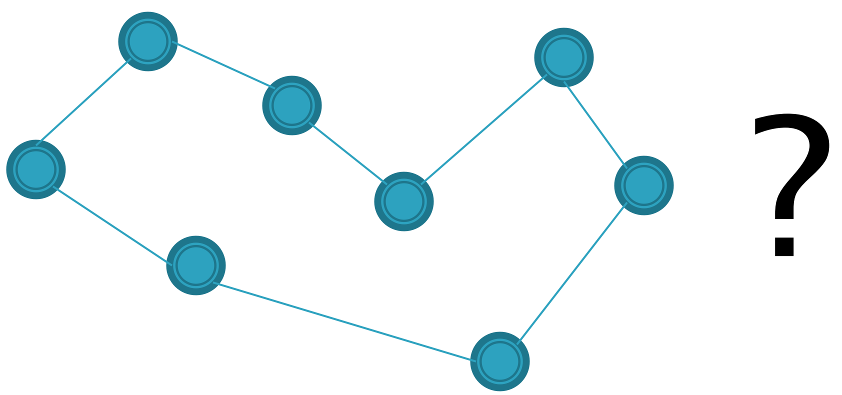
Massively Parallel Traveling Salesman Genetic Algorithm

Matt Heavner
mheavner@buffalo.edu
CSE710
Fall 2009

Traveling Salesman Problem

- Problem Statement: Given a set of cities and corresponding locations, what is the shortest closed circuit that visits all cities without loops?



Genetic Algorithms: Terminology

- Fitness Function: Function or routine to optimize
- Population: Current set of candidate solutions
- Chromosome: A specific candidate solution to optimization problem, usually encoded into a string of values
- Fitness: Fitness function output for a given chromosome

Genetic Algorithms: Pseudocode

```
generate initial population
evaluate fitness of population
while termination criteria not met:
  breed new population:
      apply elitism
      select two chromosomes from old pop:
            perform crossover?
            perform mutation?
  evaluate fitness of new population
```

Genetic Algorithms: Breeding – Elitism

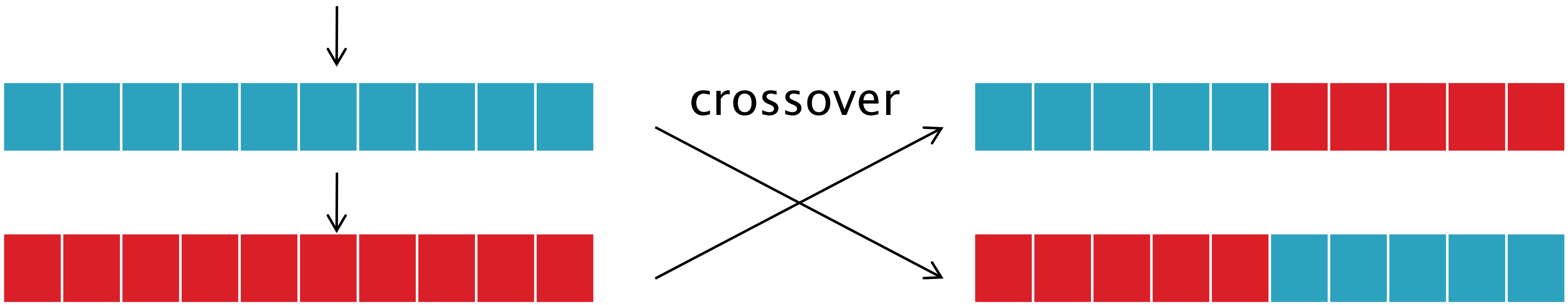
- Select a certain percentage, called the elitism percentage.
- When breeding new population, sort by fitness. Bring this percent of top performing solutions to new population.
- Ensures top performers won’t get lost.

Genetic Algorithms: Breeding – Selection

- When creating new population, need a way of selecting chromosomes from the old population for breeding. Various methods include:
  - Fitness-Proportionate
  - Tournament
  - Etc.

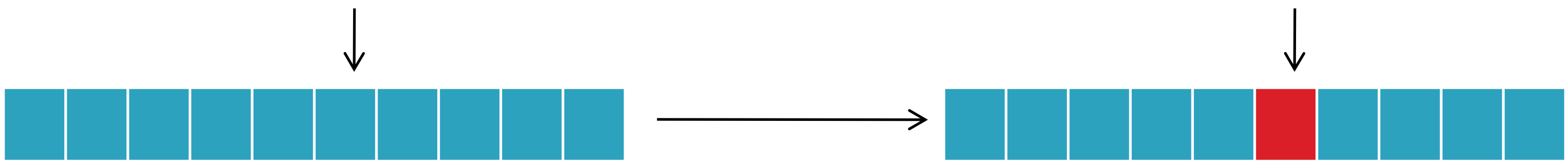
Genetic Algorithms: Breeding – Crossover

- Select crossover probability
- When two chromosomes are selected for breeding. If a random number meets this probability, crossover is performed
- Select a random crossover point
- Swap chromosome sections about this point



Genetic Algorithms: Breeding – Mutation

- Select a mutation probability
- For each new population member, select random number. If within probability mutate
- Point mutation:



- Swap mutation:

Genetic Algorithm for TSP

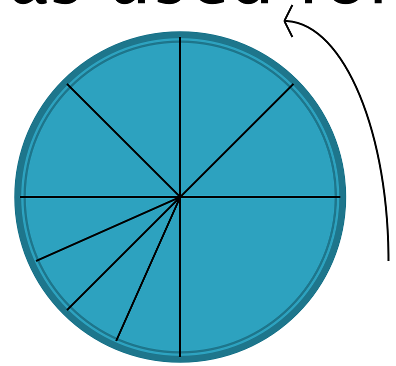
- Chromosome: Candidate permutation of ordered city visits, no repeats. Stored as a sequence is city indices corresponding to a lookup table

1	6	3	8	4	7	9	2	5	0

- Fitness: 1/(total Euclidean distance of circuit)
- Optimization: maximum fitness == chromosome with smallest closed non-looping circuit

Genetic Algorithm for TSP: Selection

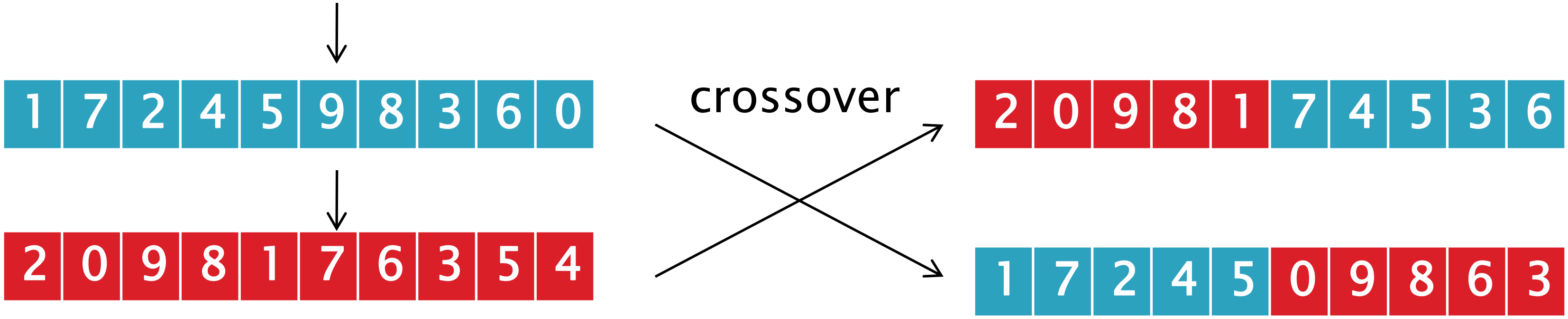
- Roulette Wheel Selection was used for this problem.



- Roulette Wheel Selection
  - Probability of a chromosome being selected is dependent on its fitness
  - Rank by fitness and normalize. Choose random number in this range and iterate through ranked chromosomes, summing fitness values, until this random number is reached. Pick corresponding member.

Genetic Algorithm for TSP: Crossover

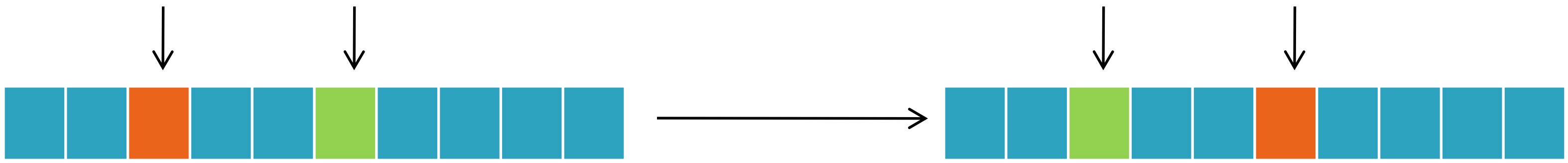
- Used modified one-point crossover
  - Randomly select swap point as before and swap.
  - Iterate through elements in old chromosome and fill in the missing elements in order
  - Necessary to preserve uniqueness of city visits



Genetic Algorithm for TSP: Mutation

- Rather than point mutation, swap mutation was used to ensure uniqueness of locations

- Swap mutation:



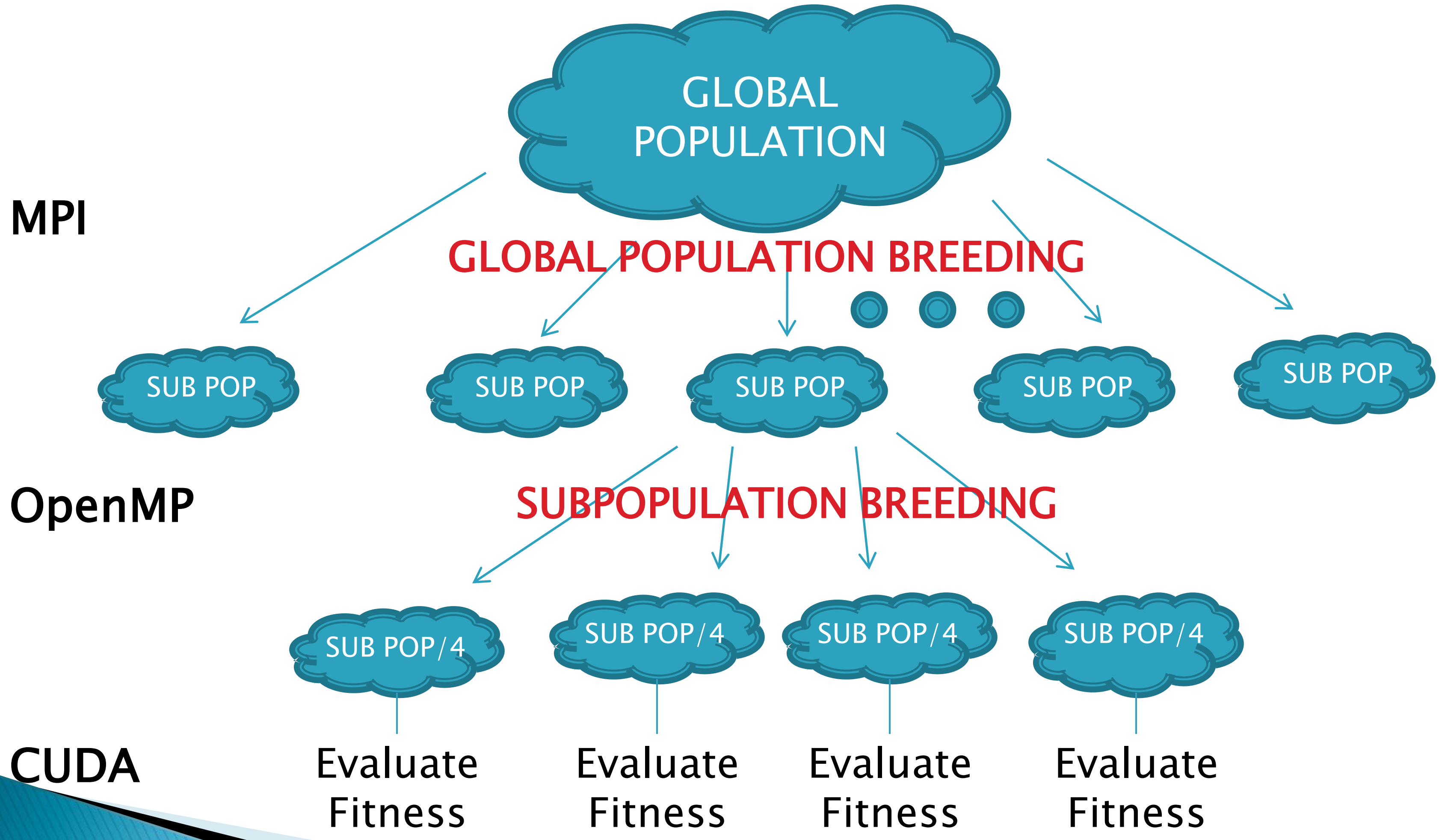
Parallelization of Algorithm

- Split global population into subpopulations – one for each node.
- On each node, split subpopulation into 4. For each of these groups use CUDA to calculate fitness and create new population using sequential method. Do this until a fixed number of sub-iterations has completed.
- Once sub-iterations have completed, recombine at a global level, redistribute and repeat until global iterations are finished.

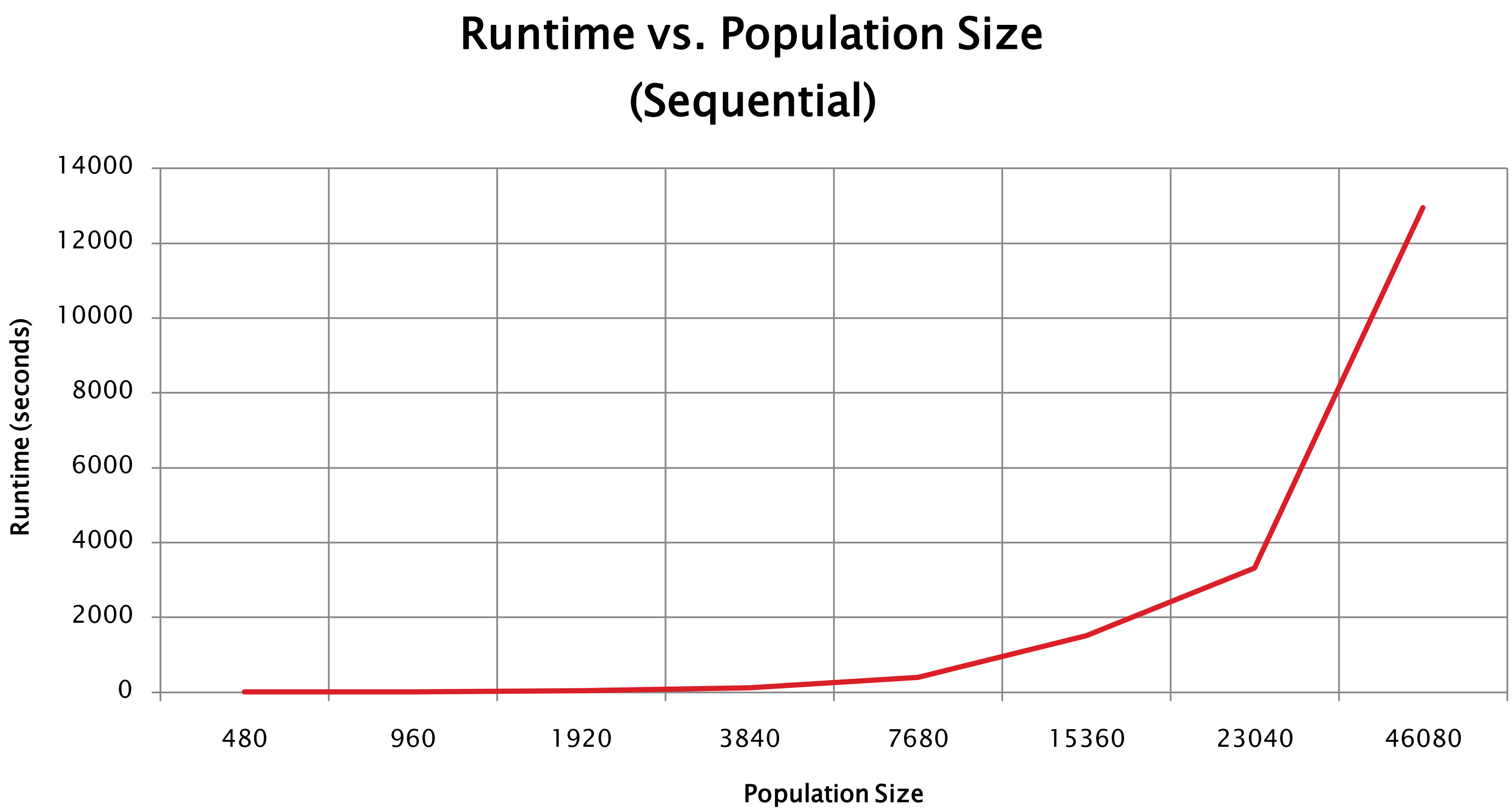
Parallelization of Algorithm: Pseudocode

```
glob_iters = 0
while glob_iters != MAX_GLOB_ITERS:
    distribute global population via MPI
    sub_iters = 0
    while sub_iters != MAX_SUB_ITERS:
        split sub-population into 4
        calculate fitness of each sub-population via CUDA
        breed new sub-population
        sub_iters++
    gather sub-populations via MPI
    breed new global population
    glob_iters++
```

Parallelization of Algorithm: Population Distribution

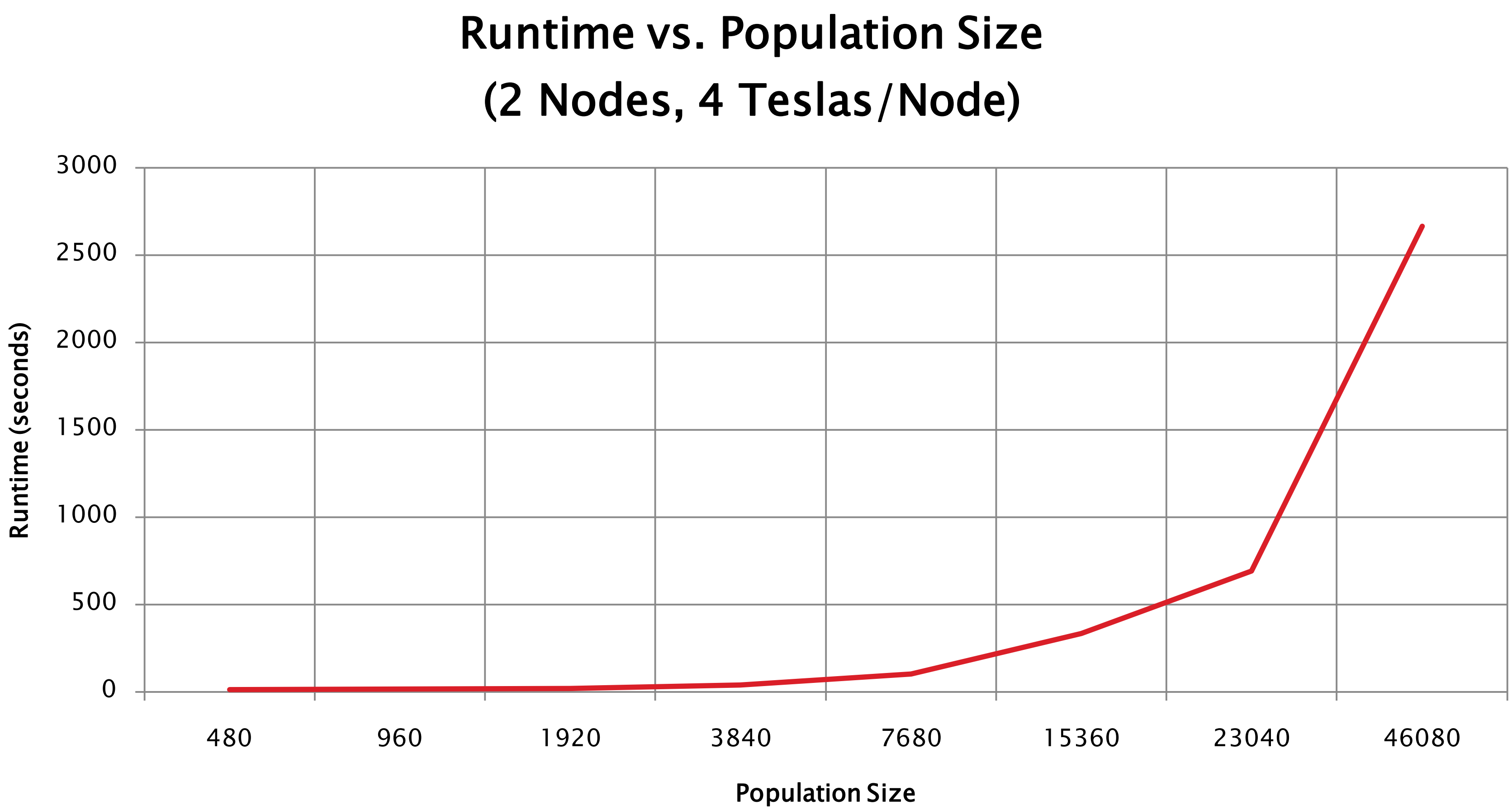


Sequential Timing Results

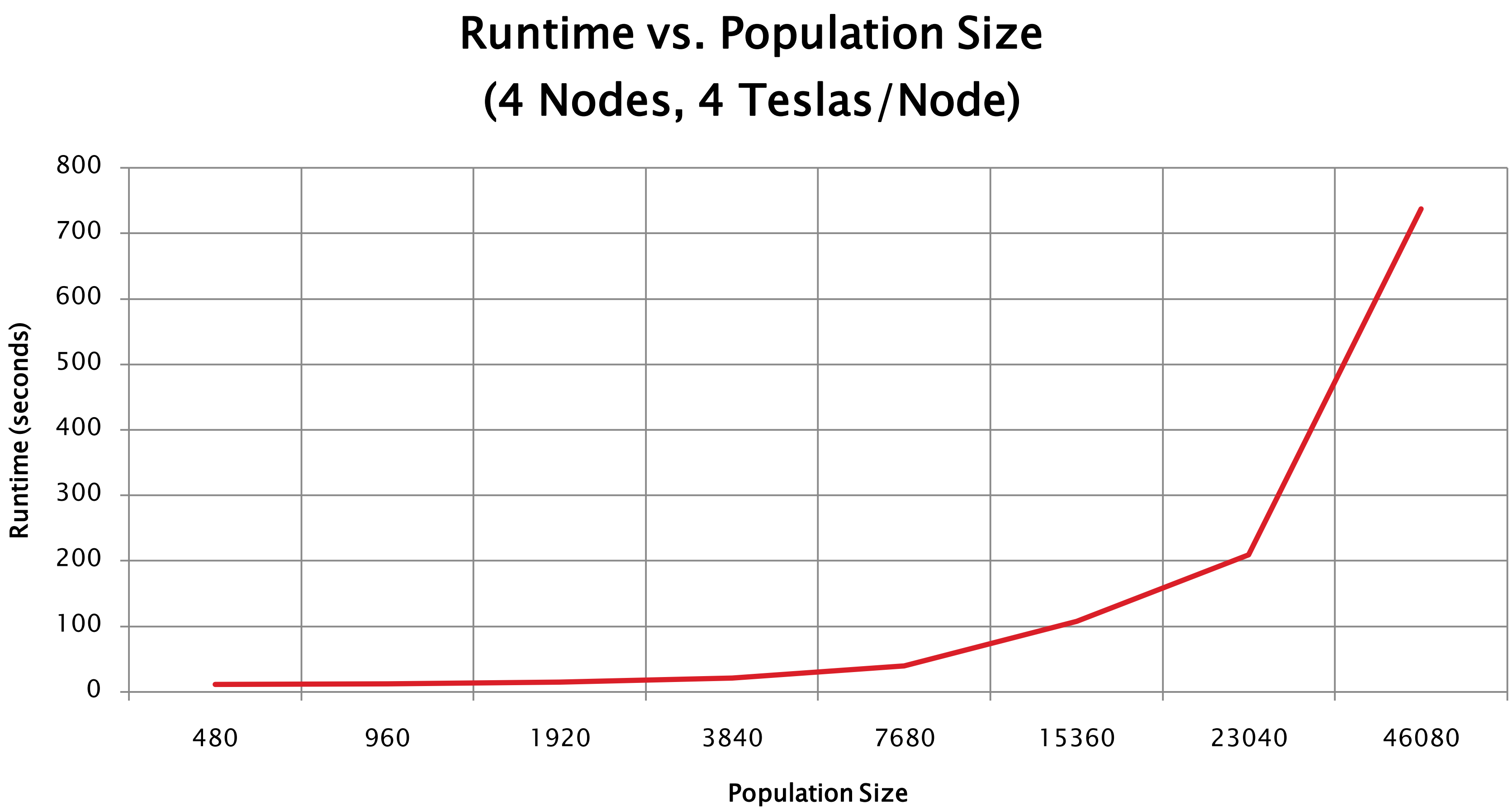


Platform: Intel(R) Xeon(R) CPU E5430 @ 2.66GHz (same as worker nodes 9-13)

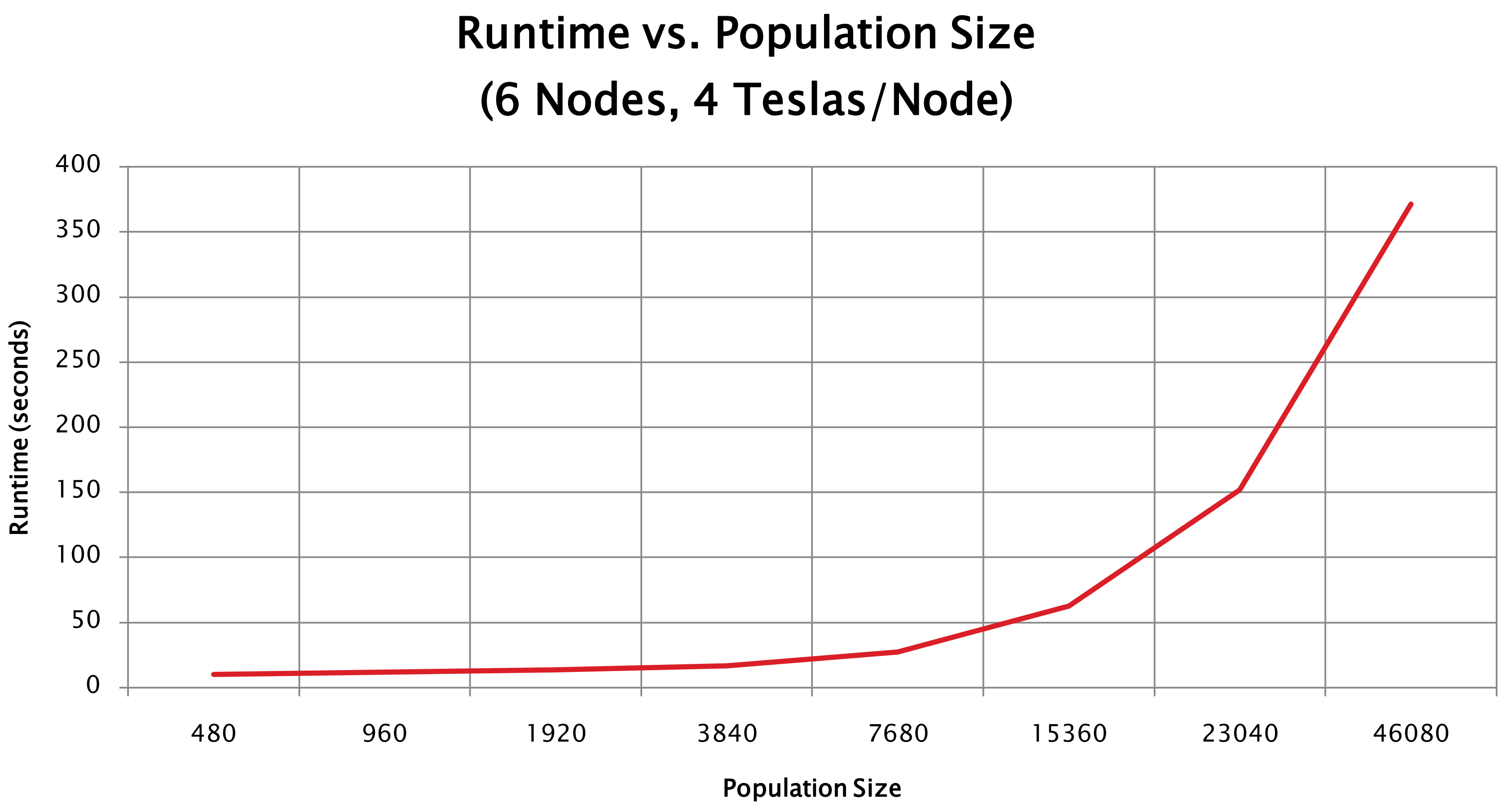
2-Node Timing Results



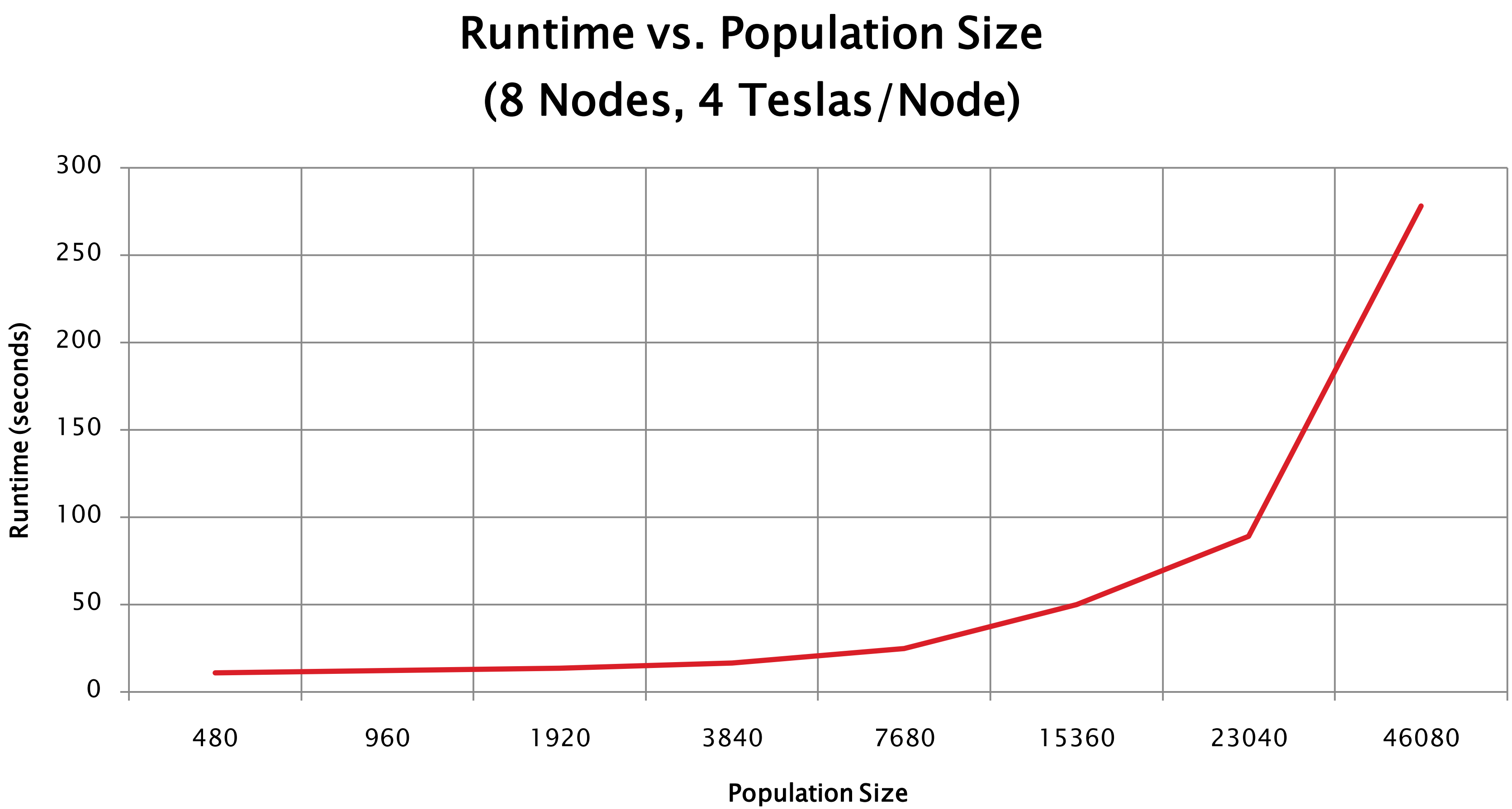
4-Node Timing Results



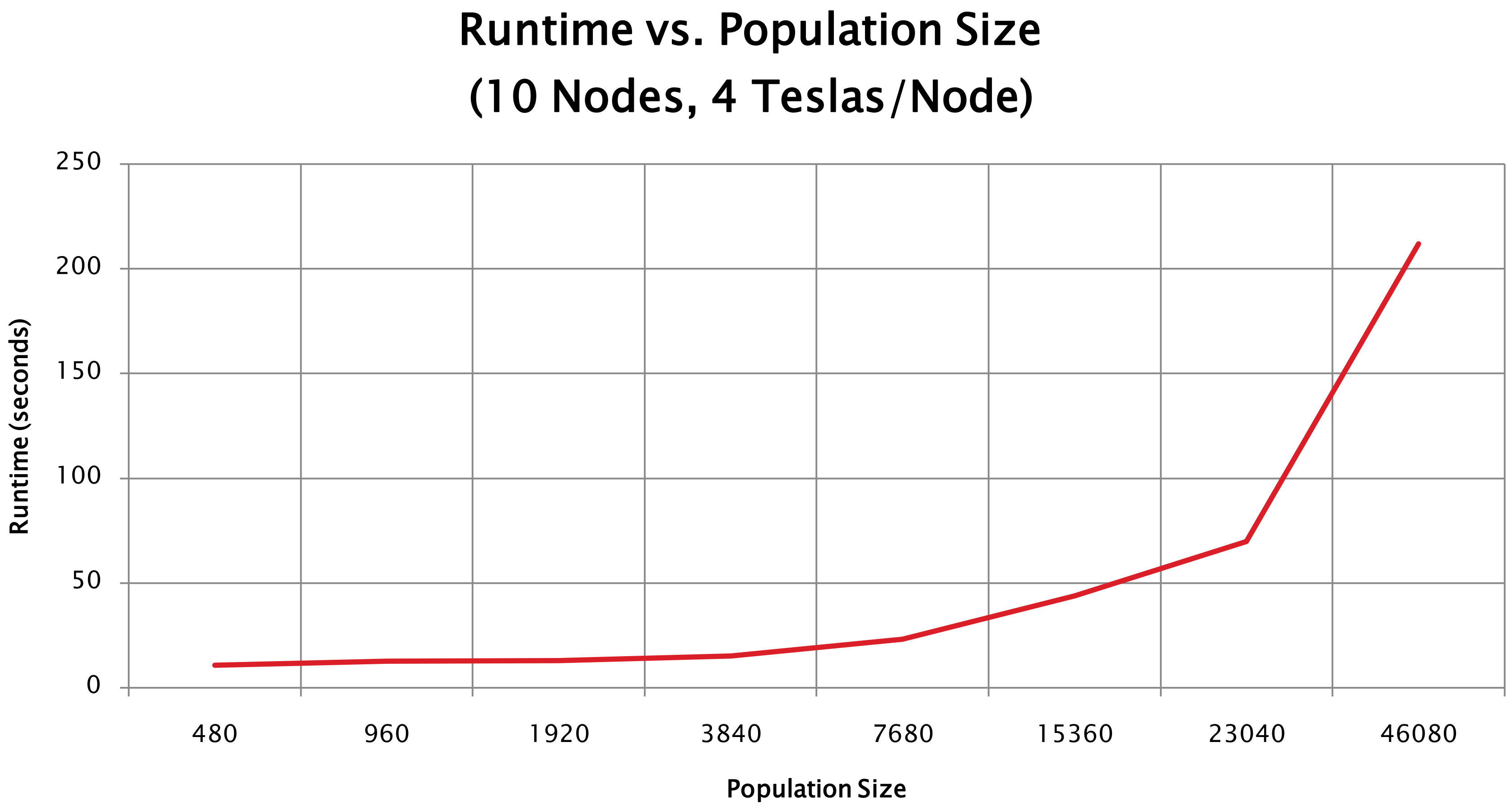
6-Node Timing Results



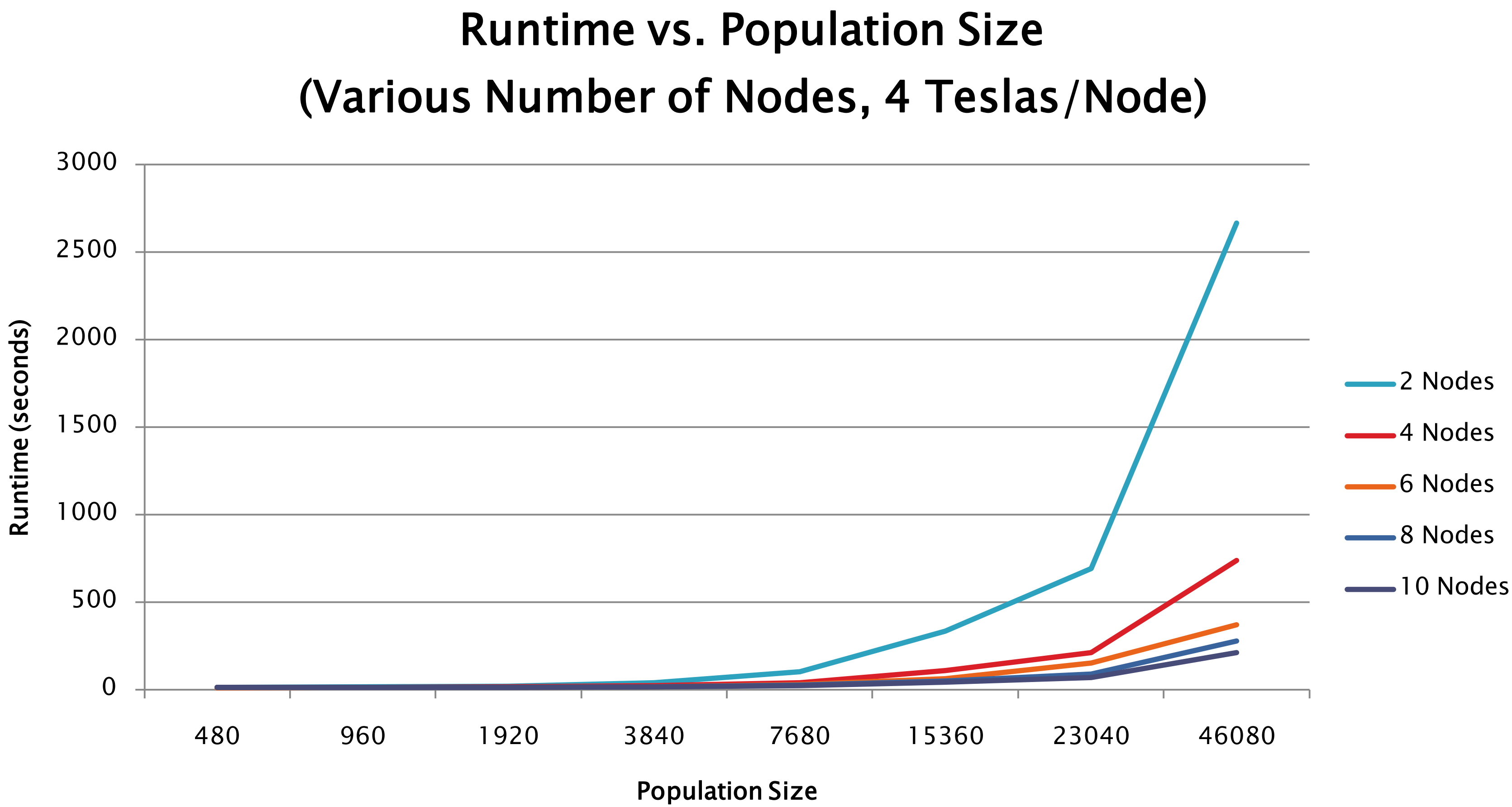
8-Node Timing Results



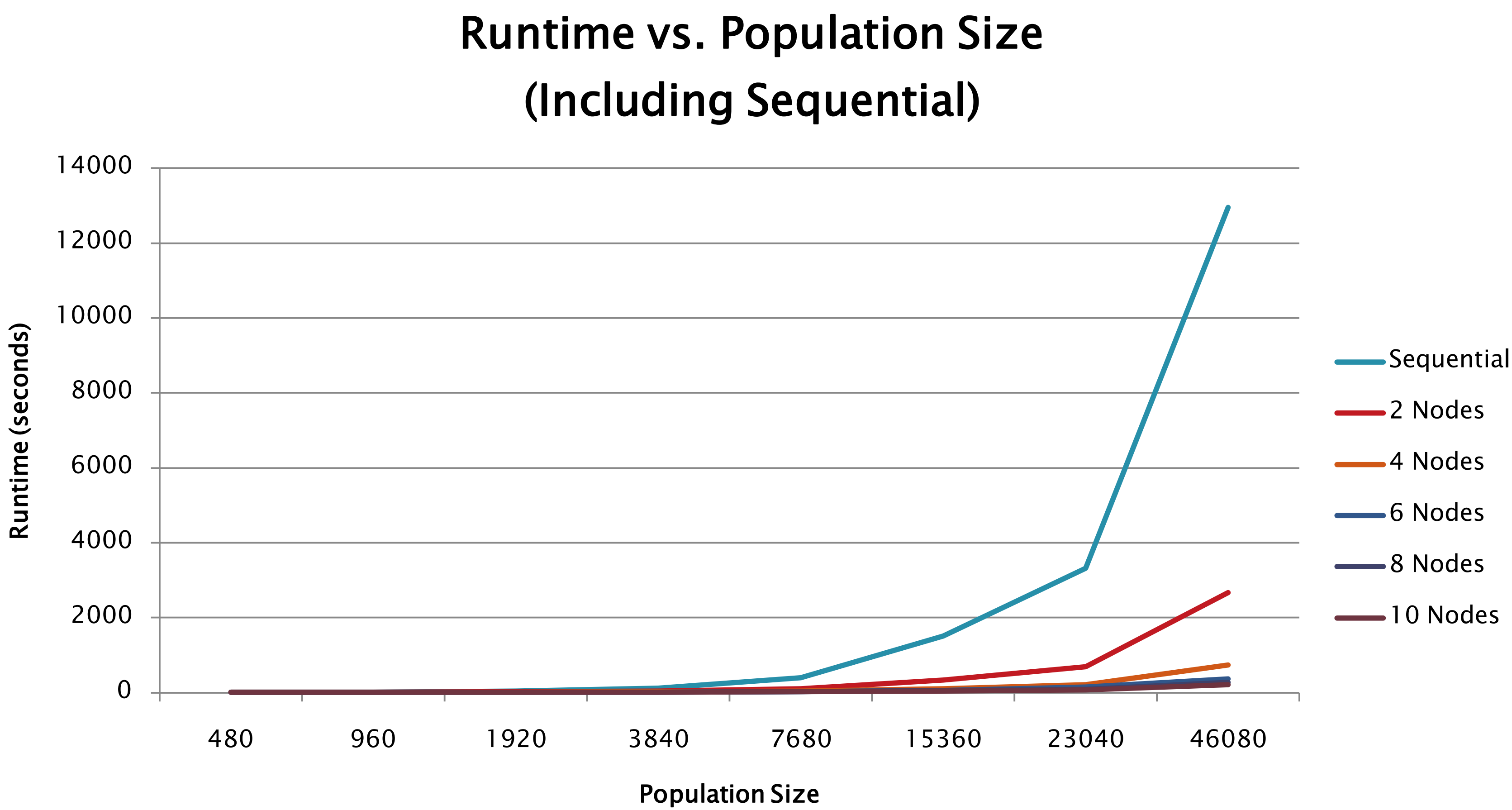
10-Node Timing Results



Timing Results – Multiple Nodes

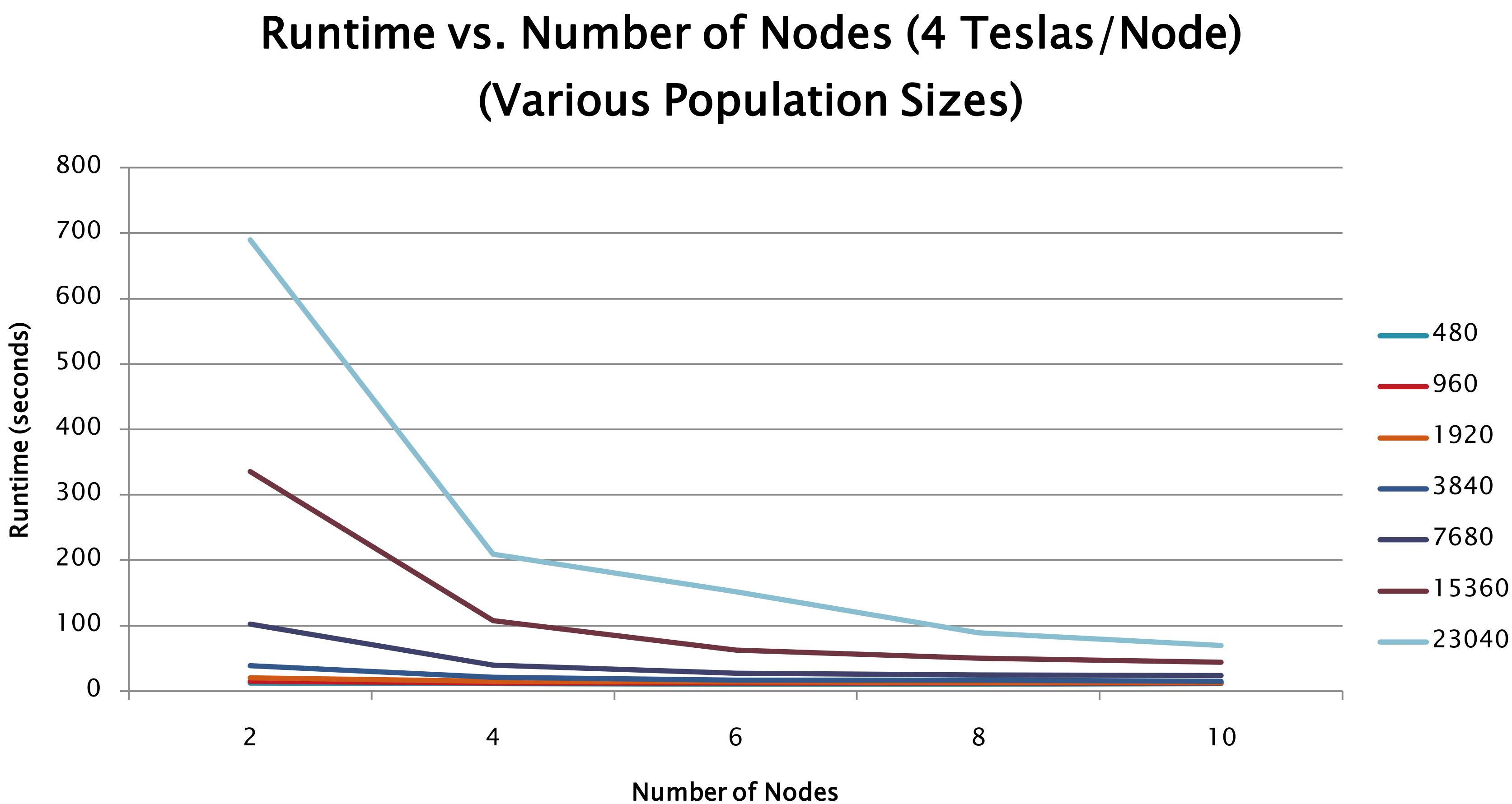


Timing Results – Multiple Nodes (with sequential)



Sequential Platform: Intel(R) Xeon(R) CPU E5430 @ 2.66GHz (same as worker nodes 9-13)

Timing Results – Fixed Population, Varying Nodes



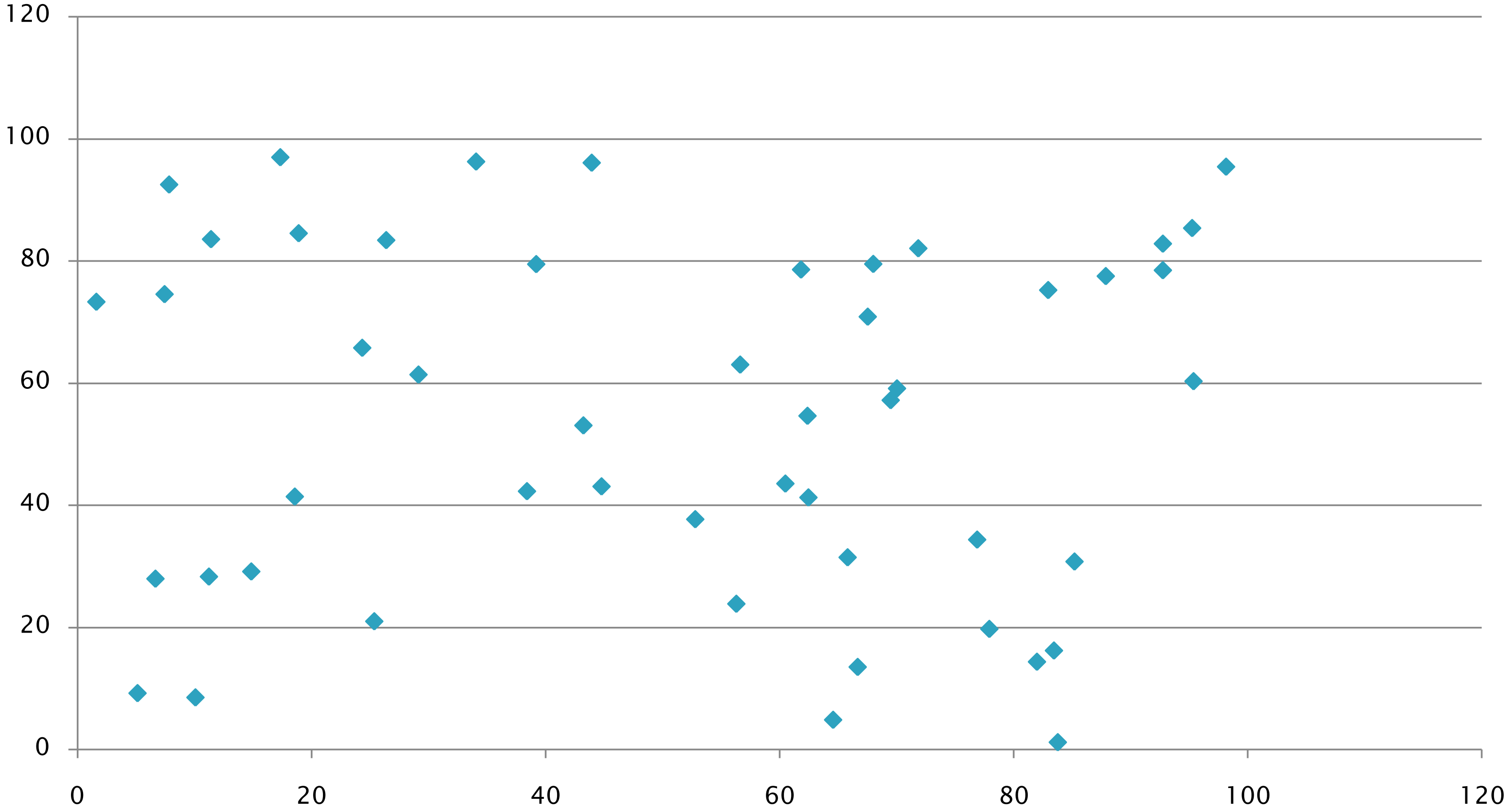
TSP Results: Background

- Sequential would eventually converge to a result and stick there.
- Simple parallelization of fitness evaluation just speeded this up but didn't result in better answers
- Advantages of parallelism (aside from speed of performance) came from use of subpopulations
  - Each node allowed to converge to a (possibly) sub-optimal answer, recombination at a global scale learned from all of these

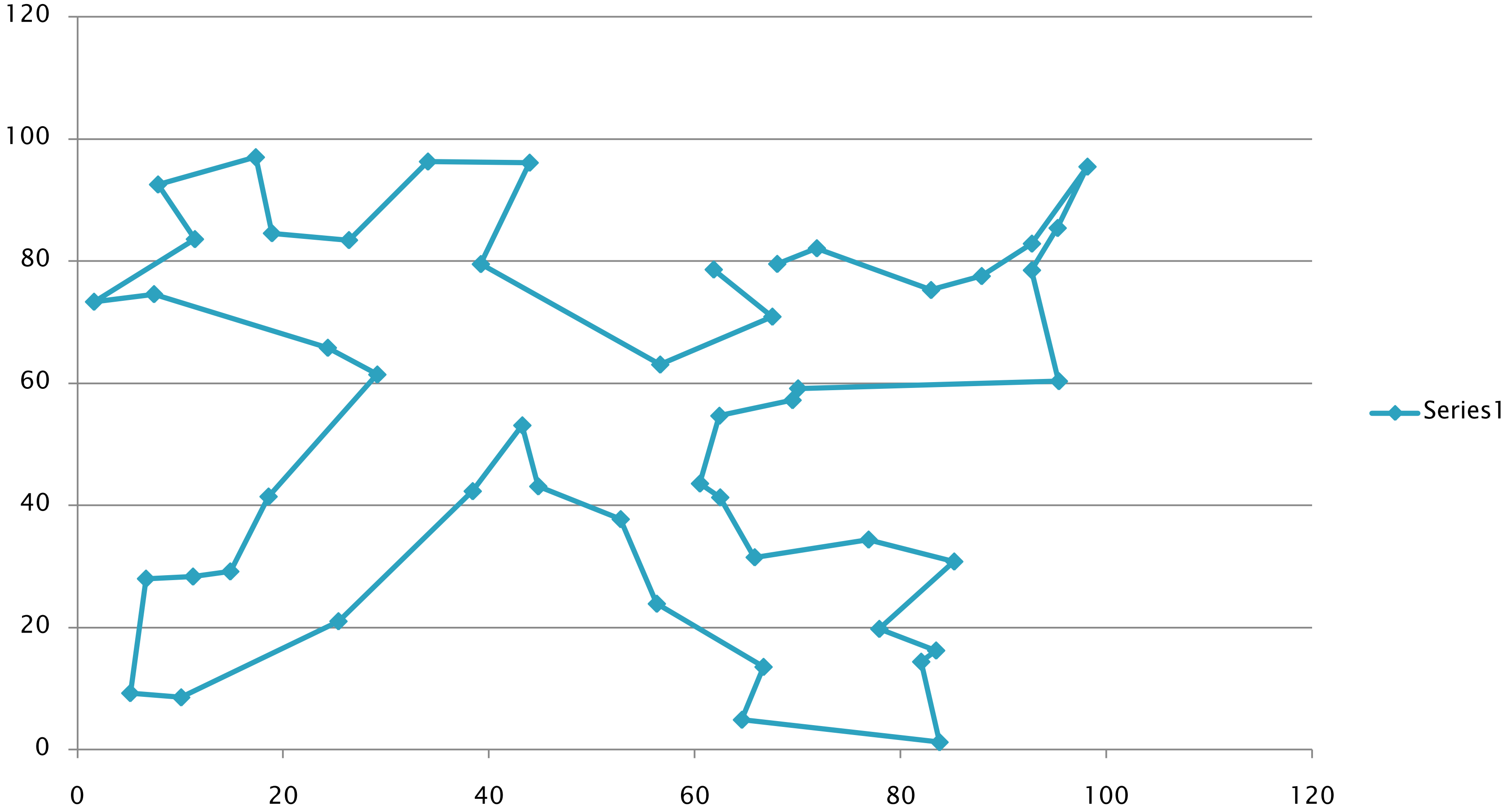
TSP Results: Parameters Used

- 50 Cities
- Crossover Probability: 65%
- Mutation Probability: 15%
  - Fairly high to help with early convergence
- Elitism: 3%

TSP Results: Test Example



TSP Results: Test Example



Conclusion

- What's next?
  - Modification of crossover / mutation operators
  - Tweaking parameters specific to this problem:
    - Population size
    - Proper balance between global and sub iterations
  - Generalize algorithmic framework for use in other optimization problems