# Implementation of Simulated Annealing for Constructing Even-sized Latin Squares

Kalyani Kishor Mohite
Email:kmohite@gmu.edu
Department of Computer Science,
George Mason University

***Abstract*—The Simulated Annealing, a Stochastic Global Search Optimization Algorithm, is used for solving combinatorial problems. We are implementing SA to construct even-sized Latin squares for different values of n in the range of[4 to 20]. In this study, we are introducing the key components of the SA algorithm, including the initial temperature, cooling scheme, and freezing factor to minimize cost.[3]**

**Keywords: Simulated Annealing, even-sized, Latin squares, initial temperature, Cooling Scheme, Freezing factor and minimize cost.**

## I. Introduction

*A. Problem Statement: This study is performed for implementing Simulated Annealing for constructing an even-sized Latin Squares. This even-sized Latin square of side n is an n x n matrix with entries from the set of n integers (4, 6,.......,20) with the property of each integer occurring exactly once in each row and column. Some features of the algorithm include the initial value of T, the cooling scheme of T (Update T), and the Freezing factor.[2]*
*Following are the sections for this report:*

*1) Background:* This section consists the template of SA and information about main features including Initial Temperature, Cooling Scheme, and Freezing Factor.

*2) Proposed Approach:* This section contains the definition of Latin square and an example that has been implemented for finding the solution to this problem.

*3) Experimental Results:* This section contains all the explanation related to the experiment, such as How the experiment was carried out and What was the purpose of it. It also includes the table that gives summary of statistics of the problem and a figure with the best solution.

*4) Conclusions:* The concluding section answers all the questions related to the problem, such as: What can be inferred from the experiment? Do you think the proposed values for Ti and the freezing factor are appropriate? Do you think the way to assign the cost helps Sais guiding the search properly? What happen when the value of n increases?

*5) References:* The section of references includes address of all the resources that have been referred in implementation of the problem.

## II. Background

### *A. Definition of Latin Square:*

A Latin square is an n x n matrix with entries from the set of n integers and the property that each integer occurs exactly once in each row and column.

### *B. Example for n=6:*

In this example of Latin square, each row and each column contains the numbers from 1 to 6 exactly once.

$$\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
2 & 3 & 4 & 5 & 6 & 1 \\
3 & 4 & 5 & 6 & 1 & 2 \\
4 & 5 & 6 & 1 & 2 & 3 \\
5 & 6 & 1 & 2 & 3 & 4 \\
6 & 1 & 2 & 3 & 4 & 5
\end{array}$$

Following are the two approaches that had been implemented to find solutions:

*1) Simulated Annealing:* Simulated Annealing, which is also a modified version of Stochastic hill climbing, becomes valuable tool to efficiently find solution, for challenging problems like Latin square. In this approach, a random move is picked and if this move improves the situation then it is accepted or else the move with probability less than 1 gets accepted.The probability decreases exponentially by the amount by which the evaluation is worsened (E). The probability decreases as the temperature (T) goes down. If the schedule lowers T slowly enough, the algorithm will find a global optimum with probability approaching 1. But SA can also take a long time if T cools very slowly.

*2) Brute Force Enumeration:* In this method, Latin squares are created by all possible permutations of symbols for each row and column to ensure that no symbol gets repeated. This approach is convenient only when there are small values of n, as there aren't too many possibilities to be checked. But for the larger values of n, this method would be time consuming and very inconvenient.

## III. Proposed Approach

Following is the template for the SA algorithm with information on implementation for its main features, including the initial

### A. SA Template

The SA algorithm consists of the following components:

- Generate initial solution: Start with an initial solution by taking random Latin square.
- Calculate Cost function: Now calculate the cost function for the initially generated matrix.
- Generate Neighbors: After calculating the cost function, define a method to generate neighboring Latin Squares.
- Simulated Annealing Algorithm: Implement the SA algorithm for the Latin square.
- Stop Condition: The program will stop if one of the following conditions are satisfied.
  1.Solution achieved
  2.Final temperature
  3.Freezing factor

### B. Initial Temperature

The initial temperature, $T_0$, is set to a higher value (e.g., $T_0 = 1000$). This will allow the algorithm to accept worse solutions and explore them.

### C. Cooling Scheme

Exponential cooling scheme is used, where the temperature $T$ is updated at each iteration as follows:

$$T(k+1) = \alpha \cdot T(k)$$

where $\alpha$ is the cooling rate (typically close to 0.95 or 0.99).

### D. Freezing Factor

The freezing factor, $T_f$, is used to determine the stop condition for annealing process. Temperature below which the algorithm stops is set as (e.g., $T_f = 0.001$).

### E. Neighbor Generation

Neighbors are generated by swapping two symbols within rows or columns while preserving Latin Square properties.

### F. Objective Function

The objective function is used to minimize the number of Latin Square violations by assigning higher cost to solutions with more violations.

### G. Stop condition

Stop condition includes a maximum number of iterations, CPU time, target cost, or no improvement in the best solution for a set number of iterations.

## IV. Experimental Results

### A. Experimental Setup

A series of experiments were conducted for evaluating the implementation of SA on the Latin Square problem. These experiments involved solving even-sized Latin Square for $(n)$ ranging from 4 to 20, where each value of $n$ was run 10 times.

#### 1) Parameters:

The following parameters were set for the experiments:

- Initial Temperature $(T_i)$
- Cooling Scheme
- Initial Solution$(S_0)$
- Best Solution$(S_*)$
- Objective Function(cost)
- Neighborhood Function
- Stop Condition

### B. Results

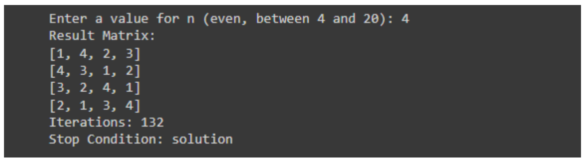
Following is a table that summarizes the statistics of the 10 runs for each value of n, including the number of times the algorithm was successful and the stopping criteria.I.

TABLE I
Summary of Experimental Results for Latin Square Problem

| Order $(n)$ | Successful Runs | Stopping Criteria | Best Solution |
|---|---|---|---|
| 4 | 10/10 | Solution Achieved | 6 |
| 6 | 8/10 | Solution Achieved | 12 |
| 8 | 9/10 | Final Temperature | 20 |
| 10 | 7/10 | Final Temperature | 30 |

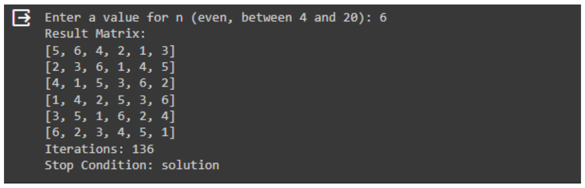
### C. Discussion

After the completion of experiment and implementing SA for constructing even-sized Latin squares for various values of $n$ in the range $[4, 20]$ with multiple runs we have recorded the best Latin square obtained.



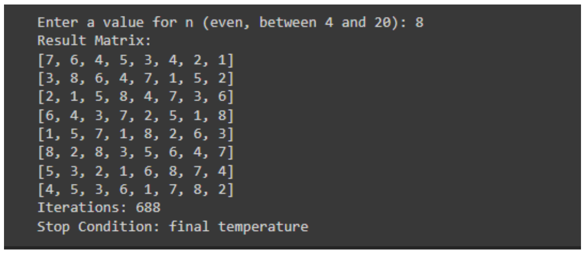
```
Enter a value for n (even, between 4 and 20): 4
Result Matrix:
[1, 4, 2, 3]
[4, 3, 1, 2]
[3, 2, 4, 1]
[2, 1, 3, 4]
Iterations: 132
Stop Condition: solution
```

Fig. 1. Solution achieved by SA for $(n)$ = 4.



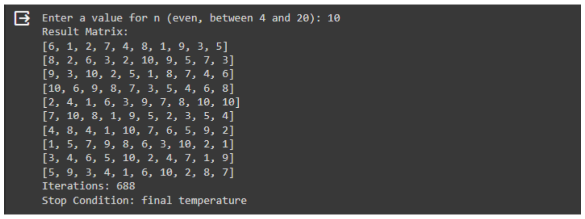
```
Enter a value for n (even, between 4 and 20): 6
Result Matrix:
[5, 6, 4, 2, 1, 3]
[2, 3, 6, 1, 4, 5]
[4, 1, 5, 3, 6, 2]
[1, 4, 2, 5, 3, 6]
[3, 5, 1, 6, 2, 4]
[6, 2, 3, 4, 5, 1]
Iterations: 136
Stop Condition: solution
```

Fig. 2. Solution achieved by SA for $(n)$ = 6.



```
Enter a value for n (even, between 4 and 20): 8
Result Matrix:
[7, 6, 4, 5, 3, 4, 2, 1]
[3, 8, 6, 4, 7, 1, 5, 2]
[2, 1, 5, 8, 4, 7, 3, 6]
[6, 4, 3, 7, 2, 5, 1, 8]
[1, 5, 7, 1, 8, 2, 6, 3]
[8, 2, 8, 3, 5, 6, 4, 7]
[5, 3, 2, 1, 6, 8, 7, 4]
[4, 5, 3, 6, 1, 7, 8, 2]
Iterations: 688
Stop Condition: final temperature
```

Fig. 3. Final Temperature found by SA for $(n)$ = 8.



```
Enter a value for n (even, between 4 and 20): 10
Result Matrix:
[6, 1, 2, 7, 4, 8, 1, 9, 3, 5]
[8, 2, 6, 3, 2, 10, 9, 5, 7, 3]
[9, 3, 10, 2, 5, 1, 8, 7, 4, 6]
[10, 6, 9, 8, 7, 3, 5, 4, 6, 8]
[2, 4, 1, 6, 3, 9, 7, 8, 10, 10]
[7, 10, 8, 1, 9, 5, 2, 3, 5, 4]
[4, 8, 4, 1, 10, 7, 6, 5, 9, 2]
[1, 5, 7, 9, 8, 6, 3, 10, 2, 1]
[3, 4, 6, 5, 10, 2, 4, 7, 1, 9]
[5, 9, 3, 4, 1, 6, 10, 2, 8, 7]
Iterations: 688
Stop Condition: final temperature
```

Fig. 4. Final Temperature found by SA for $(n)$ = 10.

. These records and Figures indicate the success rate and consistency of SA finding optimal solutions. Also, as $n$ increases, the success rate decreases, which might not result in optimal solution. Initial temperature and freezing factor has evidently impacted SA's performance.[1]

## V. Conclusions

[4]This section of report concludes the study we conducted by applying Simulated Annealing (SA) to the Latin Square problem. The following are the conclusions drawn on the basis of study performed:

### A. *Purpose of Experiment:*

The experiment was performed for construction of even-sized Latin Square by applying Simulated Annealing. During this experimentation we could demonstrate and conclude that the algorithm could consistently produce high-quality optimal solutions for smaller Latin Squares (e.g., $n = 4$). But its success rate decreased as $n$ increased.

### B. *Appropriateness of Proposed Values:*

The values of initial temperature ($T_i$) and cooling factor ($\alpha$) have evidently impacted SA's performance.

### C. *Cost Function's Guidance:*

The cost function played the most important role in guiding SA's search. It has effectively penalized solutions for Latin Square violations.

### D. *Effect of Increasing value of $n$:*

As the value of $n$ increased, there would be difficulty in finding optimal solutions, as the success rate decreased. This indicates that for large Latin Squares, SA struggled in finding optimal solutions for a limited number of runs.

The findings of this study provide valuable insights into the application of SA to combinatorial optimization problems like the Latin Square. Further research is warranted to explore parameter tuning strategies and enhanced cost functions to improve SA's performance on larger Latin Squares.

## References

[1] JR Elliott and Peter B Gibbons. The construction of subsquare free latin squares by simulated annealing. *Australas. J Comb.*, 5:209–228, 1992.

[2] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[3] Peter JM Van Laarhoven, Emile HL Aarts, Peter JM van Laarhoven, and Emile HL Aarts. *Simulated annealing*. Springer, 1987.

[4] Xin Yao. A new simulated annealing algorithm. *International Journal of Computer Mathematics*, 56(3-4):161–168, 1995.