Comparing Parallel CPU and GPU Strategies for the Traveling Salesman Problem

Vikas Malepati (vmalepat) and Shrikara Arun (shrikara)
https://parallel.sudokara.com

November 2026

1 Summary

We will compare CPU/GPU implementations of a route-improvement method for the traveling salesman problem on GHC and NCSA. We focus on how choices for i) computation placement and ii) distance storage affect runtime, and whether faster search finds shorter routes within a fixed time.

2 Background

Given a list of cities and pairwise distances between them, the traveling salesman problem (TSP) is to find the shortest route that visits every city once and returns to the starting city. It is NP hard and no algorithm is known that always finds the optimum route in polynomial time. We study the symmetric Euclidian case where travel costs are straight line distances between cities and are the same in both directions. This problem has lots of applications, including route planning and circuit board drilling.

We study the 2-opt method, which when given an arbitrary route, removes two connections and reconnects the route, reversing the section between them.

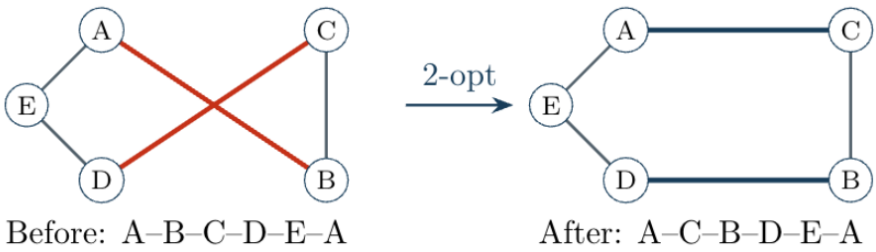


Figure 1: An illustration of the 2-opt algorithm. Choose a pair of edges to remove and reconnect nodes.

Each round checks all valid pairs, applies the best shortening change and repeats until no replacement will improve the route. A restart repeats this search from a different starting route. This finds good routes without guaranteeing the optimum.

3 The Challenge

- While multiple 2-opt replacements can be checked in parallel, they all must be based on the same initial route. The selection of the best replacement must happen and be propagated before the start of the next round.
- Also, some starting routes can finish their rounds quickly while others might need multiple rounds before 2-opt can no longer find a better route. This necessitates some notion of dynamic workload division, and we must take care to not cause warp divergence.
- The distance matrix is $O(N^2)$ with the number of cities, which can be larger than our cache size. The locality of accesses is also not expected to be very high across rounds. An alternative is to only store coordinates but recompute distances every time. On the GPU implementations, we have to trade off using shared memory with allowing running threads to access and use L1$.
- Finally, the CPU managed GPU version needs to return results to the CPU every round and continue work, while the GPU only version has sync points within the device code. The better version could depend on the problem size.

4 Resources

We will test our code on GHC and NCSA machines. While we are unsure of GPU availability via NCSA, GHC machines have a single Nvidia RTX 2080 which we can use for the GPU implementations.

[O'Neil and Burtscher(2015)] describes at a high level an OpenMP based CPU version which we will use as reference. It also describes the GPU resident version. [Fosin et al.(2013)] describes a version that does each parallel exploration on GPU but the CPU selects and applies the best change each round. It also describes the 3-opt version in our stretch goal.

While [O'Neil and Burtscher(2015)] provide a reference implementation, we treat it as a reference only and reimplement since we need to make changes to how the search is started and how the distances are represented. We thus make each implementation from scratch.

Benchmark inputs are available from [Fosin et al.(2013)], [DIM([n. d.])] and [tsp([n. d.])].

5 Goals and Deliverables

We seek to determine which hardware and work arrangement executes the same 2-opt search most efficiently and why. The main contribution is a controlled performance study, and we also study the effect on route cost, i.e. with a faster implementation, can we get a better route in the same real clock time?

We study:

- Computation placement - CPU (serial/OpenMP) vs GPU (CPU managed/GPU only)
- Distance storage - full distance matrix vs store only coordinates and recompute distances (for GPU versions only)

affect runtime and route quality across different problem sizes and types (uniform city distribution/clustered...) for different number of CPU threads/GPU blocks.

As a stretch goal, we hope to implement and try out the 3-opt variant from [Fosin et al.(2013)] for the best performing distance storage method we find to see how it differs from 2-opt in computation characteristics and route quality.

If things go slower than expected, we hope to complete at least one GPU version with the two distance storage ablations.

6 Platform Choice

We will use C++ with OpenMP for the CPU implementation and CUDA on Nvidia GPUs for the GPU implementations. A single GPU and its host CPU are sufficient for our planned experiments. The GHC and NCSA clusters provide access to CPUs with multiple cores and an Nvidia GPU for us to debug and run our experiments on.

OpenMP is a good fit because searches from different starting routes are independent. Each CPU thread can improve its own route while sharing read-only input data with other threads. Communication is limited mainly to distributing work and recording the shortest route found. While an MPI solution is also possible, we chose OpenMP since it is conceptually easier to think about (with the shared address space) and we have a high level reference algorithm described in [O'Neil and Burtscher(2015)]. CUDA is suitable because each improvement round contains many independent 2-opt checks. These checks can be distributed among threads, while separate thread blocks search different starting routes.

7 Schedule

In this schedule we assume that we incrementally update our report as we go along, so we have left only a few days at the end for finalizing the report and presentation.

- Week 1 (Nov 9-15): Gather the input benchmark data, build helper input readers, route validation and scoring, the serial search baseline.
- Week 2 (Nov 16-22): Implement the OpenMP and [O'Neil and Burtscher(2015)] GPU version. Get preliminary results for these versions for the midpoint report.
- Week 3 (Nov 23-29): Recalibrate after midpoint report if necessary, implement [Fosin et al.(2013)] GPU version and extend the GPU methods to support both distance storages
- Week 4 (Nov 30-6): Benchmarking and experimentation, plotting results, stretch goal of 3-opt if time permits
- Dec 7-10: Finalize report and analysis, prepare presentation and poster

References

[DIM([n. d.])] [n. d.]. Dimacs. `https://archive.dimacs.rutgers.edu/Challenges/TSP/about.html`

[tsp([n. d.])] [n. d.]. TSPLIB. `https://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

[Fosin et al.(2013)] Juraj Fosin, Davor Davidović, and Tonči Carić. 2013. A GPU implementation of local search operators for symmetric travelling salesman problem. *Promet-Traffic&Transportation* 25, 3 (2013), 225–234.

[O'Neil and Burtscher(2015)] Molly A O'Neil and Martin Burtscher. 2015. Rethinking the parallelization of random-restart hill climbing: A case study in optimizing a 2-opt TSP solver for GPU execution. In *Proceedings of the 8th workshop on general purpose processing using GPUs*. 99–108.