Parallel Route Planning with Time-Dependent Graphs

Felipe Mautner, Ethan Muchnik

Carnegie Mellon University

Time-Dependent Graphs

How to model roads?

- Road Segments into vertices
- Connect Segments with edges
- → Directed graph

How to model traffic on roads?

- Traffic is periodic. Edges are heaviest at rush hour, lightest at early morning.
- Edge weight function must also be periodic, parameterized by time.

Time-Dependent Graphs

$$G = (V, E, T)$$

$$T : E \times \mathbb{Z}^+ \to \mathbb{R}$$

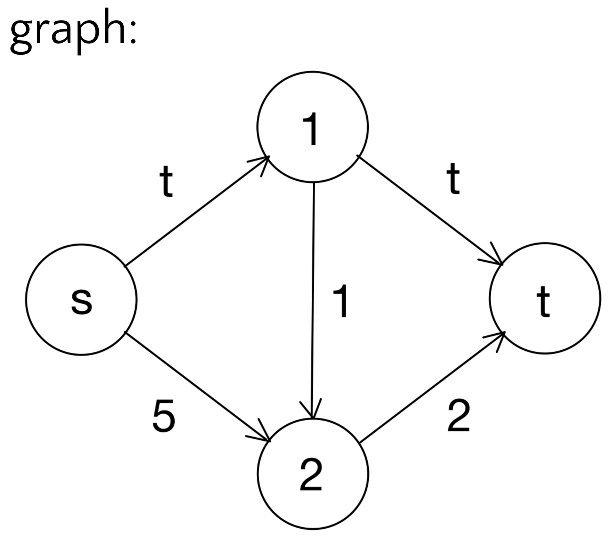
Properties: First-in-First-Out (FIFO)

If two vehicles follow the same path, the one leaving later can not arrive any earlier:

$$t_x \leq t_y \implies T(e, t_x) + t_x \leq T(e, t_y) + t_y$$

This assumption is necessary to ensure the correctness of the algorithms used. It also makes sense in the real world.

Example graph:



Depending on the time t of departure from s, the shortest path from s to t can be all of s→1→t; s→2→t; S→1→2→t. Time-dependent graphs are quite complex!

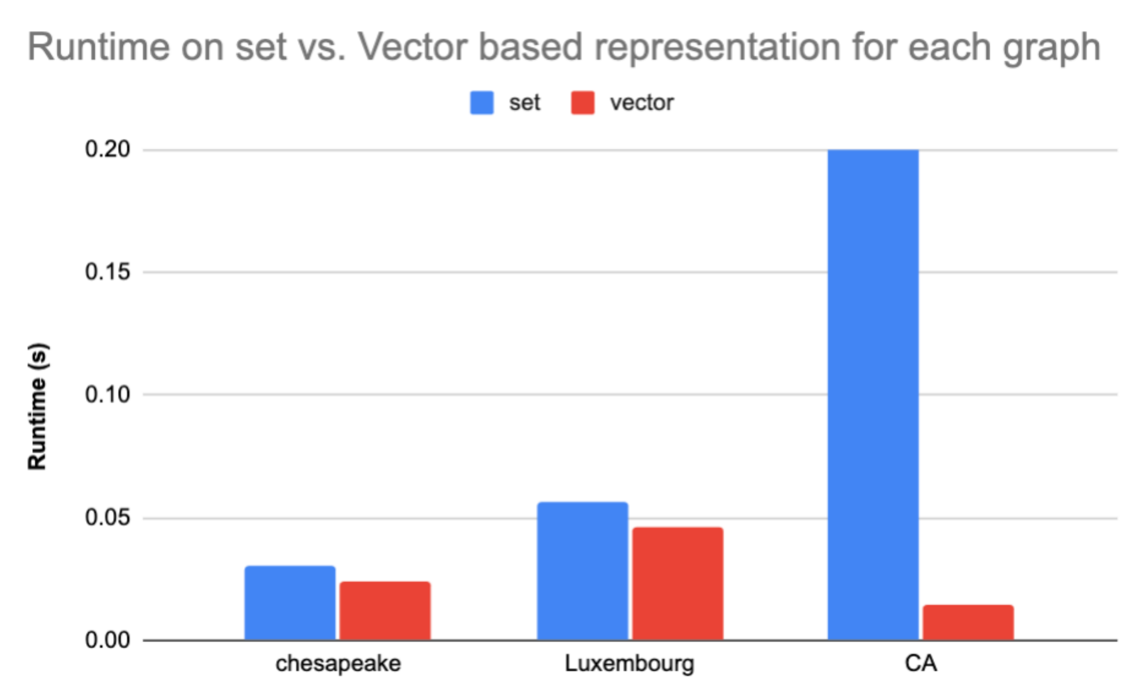
The definitions and assumptions made allow for extensions of classic graph algorithms into TDG's.

The 4 SSSP Variations

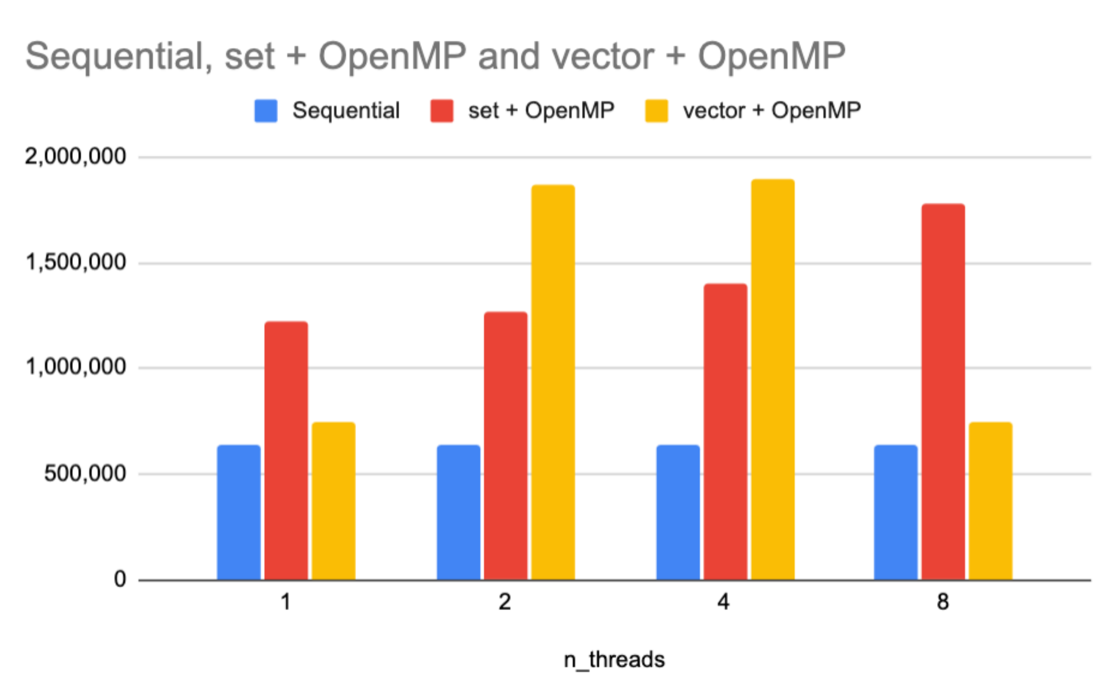
- Sequential: An adaptation on Dijkstra's Algorithm called Time-Dependent Dijkstra's (TDD).
- (Naïve) Parallel Edge Relaxation: uses OpenMP to parallelize edge relaxation loop. PQ not thread-safe, needs critical section delimitation.
- Expanding Visited vertices with set: Slight adaptation of TDD where visited vertices are kept in a set.
- Expanding Visited vertices with vector: Same idea, but visited nodes represented by Boolean vector.

Evaluating how they compare

- **Set vs. Vector**

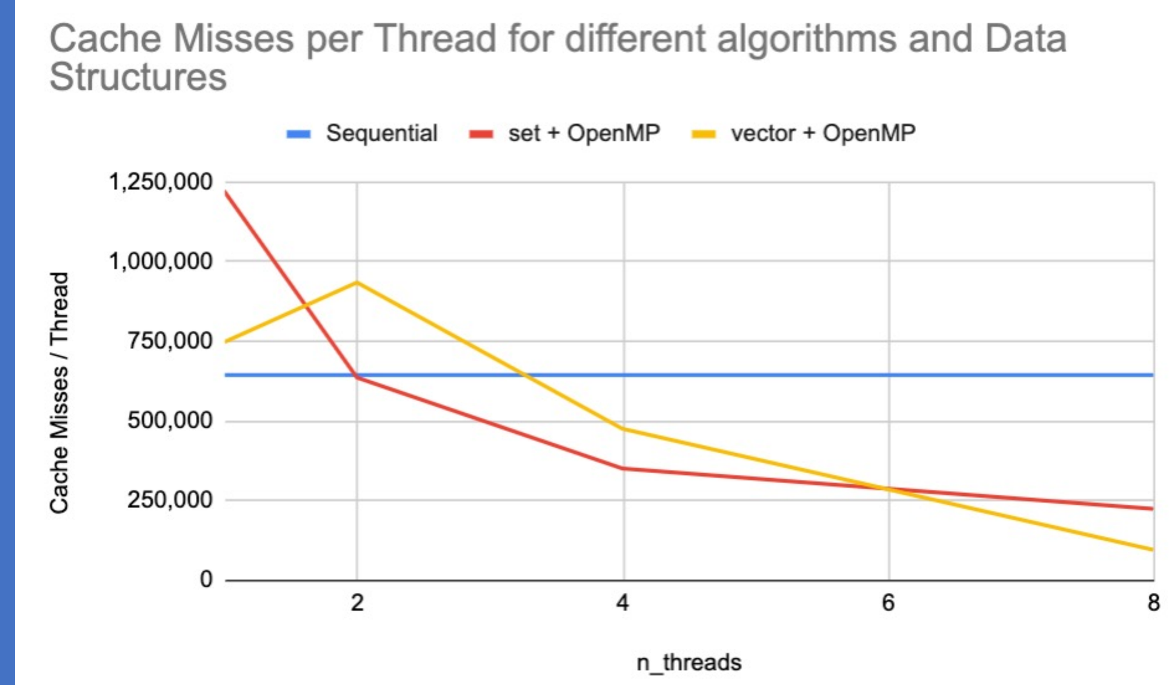


- **Cache misses**



(More on cache use)

Per thread cache misses



Both show reasonable scaling capabilities, with cache misses per thread decreasing at a similar rate as the number of threads increases.

Multiple Vehicle Routing

We can use the fast sequential SS-TDD algorithm as a subprocedure to route multiple vehicles.

Algorithm 3 Multiple Vehicle Routing

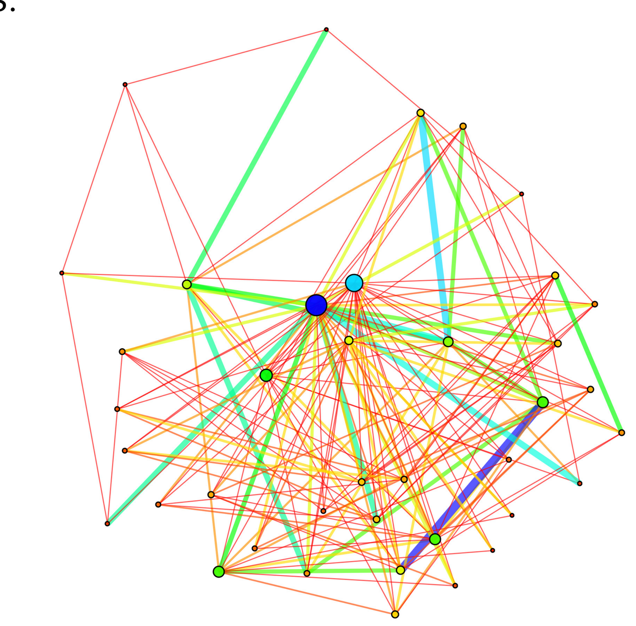
Input: Vertices $\{u_1, u_2, \ldots, u_k\}$, current time t_{curr}, graph $G = (V, E, T)$
for each *car* in parallel **do**
  run SS-TDD(graph, car)
  Every N iterations, accumulate graphs
end for

Synchronize every N steps → Tradeoff: runtime for quality of routing.

The Data we used

Real world road network graphs:

- Small: Chesapeake, VA. Small town, few roads. 39 Vertices, 170 Edges:



- Medium: Luxembourg. 114.6K Vertices, 119.7K Edges.
- Large: California Road System. 1.97M Vertices, 2.77M Edges.

Conclusion

Inherent challenges to parallelism

- Road Networks are usually very sparse. As such, the number of out edges from a node are usually less than the number of processors. Bad for parallelism.
- Dynamic assignment usually imbalanced, poor utilization of processors.
- Parallelizing over nodes is a lot more complicated due to the dynamic nature of the graphs.

Next Steps

- Graph partitions and contraction hierarchies have been shown to work on dynamic graphs.
- Problem and hardware-specific, tailored solutions are best on a case by case basis.
- Fine-tuned parallelism.