Genetic Vehicle Routing Problem

Cioltan Marian Alexandru
Tulba-Lecu Theodor-Gabriel

Asistent: Ouatu Andrei Catalin

Part 1 - Problem Description

Problem description

- Trucks have a set capacity

Problem description

- Trucks have a set capacity
- 2D problem with Euclidean coordinates and Euclidean distance

Problem description

- Trucks have a set capacity
- 2D problem with Euclidean coordinates and Euclidean distance
- All trucks start at the depot and must return to the depot at the end

Problem description

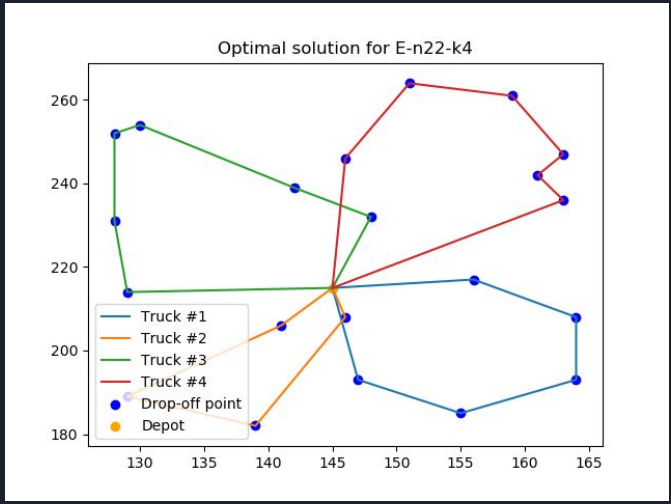
- Trucks have a set capacity
- 2D problem with Euclidean coordinates and Euclidean distance
- All trucks start at the depot and must return to the depot at the end
- Drop-off points require a set quantity to be satisfied

Problem description

- Trucks have a set capacity
- 2D problem with Euclidean coordinates and Euclidean distance
- All trucks start at the depot and must return to the depot at the end
- Drop-off points require a set quantity to be satisfied
- We want to minimize the total distance traveled by the trucks

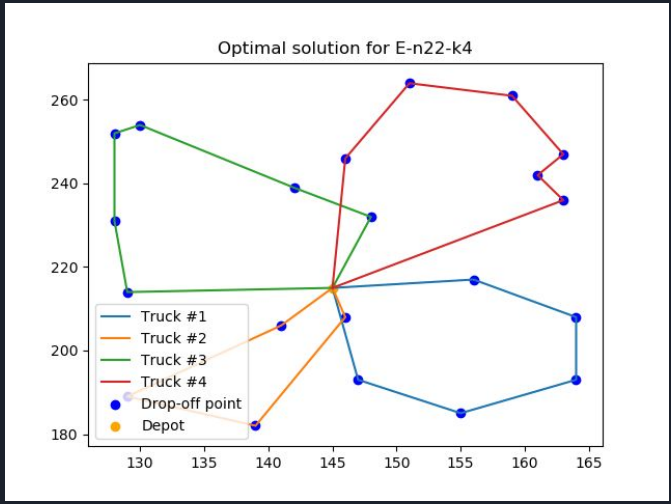
A visual example

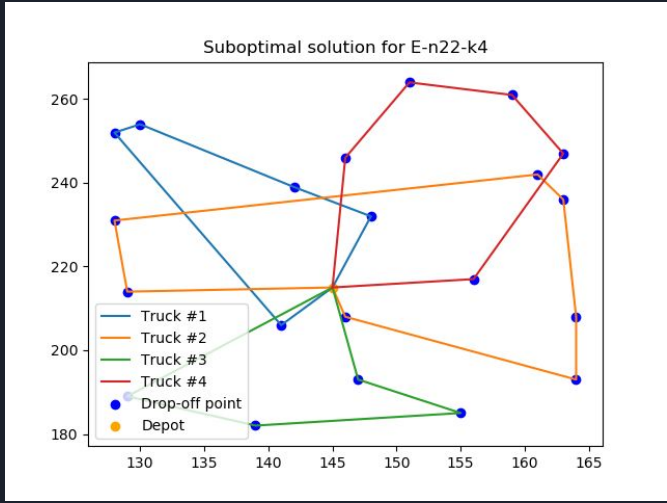
- Optimal length is 375



A visual example

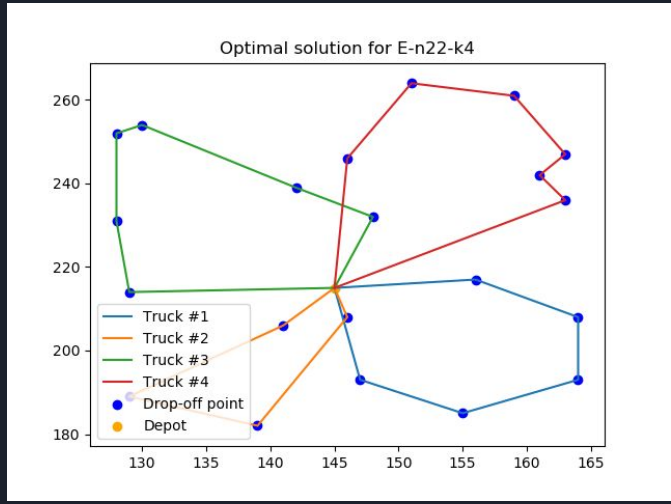
- Optimal length is 375
- Suboptimal solution has length 461.06

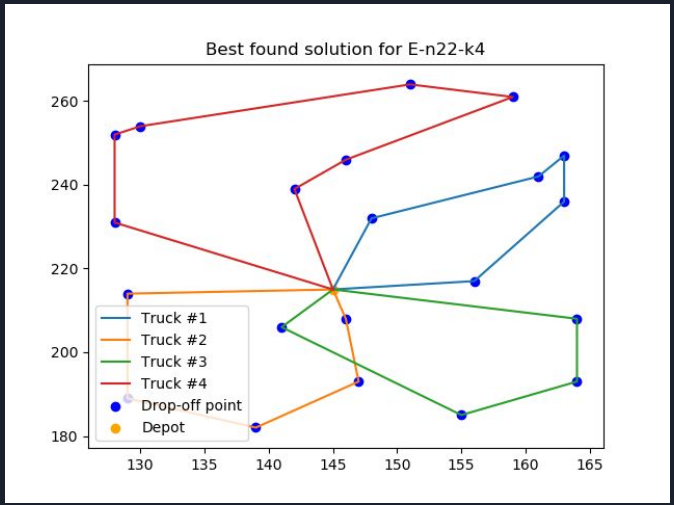


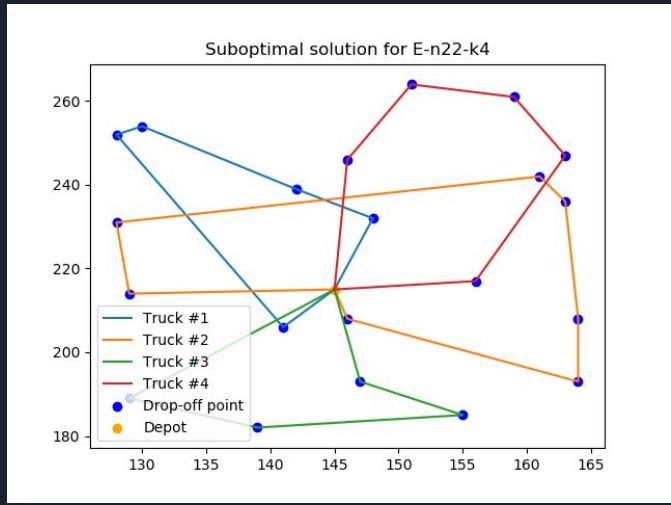


A visual example

- Optimal length is 375
- Suboptimal solution has length 461.06
- Best found solution has length 383.87

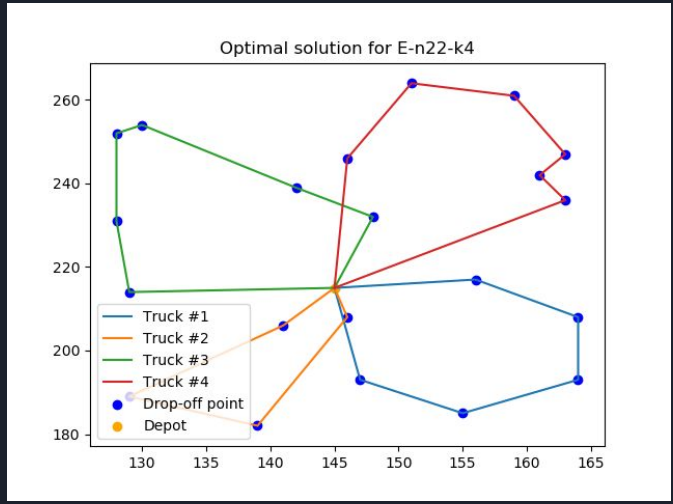


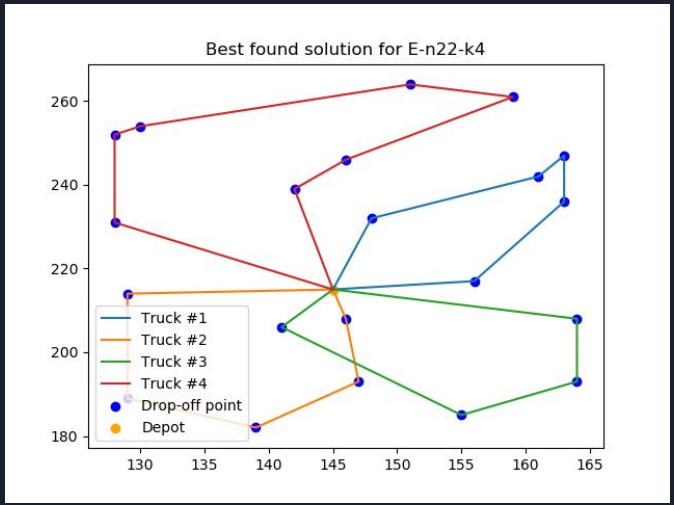


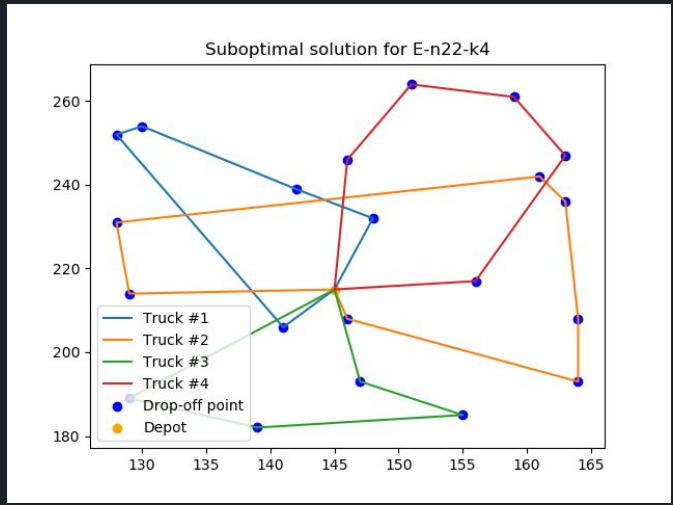


A visual example

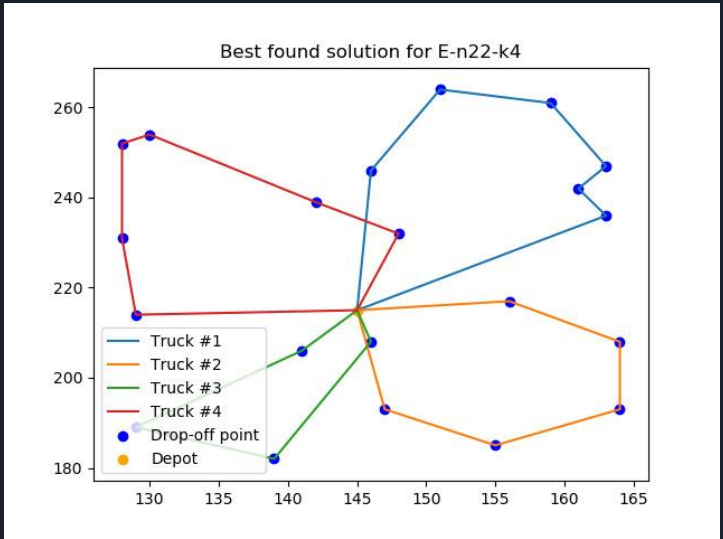
- Optimal length is 375
- Suboptimal solution has length 461.06
- Best found solution has length 383.87
- Even near-optimal solutions are very different

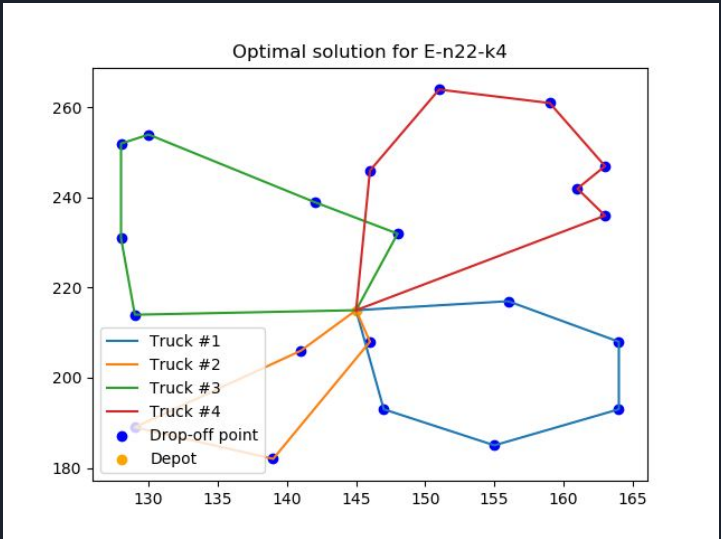






Best solution after enough generations





Part 2 - Genetic Algorithm

Genetic algorithm

- Individuals represented by genomes, similar to DNA.

Genetic algorithm

- Individuals represented by genomes, similar to DNA.
- Initially, individuals start with random genes, and evolve in time to find better solutions

Genetic algorithm

- Individuals represented by genomes, similar to DNA.
- Initially, individuals start with random genes, and evolve in time to find better solutions
- Population is represented by multiple individuals in a generation.

Genetic algorithm

- Individuals represented by genomes, similar to DNA.
- Initially, individuals start with random genes, and evolve in time to find better solutions
- Population is represented by multiple individuals in a generation.
- Evolution takes place over multiple generations.

Genetic algorithm

- Individuals represented by genomes, similar to DNA.
- Initially, individuals start with random genes, and evolve in time to find better solutions
- Population is represented by multiple individuals in a generation.
- Evolution takes place over multiple generations.
- Survival of the fittest: better performing individuals have a greater chance to reproduce and keep their genes in the gene pool.

Representing VRP Solution as a Genome

- Genome format

  3.353225, 3.374998, 1.496267, 1.425025, 3.482352, 1.611691, 3.629004, 1.875911, 3.834087, 3.005951, 2.134070, 2.846377, 2.427292, 2.993102, 2.495519

Representing VRP Solution as a Genome

- Genome format

  3.353225, 3.374998, 1.496267, 1.425025, 3.482352, 1.611691, 3.629004, 1.875911, 3.834087, 3.005951, 2.134070, 2.846377, 2.427292, 2.993102, 2.495519

- Each chromosome has the following format: {truck_number . order}

Representing VRP Solution as a Genome

- Genome format

  3.353225, 3.374998, 1.496267, 1.425025, 3.482352, 1.611691, 3.629004, 1.875911, 3.834087, 3.005951, 2.134070, 2.846377, 2.427292, 2.993102, 2.495519

- Each chromosome has the following format: {truck_number . order}
- A point is visited by a truck only after all points with smaller order are visited

Representing VRP Solution as a Genome

- Genome format

  3.353225, 3.374998, 1.496267, 1.425025, 3.482352, 1.611691, 3.629004, 1.875911, 3.834087, 3.005951, 2.134070, 2.846377, 2.427292, 2.993102, 2.495519

- Each chromosome has the following format: {truck_number . order}
- A point is visited by a truck only after all points with smaller order are visited
- Size of the genome is as big as the number of Drop-off points

Genetic parts - Fitness Calculator

- Fitness calculator
- Computes the total cost of all vehicle routes using a fitness function

Genetic parts - Fitness Calculator

- Fitness calculator

- Computes the total cost of all vehicle routes using a fitness function
- Fitness function greatly punishes trucks that are overloaded

Genetic parts - Fitness Calculator

- Fitness calculator

- Computes the total cost of all vehicle routes using a fitness function
- Fitness function greatly punishes trucks that are overloaded
- While encouraging trucks with shorter routes

Genetic parts - Selector

- Fitness calculator
- Selector

- Selects the pairs of individuals that will reproduce

Genetic parts - Selector

- Fitness calculator
- Selector

- Selects the pairs of individuals that will reproduce
- Sorts individuals and creates pairs of parents that create offspring

Genetic parts - Selector

- Fitness calculator
- Selector

- Selects the pairs of individuals that will reproduce
- Sorts individuals and creates pairs of parents that create offspring
- Elitism: A small fraction of the best performers also survive

Genetic parts - Selector

- Fitness calculator
- Selector

- Selects the pairs of individuals that will reproduce
- Sorts individuals and creates pairs of parents that create offspring
- Elitism: A small fraction of the best performers also survive
- Introduce new individuals with completely random genomes

Genetic parts - Crossover Calculator

- Fitness calculator
- Selector
- Crossover calculator

- Creates new individuals from the computed pair of parents

Genetic parts - Crossover Calculator

- Fitness calculator
- Selector
- Crossover calculator

- Creates new individuals from the computed pair of parents
- Uniform crossover: Each gene is selected randomly from one of the parent chromosomes.

Genetic parts - Mutator

- Fitness calculator
- Selector
- Crossover calculator
- Mutator

- Randomly mutates the genome with a small chance

Genetic parts - Mutator

- Fitness calculator
- Selector
- Crossover calculator
- Mutator

- Randomly mutates the genome with a small chance
- Trigger global mutations after a set number of generations, representing major environment changes

Part 3 - Locating the bottleneck

Bottleneck - Empirical Approach

- Manually analyzing the algorithm points out to the Fitness Calculator as being the bottleneck
- It has the biggest asymptotic complexity of O(num_cars * num_dropoff)
- Best candidate for parallelization
- This is the only part that is worth parallelizing

```
void VRP::LinearFitnessCalculator::calculateInterval(double *genome, int len, int genomeSize, vector<double> &fitness) {
    for (int i = 0; i < len; i++) {
        fitness[i] = calculate(genome + i * genomeSize, genomeSize);
    }
}
```

Bottleneck - Confirmation via Profiling

- Running Intel VTune Profiler for linear implementation confirms the previous claim
- Out of all the functions we've implemented it's the most time consuming by a huge margin
- It is trivial to parallelize
- Next function in the profiling is the Crossover which takes only 3.7% of total CPU time

Hotspots

Analysis Configuration | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Flame Graph | Plat

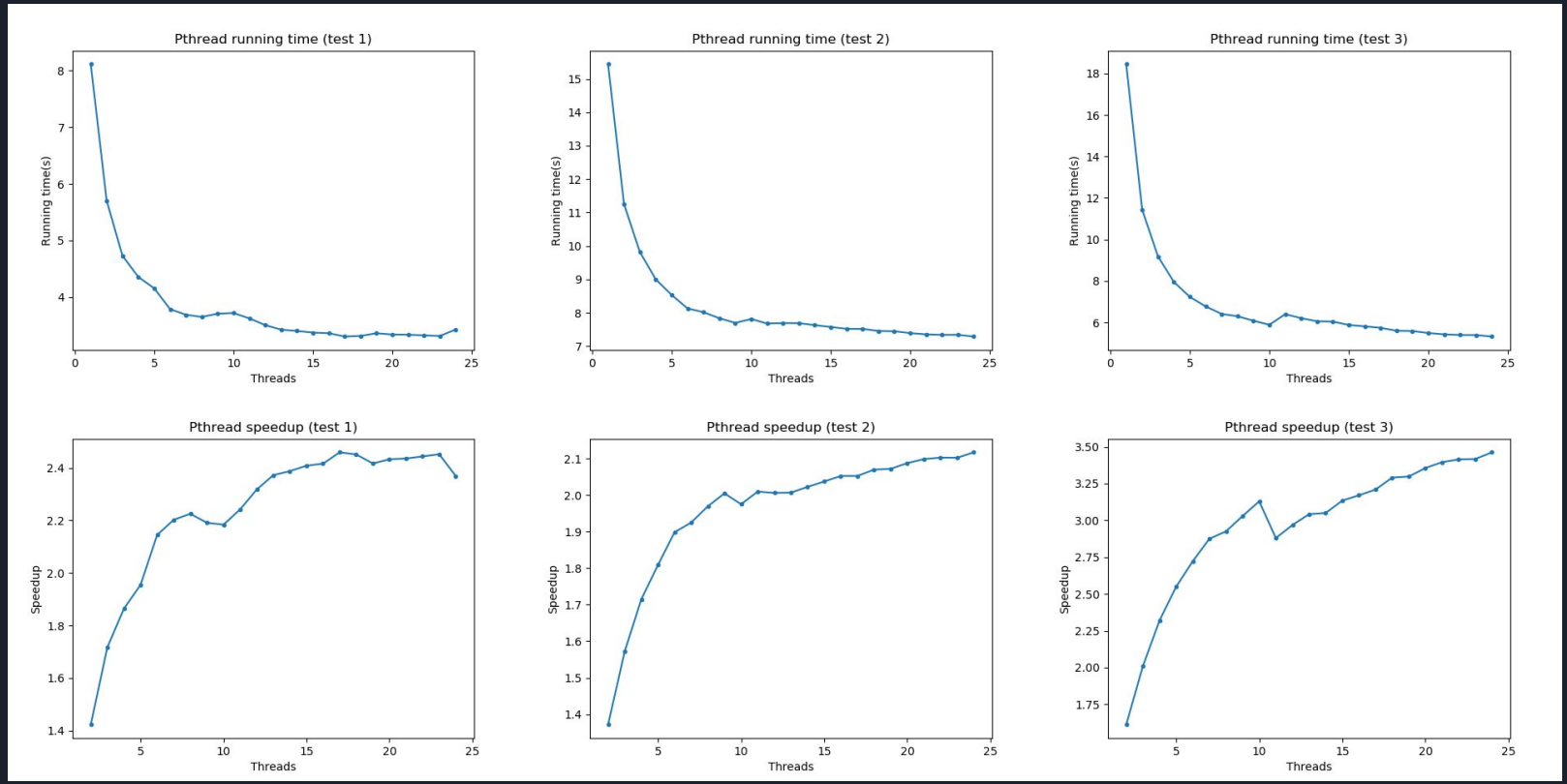
Function	CPU Time: Total	CPU Time: Self		
		Effective Time ▼	Spin Time	Overhead Time
std::vector<std::pair<double, int>, std::alloca	40.3%	35.002s	0s	0s
std::sort<__gnu_cxx::__normal_iterator<std::	17.2%	8.214s	0s	0s
VRP::LinearFitnessCalculator::calculate	86.4%	8.016s	0s	0s
__gnu_cxx::__normal_iterator<std::pair<doub	4.5%	5.528s	0s	0s
std::sqrt<int>	3.7%	4.610s	0s	0s
std::make_pair<double, int&>	3.4%	4.176s	0s	0s

Part 4 - Implementation comparison

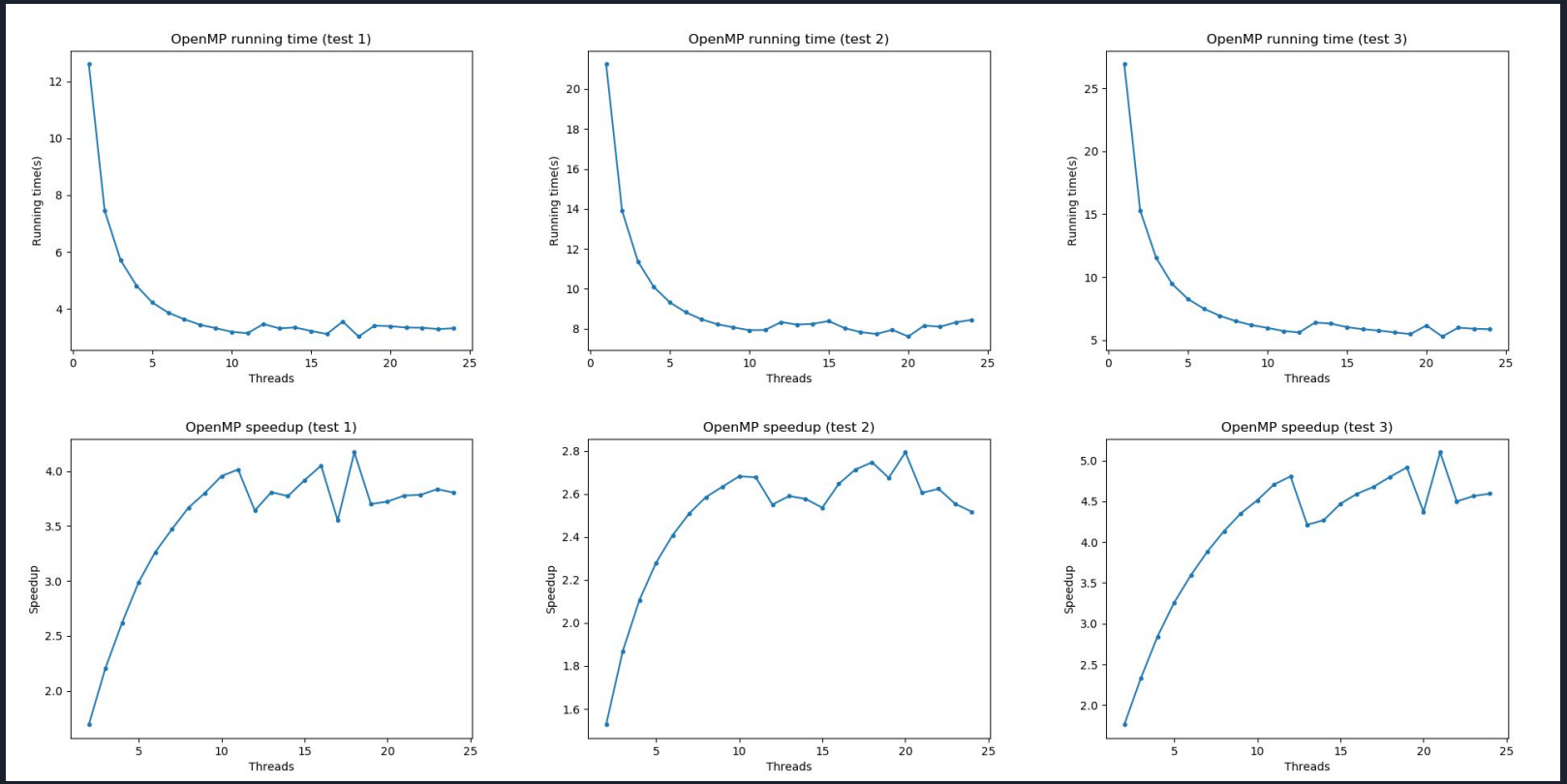
A starting point - Linear implementation

- For all the tests we chose the following parameters
  - Genome size - 400
  - Number of generations - 3000
- We ran our implementations on 3 different tests
- The times for the linear implementation are:
  - Test #1 - 12.705s
  - Test #2 - 21.237s
  - Test #3 - 26.891s

Pthread implementation results



OpenMP implementation results



MPI implementation results