engineering
University of Moratuwa
Sri Lanka
+94 714 407 683
akiladarshana@gmail.com

Pasindu Perera
Department of Computer Science and Engineering
University of Moratuwa
Sri Lanka
+94 715 363 253
perera.pasindu@gmail.com

Nipuni Perera
Department of Computer Science and Engineering
University of Moratuwa
Sri Lanka
+94 715 626 680
pereranipuni@rocketmail.com

ABSTRACT

In this paper, we discuss about using stimulated annealing approach to solve N-queen problem which represents the class of NP-complete problem. This paper includes brief description about NP-complete problems, a detailed discussion on using stimulated annealing approach and a demonstrative algorithm to solve N-queens problem.

Categories and Subject Descriptors

B.2.4 [HIGH-SPEED ARITHMETIC] Optimization : Stimulated annealing

General Terms

Algorithms, N-Queens Problem

Keywords

Algorithms, Stimulated Annealing, NP complete, N queens, Intelligent Systems

1. Introduction to NP complete problems

The class P (Polynomial) refers to the set of decision problems for which set of polynomial time algorithm exist [4]. NP problems are any class of computational problems for which no efficient solution algorithm has been found. There are many significant computer science problems which belong to this class.

The comparison of P and NP problem is a major unsolved problem in the computer science. Informally it is agreed that P ≠ NP. This stated that a problem which can be verified in polynomial time may not be solved in polynomial time. For the time being only known algorithms for NP problems are exponential in number of operations. So they are not practically solvable for large n values. If a solution for a problem of size n, the time or number of steps needed to find the solution is polynomial function of n, then the problem is tractable. In contrast, algorithms for solving intractable problems require times that are exponential functions of the problem size n. These exponential time algorithms are considered inefficient since, the execution time increases exponentially with the growth of problem size n.

A problem is called NP (non deterministic polynomial), if its solution can be verified in polynomial time. Due to high complexity NP problems are not solvable in a reasonable amount of time. A problem is said to be NP hard if all the problems in the NP hard can be reducible to that problem. A problem which is both NP (verifiable in nondeterministic polynomial time) and NP hard (any NP problem that can be translated into this problem) are classified as NP complete.

Finding an efficient algorithm for any NP complete problem implies that an efficient algorithm can be found for all such problems, since any problem that belongs that class can be mapped to any other member of that class. A common property of solution an algorithm of these problems is that, as the problem size increases, the number of steps (or the time required) increases exponentially. One approach to solve an NP-complete problem is to use a polynomial algorithm to approximate the solution, so that the solution will be reasonably close to the optimal solution. Thus it is generally agreed that if a problem can be shown to be in class NP complete problems, no efficient algorithm can be found. Proving that a problem is in the intersection of NP and NP hard problems will show that the problem is NP complete.

Therefore heuristic methods are used to solve these problems in a reasonable amount of time. This paper analyses a heuristic algorithm, simulated annealing to solve an NP complete problem.

2. N-Queens problem

The N- queen problem was originally introduced in 1850 by Carl Gauss. It is a classical combinatorial problem in the artificial intelligence area. It can be stated as follows: find a placement of n queens on an *n x n* chessboard, such that no one queen can be taken by another. A queen can attack another queen vertically, horizontally or diagonally. E.g placing a queen on a central square of the board blocks the raw and column where it is placed, as well as the two diagonals (rising and falling) at whose intersection the queen was placed.

The eight queen’s problem is a classical sub problem of this which stated to find a way to place eight queens on chess board so that no queen would attack any other queen. It has been found that there are 92 solutions to this problem. All these 92 solutions are based on 12 patterns, where each of the 92 solutions can be mapped to one of those 12 patterns. N- Queen problem has generalized the eight queen problem altering n queens on an *n x n* chessboard.

This problem can be generalized as placing n non tracking queens on an n x n chessboard. Since each queen must be in different row and a column (to not to attack by another queen), it is possible to label a queen by its column number. All solution to the problem therefore can be represented in n-tuples that are permutations of an n-tuples. Figure 1 shows a one solution to eight-queen problem.

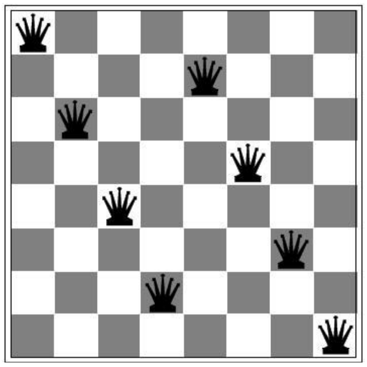

Figure 1: A solution of eight queens problem

The amount of time required to find all solutions for any order 'N' is roughly proportional to factorial N. It takes about 10 hours to get the results for N=26. Therefore the time requirement increases exponentially with the problem size n.

The exact cover problem which is kind of constraint satisfaction problem is a NP complete problem. The N queen's problem is a generalized exact cover problem, as the constraints corresponding to the diagonals of the chessboard need to be satisfied. The problem involves four kinds of constraints:

- *Rank : for each of the N ranks, there must be exactly one queen*
- *File: For each of the N files, there must be exactly one queen*
- *Diagonals: For each of the 2N-1 reverse diagonals there must be at most one queen*
- *Reverse diagonals: For each of the 2N − 1 reverse diagonals, there must be at most one queen. [5]*

A brute force algorithm for solving the n-queen problem which places a single queen in each row leads to n^n placements. The complexity of such approach is O(n!). As described above n-tuple representation only eliminates row and column conflicts, so that the wrong solutions can only arise due to diagonal attacks between queens. This approach leads to O(n) complexity while in the case of simulated annealing approach it is possible to reduce the complexity to O(1) [4]. There are many possible algorithms for finding a valid placement for a queen. This paper presents simulated annealing approach.

3. STIMULATED ANNEALING

Stimulated Annealing (SA) is a probabilistic Metaheuristic approach to solve global optimization problems. It simulated the phenomena of annealing in metallurgy, technique which uses controlled cooling of a heated material to obtain preferred characteristics.

3.1. Background

This method was described in two independent articles. Scott Kirkpatrick, C. Daniel Gelatt and Mario P. Vecchi in 1983 [1] and Vlado Černý in 1985 [2]

This approach is an efficient solution for the global optimization problem which is capable of locating a good approximation for global optimum of a given function in a large search space. Generally SA is used in discrete search spaces.

SA uses a random search in state space which is characterized by the physical process of annealing. This algorithm gives programmer to control its parameters allowing optimizing output according to the problem and desired output. In General with careful control of parameters, SA can be very efficient in finding the global optimum (3) .

3.2. Overview of Algorithm

A general SA algorithm can be visualized as follows.

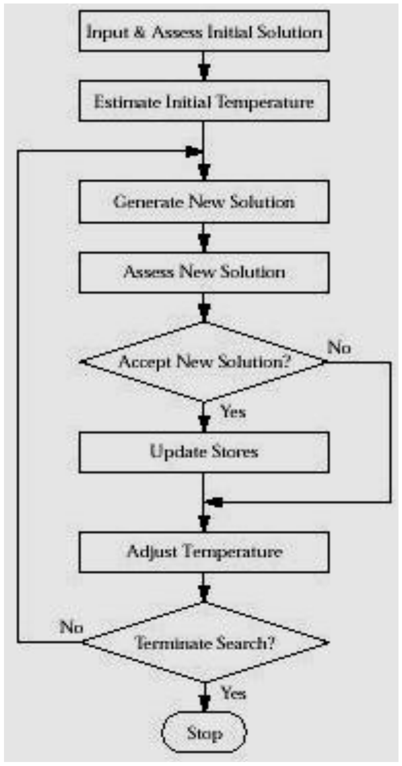

Figure 2: The structure of the simulated annealing algorithm[3]

General Explanation

SA algorithm performs random space search iteratively while accepting states via fitness function. The Major highlight of SA is temperature function which allows accepting non optimal solutions.

The best way to understand this is to visualize algorithm with example of a bouncing ball scenario in finding global minima. [3]

"bouncing ball that can bounce over mountains from valley to valley. It begins at a high "temperature" which enables the ball to make very high bounces, which enables it to bounce over any mountain to access any valley, given enough bounces. As the temperature declines the ball cannot bounce so high, and it can also settle to become trapped in relatively small ranges of valleys. A generating distribution generates possible valleys or states to be explored. An acceptance distribution is also defined, which depends on the difference between the function value of the present generated valley to be explored and the last saved lowest valley. The acceptance distribution decides probabilistically whether to stay in a new lower valley or to bounce out of it. All the generating and acceptance distributions depend on the temperature Function "

Temperature function determines the probability of accepting non optimal solution as the next state in a given iteration. Initially temperature function has a high value and it reduces with number of iterations. Another function called delta function determines the rate of decrease.

Rate of Decreasing Temperature (Delta function)

Delta function determines the way of decreasing temperature within iterations. Ultimately this function controls the probability of accepting non optimal solutions.

Next State Generator

Algorithm needs a state generator for use as inputs of evaluation function. Generally this component is either random generator or heuristic based generator depending on the selection of programmer.

Evaluation Function

Main functionality of this component is to evaluate generated next state with current state and decide whether to accept next state of keep current state as the solution.

General implementation of this includes two step evaluations. First evaluation based on heuristic and alternate probabilistic evaluation for failures in first evaluation. This allows accepting non optimal solutions based on probabilistic evaluation to avoid local minima/maxima situations.

Solution acceptance or termination Criteria

Programmer should include acceptance criteria or termination criteria to avoid infinite search instances. A lower bound for temperature is one of most commonly used termination criteria in practice.

3.3. Key Features

Flexibility

This algorithm gives programmer a significant freedom in selecting parameters. These parameters can be used to optimize output for a given criteria. Depending on nature of problem and dataset programmer can modify temperature function, Delta function, next state generator and evaluation function which ultimately controls the behavior of the algorithm.

Strengths

SA is a robust and general technique which can deal with highly nonlinear models with chaotic and noisy sets of data. [3]. When it comes to NP complete problems, it is one of few approaches which are able to handle large state spaces efficiently. [4] .

Not like other metaheuristic algorithms, SA performs heuristic evaluation in linear time. This is a huge advantage for SA in handling larger state spaces. [4]

Weaknesses

Due to highly flexible metaheuristic nature of this algorithm, Precision of selecting parameters have a high influence on quality of the results and performance of the algorithm. Therefore this algorithm needs extensive customizations in order to achieve best performance and high quality results for a given problem. In most of the cases these customizations involve tradeoffs between performance and quality. [3]

3.4. Comparison with Other Approaches

This section provides a comparison of SA with Tabu search and Genetic algorithms, two major metaheuristic approaches to solve NP complete problems. This section is based on the test results provided by **Martinjak, Ivica. and Golub, Marin.** [4] in solving N-queens problem.

SA takes polynomial time to solve N-queens problem. In contrast SA have time complexity ranging from O(n!) to O(n^2) while other two algorithms take O(n^3)

In case of SA, fitness function is calculated only once in an iteration. Due to this scenario, complexity of the fitness function of SA is O (1) while other two take O(n).

In contrast , SA approaches near solution fast and then take more time in final optimizations of the answer.

From these three metaheuristic approaches, SA is the only approach which can give answers for large dimensions in a realistic time frame.

4. DEMONSTRATION: SOLVING N-QUEENS PROBLEM

This section describes an implementation of SA on JAVA and describes how each of the functions is actually implemented.

Full solution is available on the following location with a comparison of SA method and the brute force approach with regards to time complexities. On the paper it only describes the implementation on SA and the code contains other classes used in deferent other purposes.

1. Basic Classes

This section describes the basic infrastructure to build SA on top of that. We will only describe the specific properties of the classes and all the methods will be encapsulated and getters and setters will be added to the class accordingly.

Queen class holds information about the position of the queen on the board.

```
public class Queen {
        int indexOfX, indexOfY;
        public Queen(int indexOfX, int indexOfY) {
        this.indexOfX = indexOfX;
                this.indexOfY = indexOfY;
        }
}
```

State class will hold data about the current state of the board. Both getNextState and the calculateCost are functions use user decide to enhance and extend the functionality

```
abstract class State {
        int boardSize;
        int cost;
        protected Queen q[];
        abstract public State getNextState();
        abstract public void calculateCost();
}
```

Nqueen class is the abstraction of the Nqueen Problem. It provides a common interface to get answers to the solution. It also compromise as showBoard() method simply output the current configuration for debugging purposes.

```
abstract class NQueen {

    protected  int boardSize;
    protected  State currentState, nextState;
    abstract public void solve();
    abstract public void show();
}
```

2. Simulated Annealing Logic

These are the concrete class that inherits the above classes that cater SA approach to solve the N-queens problem. We calculate the cost being the no of queens attacking each other in the current state.

```
public void calculateCost() {
        int i, j;
        cost = 0;

        for (i = 0; i < boardSize; i++) {
            for (j = 0; j < boardSize; j++) {
                if(anyQueen is attacking) {
                    cost++;
                }
            }
        }
        cost = cost / 2;
}
```

Since the number of queens are equal to the no of rows(columns) on the board. So we can safely assume that there will only be one queen on each row. (i.e.: ith queen will be on the ith row.) When generating the next state we only change the value of the column of the ith queen.

In the nextState implementation, we choose a random queen from the board and we will select randomly change its column value until it is not equal to the value it had before.

```
public State getNextState() {
        int i;
        Queen nextStateQueen[] = new Queen[boardSize];
        int rand = randomGenerator.nextInt(boardSize);

        for (i = 0; i < boardSize; i++) {
            nextStateQueen[i] = new Queen(q[i].getIndexOfX(),
                    q[i].getIndexOfY());
            if (rand == i) {
                int temp = randomGenerator.nextInt(boardSize);
                while (temp == q[i].getIndexOfY()) {
                    temp = randomGenerator.nextInt(boardSize);
                }
                nextStateQueen[i] = new Queen(q[i].getIndexOfX(),
temp);
            }
        }

        return new SimulatedAnnealingState(boardSize,
nextStateQueen);
    }
```

Simulated annealing class inherits the Nqueens class problem and it provides the logic to solve using the SA approach. Temperature function is initially set to a higher value and a random number is generated in the each temperature value. In each iteration if the new cost is less than the previous cost next state is always taken. Otherwise if that random value is less than the probability of the next state being evaluated, the next state will be taken again (even though it is not reducing the cost). Since the temperature is reducing the chances of getting a high cost state in the latter phases are reduces over time. But initially it will move to sub optimal states as the next states even the cost of those states are higher than the current state.

```
public void solve() {
        double temperature;
        double delta;
        double probability;
        double rand;

        for (temperature = 10000; (temperature > 0) &&
(currentState.getCost() != 0); temperature--) {
            nextState = currentState.getNextState();
            delta = currentState.getCost() - nextState.getCost();
            probability = Math.exp(delta / temperature);
            rand = Math.random();

            if (delta > 0) {
                currentState = nextState;
            } else if (rand <= probability) {
                currentState = nextState;
            }
        }
    }
```

5. ACKNOWLEDGMENTS

We would like to thank Mr. Thilak Fernando and Ms. Sachini Weerawardhana for their guidance and supervision in preparing this report.

6. REFERENCES

1. **S. Kirkpatrick, C. D. Gelatt, Jr. and M. P. Vecchi.** Optimization by Simulated Annealing. *Science.* May 13, 1983, Vol. 220, 4598, pp. 671-681.
2. *Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm.* **Černý, V.** 1, s.l. : Springer Netherlands, January 1, 1985, Journal of Optimization Theory and Applications, Vol. 45, pp. 41-51. ISSN 0022-3239.
3. **busetti, franco.** *Simulated annealing overview.* 2003.
4. *Comparison of Heuristic Algorithms for the N-Queen Problem.* **Martinjak, Ivica. and Golub, Marin.** Cavtat,Croatia : s.n., 2007. Proceedings of the ITI 2007 29th Int. Conf. on Information Technology Interfaces. pp. 759-764.
5. http://en.wikipedia.org/wiki/Exact_cover- Access date (24/03/2012)
6. https://github.com/rumal/N-Queens-with-Simulated-Annealing