# Parallel Genetic Algorithm to Optimize Travelling Salesman Problem

IT301 Course Project Mid-Semester Report

Saumya Karri
*181ME273*

Anshuman Sinha
*181EE209*

Sathvika B. Mahesh
*181CV141*

***Abstract***— The project aims at investigating the efficiency of the parallel computation on the Travelling Salesman Problem using Genetic Algorithm. Performance estimation and parallelism profiling will be attempted based on OpenMP-based parallel program techniques.

It is difficult to efficiently find the solution of TSP even with an enormous number of gene instances. Evolutionary approaches including the genetic algorithm, have been widely applied in research domains to tap into the enormous potential of TSP. In this project, we propose an improved constructive crossover for TSP. We will use the algorithm to obtain optimal and/or suboptimal solutions to the popular Traveling Salesman Problem. The proposed plan is to implement it using the Parallel Virtual Machine (PVM) library, which is based on a distributed-memory approach.

## INTRODUCTION

The **travelling salesman problem** is a combinatorial optimization problem that has vast applications. According to the problem, a salesman is required to sell his products by visiting a given set of cities. The distances between each pair of cities are provided. Therefore, to optimize his time, he should cover all the cities exactly once before returning to the original city by using the shortest possible route. This can be expressed in Computer Science/ Graph Theory terms where we have to find the lowest cost of a complete path that crosses all the nodes exactly once and returns to the starting node.

**TSP Algorithm:**

1. Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point.
2. Generate all (n-1)! permutations of cities.
3. Calculate the cost of every permutation and keep track of the minimum cost permutation.
4. Return the permutation with minimum cost.

**Some Applications of TSP:**

1. **Drilling of printed circuit boards:** To connect conductors of different layers, or to position the pins of integrated circuits, holes of different sizes have to drilled on the board. This can be viewed as a TSP, the aim being to minimize the travel time for the drilling machine head.
2. **Overhauling gas turbine engines:** To ensure uniform gas flow through turbines, guide vanes are provided. The problem of placing the vanes in the best possible position to get substantial benefits can be treated as a TSP.
3. **X-Ray crystallography:** An X-ray diffractometer is used to obtain information about the structure of the crystalline material. Thousands of positions of detectors are needed to get results for some experiments. To minimize the total time needed for the experiment, this is treated as a TSP.
4. **Order picking problem in warehouses:** A vehicle has to collect an order of items from a warehouse and ship it to the required destinations in least possible time. This is a classic application of TSP.

5. **Mailbox problem:** In a city, say 'n' mailboxes have to be emptied every day in a certain period of time. The problem is to find the minimum number of trucks to do this and to find the shortest time to do the collections.

**Genetic algorithms** are algorithms, which use concepts of evolutionary biology like recombination, inheritance, mutation, crossover etc. The algorithm replicates Darwin's process of natural selection to carry generation, i.e., survival of the fittest. The main goal of genetic algorithms is to search a solution space imitating the biological metaphor and, therefore, they are inherently parallel. Crossover and mutation play a significant role here.

The algorithm is divided into five stages primarily. Firstly, an initial population is generated. Next, overall fitness of individuals in the population is evaluated. From the fitness values of individuals, more favorable ones are selected for forming the next generation. After selection operation, crossover between selected individuals is performed to produce the next generation. The ultimate step is to mutate individuals with a given probability for diversity of search traits. If the resulting generation has a solution which is optimal, then the algorithm is terminated. Otherwise, fitness evaluation, selection, crossover, and mutation operators are applied on the next generation iteratively.

**Why the Genetic Algorithm?**

As mentioned above, this algorithm is designed to replicate the natural selection process, i.e., survival of the fittest. It is usually implemented to solve various optimization problems. In our proposed implementation: cities are analogous to genes, string generated using cities is called a chromosome, while a fitness score equal to the path length of all the cities can be used to target a population.

**Relevance of Parallel Execution**

The most time-consuming step in the algorithm is the fitness evaluation. If the population size is exceptionally large, fitness evaluation is very costly when done using sequential genetic algorithm. Another problem encountered in a sequential genetic algorithm is that it sometimes gets stuck in the zone of a local minima. To overcome the shortcomings mentioned above, Parallel Genetic Algorithm (PGA) is used.

**Parallel Genetic Algorithms (PGA)**

We could find three types of PGAs so far - **Master-slave, multiple populations with migration and multiple populations without migration**. We will be choosing one of these PGAs for our project after weighing the pros and cons of each.

**Genetic Algorithm:**

```
1. Initialize the population randomly.
2. Determine the fitness of the chromosome.
3. Until done repeat:
    (a) Select parents.
    (b) Perform crossover and mutation.
    (c) Calculate the fitness of the new population.
    (d) Append it to the gene pool.
```

## LITERATURE SURVEY – RELATED WORK

[1] Kang, S., Kim, S.S., Won, J., and Kang, Y.M., 2016. GPU-based parallel genetic approach to large-scale travelling salesman problem. *The Journal of Supercomputing*, *72*(11), pp.4399-4414 - It is difficult to efficiently solve TSP even with the large number of gene instances. This paper proposes an improved constructive crossover for TSP with which large number of genes can evolve by exploiting the parallel computing power of GPUs and an effective parallel approach to genetic TSP where crossover methods cannot be easily implemented in parallel fashion.

[2] Sena, G.A., Megherbi, D. and Isern, G., 2001. Implementation of a parallel genetic algorithm on a cluster of workstations: traveling salesman problem, a case study. *Future Generation Computer Systems*, *17*(4), pp.477-488 - The proposed algorithm is implemented using the Parallel Virtual Machine (PVM) library over a network of workstations. A master–slave paradigm is used to implement the parallel/distributed Genetic Algorithm (PDGA), which is based on a distributed-memory approach.

[3] Borovska, P., 2006, June. Solving the travelling salesman problem in parallel by genetic algorithm on multicomputer cluster. In *Int. Conf. on Computer Systems and Technologies* (pp. 1-6) - This paper investigates the efficiency of parallel computation of TSP using the genetic approach on a slack multicomputer cluster. For the parallel algorithm design functional decomposition of the parallel application has been made and the manager-workers paradigm is applied. Performance estimation and parallelism profiling have been made on the basis of MPI-based parallel program implementation.

[4] Harun Raşit Er, Dr. Nadia Erdoğan, March 2013, Parallel Genetic Algorithm to Solve Traveling Salesman Problem on MapReduce Framework using Hadoop Cluster, *Doi: 10.7321/jscse.v3.n3* - This paper shows a parallel genetic algorithm implementation on MapReduce framework in order to solve TSP. MapReduce is a framework used to support distributed computation on clusters of computers.

[5] Geeks for geeks - https://www.geeksforgeeks.org/genetic-algorithms/

[6] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal (2010), Travelling Salesman Problem, An overview of Applications, Formulations, and Solution Approaches, Prof. Donald Devendra (Ed.), ISBN: 978-953-307-426-9, In Tech

## METHODOLOGY

### Pseudo Code for TSP

In this algorithm, we take a subset N of the required cities that needs to be visited, Input taken is a) distance among the cities “dist” and b) starting city “s” as input. Each city is identified by unique city id like{1,2,3,4........n}. Initially, all cities are unvisited. We assume that the initial travelling cost is equal to 0. Next, the TSP distance value is calculated based on a recursive function. If the number of cities in the subset is two, then the recursive function returns their distance as a base case.

On the other hand, if the number of cities is greater than 2 then we’ll calculate the distance from the current city to the nearest city, and the minimum distance among the remaining cities is calculated recursively.

Finally, the algorithm returns the minimum distance as a TSP solution. Using recursive calls, we calculate the cost function for each subset of the original problem.

```
Algorithm 1: Dynamic Approach for TSP
  Data: $s$: starting point; $N$: a subset of input cities; $dist()$:
        distance among the cities
  Result: $Cost$ : TSP result
  $Visited[N] = 0$;
  $Cost = 0$;
  Procedure TSP($N$, $s$)
      $Visited[s] = 1$;
      if $|N| = 2$ and $k \neq s$ then
          $Cost(N, k) = dist(s, k)$;
          Return Cost;
      else
          for $j \in N$ do
              for $i \in N$ and $visited[i] = 0$ do
                  if $j \neq i$ and $j \neq s$ then
                      $Cost(N, j) = \min\ (\ TSP(N - \{i\}, j) + dist(j, i))$
                      $Visited[j] = 1$;
                  end
              end
          end
      end
      Return $Cost$;
  end
```

### Pseudo Code for GA

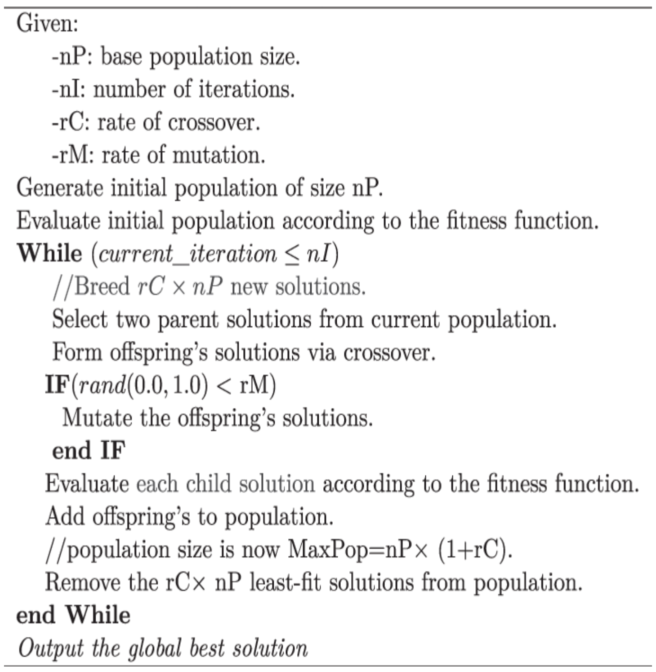
Once the initial generation is created, the algorithm evolve the generation using following operators

a) Selection Operator: preference is given to the individuals with good fitness scores and allow them to pass their genes to the successive generations for better offspring.

b) Crossover Operator: This represents mating between individuals. Two individuals selected using selection operator and crossover sites are chosen creating a completely new individual

c)Mutation Operator: The key idea is to insert genes in offspring to maintain the diversity in population.

Fitness of the offspring after all the operations is found, this process carried out repeatedly until the best optimal solution is found.



```
Given:
    -nP: base population size.
    -nI: number of iterations.
    -rC: rate of crossover.
    -rM: rate of mutation.
Generate initial population of size nP.
Evaluate initial population according to the fitness function.
**While** ($current\_iteration \leq nI$)
    //Breed $rC \times nP$ new solutions.
    Select two parent solutions from current population.
    Form offspring's solutions via crossover.
   **IF**($rand(0.0, 1.0) < \text{rM}$)
      Mutate the offspring's solutions.
    **end IF**
   Evaluate each child solution according to the fitness function.
   Add offspring's to population.
   //population size is now MaxPop=nP$\times$ (1+rC).
   Remove the rC$\times$ nP least-fit solutions from population.
**end While**
*Output the global best solution*
```

## INDIVIDUAL CONTRIBUTION

| | |
|---|---|
| **TSP framework using Graph Theory:** | Sathvika B. Mahesh |
| **Functions of Genetic Algorithm:** | Saumya Karri |
| **Parallelize TSP using GA using CUDA:** | Anshuman Sinha |