Travelling Salesman Problem

Parallel Systems MOSIG M2

Outline

- Traveling Salesperson Problem (TSP)
- Genetic Algorithms (GA)
- TSP using GA
- Implementation
- Parallelization
- Evaluation
- Conclusion

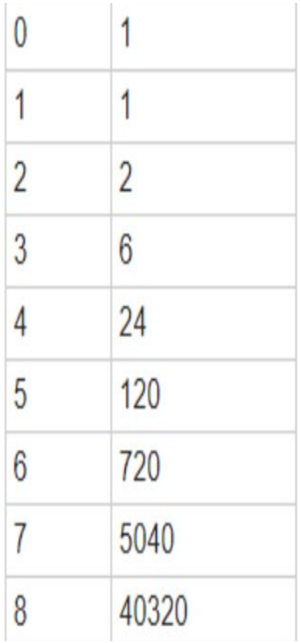
Traveling Salesperson Problem (TSP)

- Traveling Salesman Problem (TSP) is Challenge of finding the shortest route visiting each member of a collection of locations and returning to your starting point.
- E.g. Traveling Santa Problem ☺

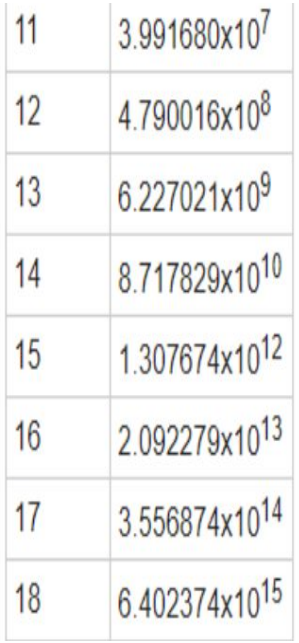
https://pressidium.com/blog/2016/travelling-santa-problem/

Traveling Salesperson Problem (TSP)

- Goal: to find minimal Hamilton cycle start at a given vertex, visit every other vertex exactly once, and ends at the same original starting vertex.
- The exact solution would be to try all permutations and choosing the cheapest one using brute-force search
- NP-Complete, O(n!)
- Impractical, even for 20 cities



0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320



11	3.991680×10^{7}
12	4.790016×10^{8}
13	6.227021×10^{9}
14	8.717829×10^{10}
15	1.307674×10^{12}
16	2.092279×10^{13}
17	3.556874×10^{14}
18	6.402374×10^{15}

TSP using Genetic Algorithm (GA)

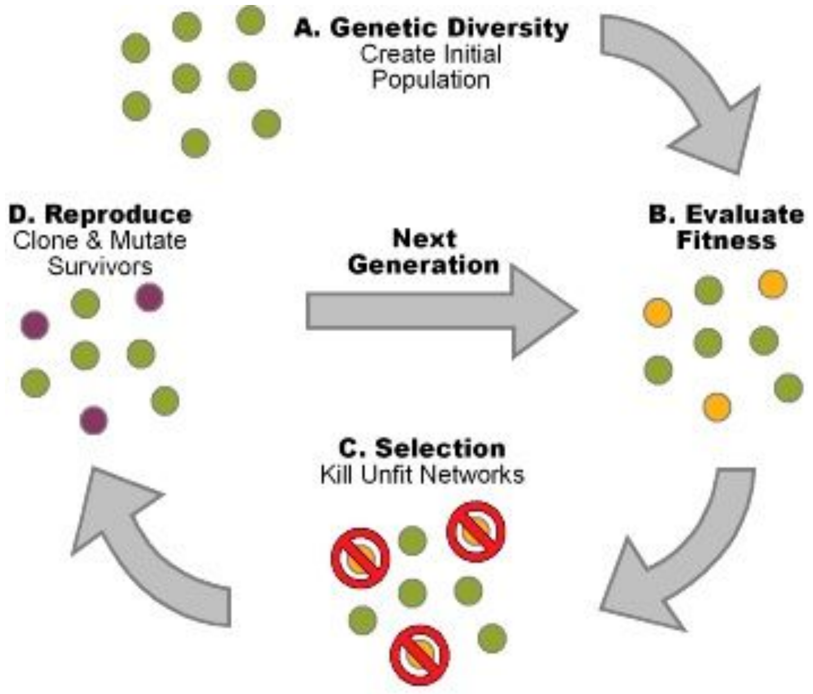
Main Idea

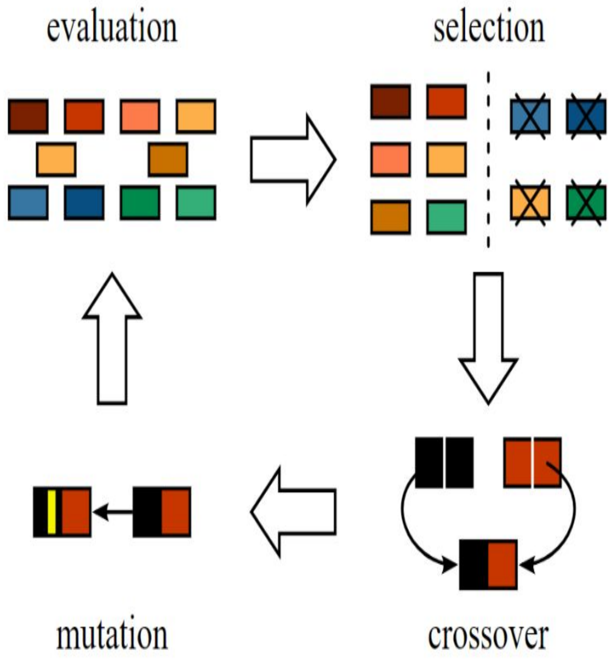
- Based on the evolution theory, i.e. survival of the fittest
- Practically: random exploration of the search space
- Well suited for large combinatorial optimization problems

Main Structure

- Candidate Solution => Chromosome, i.e. sequence of genes (cities)
- Population => (large) set of chromosomes
- Initial Generation => randomly generated population(s)
- Next generation => evolution of previous generation

TSP using Genetic Algorithm (GA)





Implementation

- Arguments parsing
- Reading input file (wi29.txt)
- Population initialization
- Fitness Calculation
- Loop
  - Selection
  - Crossover
  - Mutation
  - Evaluation of fitness
  - Replacement

Parallelization

Multiple populations on multiple nodes collecting in Master (node 0)

I choose this because it require:

- ➢ fewer communications than the master-slave
- ➢ Low communication to computation ratio

Aim: minimize total execution time

- ➢ Maximize CPUs/Cores utilization
- ➢ Minimize communication time

Parallelization

Main Idea:

- Each node initialize subset of the full population (N/P)
- process start to do the algorithm (selection - crossover - mutation - calculate fitness and sorting the population)
- Process stop once it reach max number of generations
- Process send back the population to the master node
- After master collect all the chunks from processes it sort and print the fittest path (fittest chromosome).
- OpenMP for parallel, mainly in calculating fitness and crossover creating new chromosome in case of hybrid version

Implementation

three versions:

- Sequential
- Pure MPI
- Hybrid (MPI,openMP) using collective communication (MPI_Gather)
- Hybrid (MPI,openMP) using normal MPI_Send/Recv

PS: the code is little bit big 600 lines but it is well commented, and easy to read.

For more information about implementation, you can check my report
https://docs.google.com/document/d/1yo4KQAXF_zGLQ6nUoBeSucCX_ez3ogAt1AWPQLc9DTI/edit#

Evaluation

1. We will study the performance between the 3 versions (sequential , Pure mpi, mpi_openMP) (mpi using 4 nodes). The study will be conducted on the Grid’5000 and on my personal laptop.
2. Also we will compare the performance of using mpi_Gather Vs using normal mpi_Send/Recv

Number of cities used is 29 (wi29.tsp input file)

I created a shell script to run the algorithm with different number of generations automatically (**run.sh**).

Evaluation

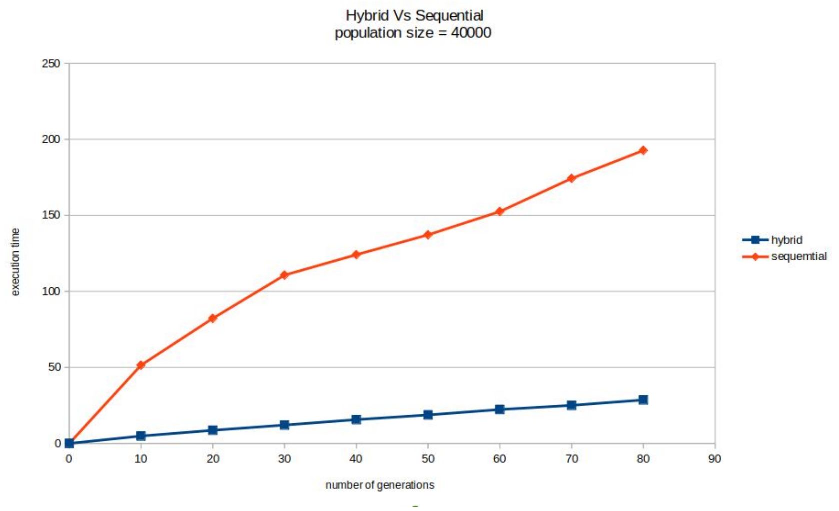
Critical parameters to change while evaluating:

1. Population size (number of chromosome per node)
2. Number of generations (how many generations to be done by each node)
3. Number of thread (in hybrid version)
4. Number of processes

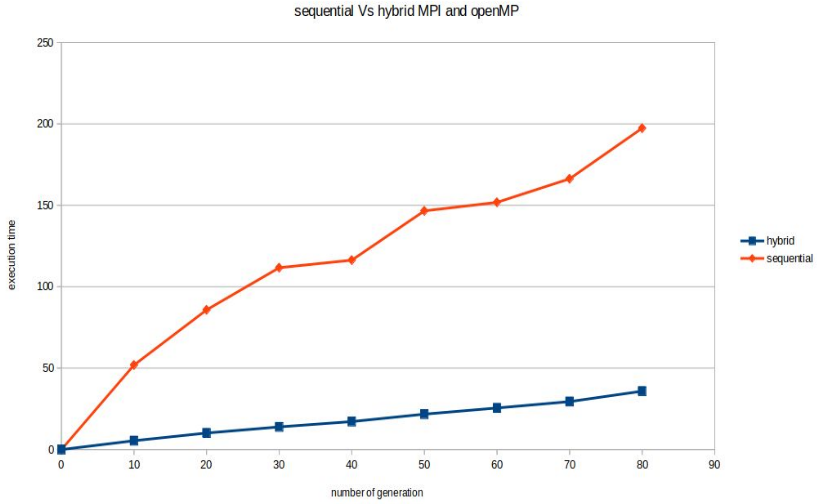
Evaluation

For both population size is 40000 sequential Vs Hybrid (4 processors)

Executed on
Grid'5000



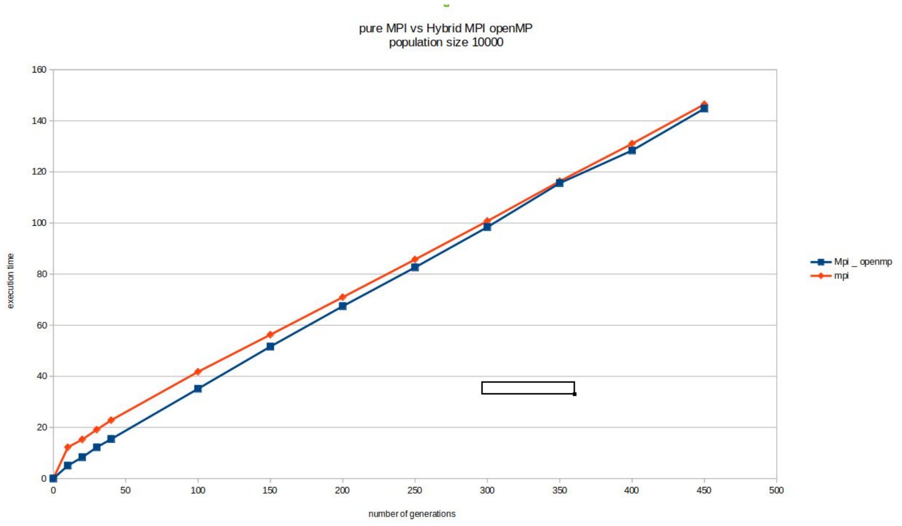
Executed on
personal machine



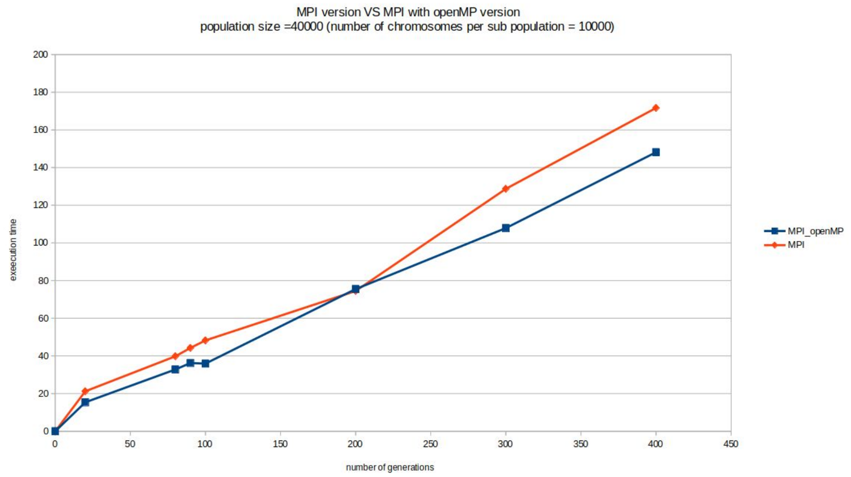
Evaluation

For both population size is 40000 Hybrid Vs pure MPI (4 processors)

Executed on
Grid'5000



Executed on
personal machine

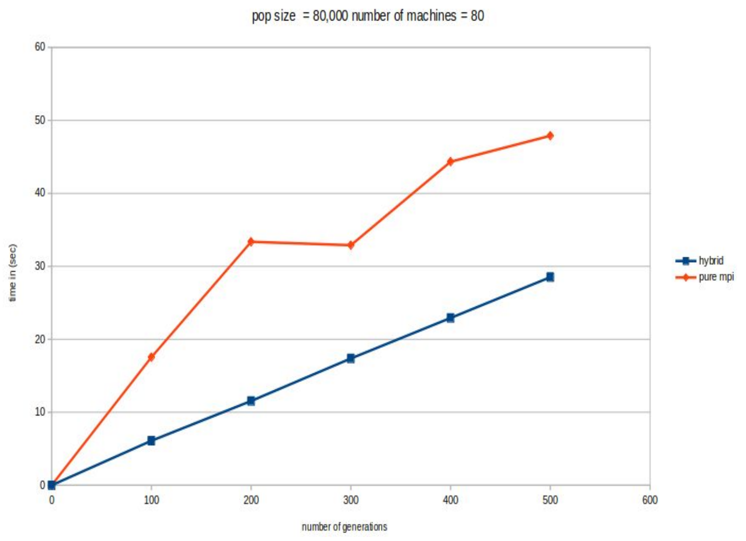


Evaluation

Population size is 80000 Hybrid Vs pure MPI (80 processors)

Executed on Grid’5000

80 processors because i wanted the program to use 3 different machines dahu-6 , 7 and 8 each machine with 32 cores.

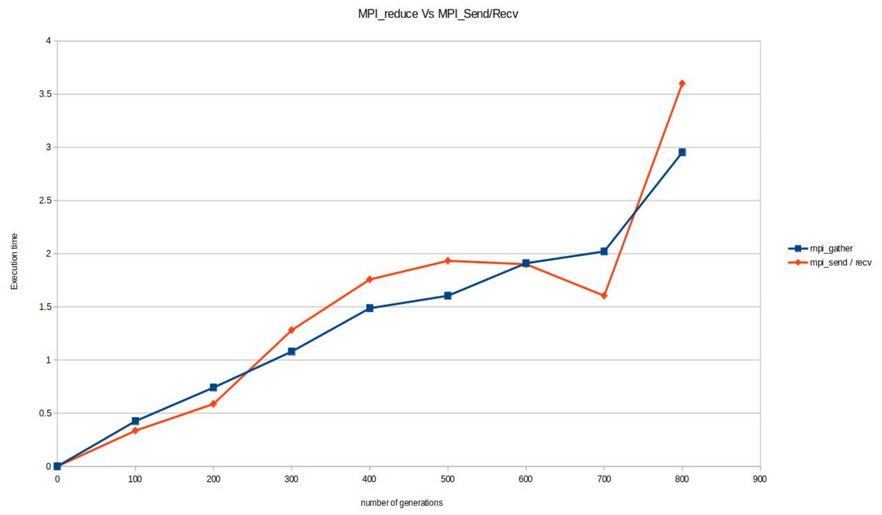


Evaluation

Population size is 40000 Hybrid Vs pure MPI (4 processors)

Executed on Grid’5000

In the figure, we can see that the mpi_reduce is slightly better than using the normal send and receive in mpi.



Evaluation Conclusion

So as we saw before the execution time of hybrid or pure mpi is way better than the sequential version, and it deserves to be done, while for hybrid vs pure mpi it wasn’t as I was expecting, I expected the hybrid to be way more efficient than the pure mpi while the results shows that it is slightly better anyway it always depend on the size of the problem (bigger better).
We studied performance in terms of time execution. In terms of most fittest path as we increase the number of generations and number of chromosomes per population we will be more confident about the best path (the fittest one).
With small population size Hybrid is not good in terms of execution time, because we are creating threads while number of chromosomes in the population is small.