CS 509 Lab Assignment 1

ISHAN TRIPATHI 2021AIM1009

Feb 2021

1 Introduction

The problem of path finding in road networks has been of great importance. Its value addition potential gets boosted even further when the path-finding algorithms start to consider the traffic congestion patterns. A recent report from McKinsey [4] estimates that we can save billions of dollars (annually) by helping vehicles avoid traffic congestion.

Given the overall significance of the problem area, several researchers (e.g., [2, 3]) have been exploring it from different aspects. Among these, the most fundamental being: computing the fastest path between a source and destination for a given departure-time (at source). In this problem, the underlying road network (and its time-varying traffic congestion) is conceptualized as a time varying graph (more details later). Given a time-varying graph representation of the road network, a source s , a destination d and, a departure-time (t_{dep}); the goal is to determine a journey $\mathcal{L}$ between s and d which departs from s at time t_{dep}. The key property of $\mathcal{L}$ being that there does not exist any other journey $\mathcal{L}'$ between s and d which departs from s at time t_{dep} but arrives at destination earlier than $\mathcal{L}$. This optimal journey $\mathcal{L}$ has been referred to as a shortest lagrangian path in some literature [3] (more details later).

Limitations of the Related Work: Though there have been several works in the area of parallel algorithms for the shortest paths problem (e.g., [5, 7, 6, 1]), they cannot be extended for our SLP problem because they cannot be modified to follow the "Lagrangian reference frame". Parallel algorithms [6, 1] based on label correcting based approaches are in principle *not suitable* for implementing Lagrangian reference frame.

2 Problem Statement

We now formally define the problem of capacity constrained assignment (CCA) by detailing the input, output and the objective function:
Given:

- A road network represented as a directed $G(V, E)$, where each edge $e \in E$ has a positive cost.
- A set of service centers $S = \{s_1, ..., s_{n_s}\}$ where $S \subset V$. Each service center s_i has a capacity c_{s_i} and a penalty function p_{s_i}.
- A set of demand vertices $D = \{d_1, ..., d_{n_d}\}$ where $D \subset V - S$.

Output: An allotment consisting of pairs $< d_k, s_j >$.
Objective Function:

$$Min\left\{ \sum_{\substack{s_i \in \text{Service} \\ \text{Centres}}} \left\{ \sum_{\substack{d_j \in \text{Demand vertices} \\ \text{allotted to } s_i}} SCost(d_j, s_i) \right\} + Total\ \ Penalty\ \ across\ \ all\ \ s_i \right\} \quad (1)$$

3 Sample Equations

$$
\begin{aligned}
E_{sys}(t) &= E_{edge}^{comm}(t) + E_{edge}^{hover}(t) + \left(\sum_{i=1}^{N} (E_i^{comm}(t)) \right) \\
&= \kappa * \|vel(t)\|^2 \\
&= (2^{\frac{D_{edge}}{W}} - 1) * \frac{N_0 W}{\alpha}
\end{aligned} \tag{2}
$$

4 Overview of solution

This assignment could have happened under following two cases: **Case (a)** s_i had free space (in terms of capacity) or; **Case (b)** s_i was already full. Case (a) is quite straightforward and no further operations are needed. Note that in this case, we also need not pay any penalty for this assignment. However, in case(b), after assigning d_i to s_j , the algorithm undertakes the process of re-organizing the current solution with an intention of lowering the current objective function value. Note that in case (b), the objective function would increase by the quantity $\delta = SCost(d_i, s_j) + Penalty(s_i)$ after the assigning d_i to s_j. We now briefly present our idea of re-organization at high level.

Service Center	Capacity	Penalty
S1	2	1
S2	1	15
S3	1	5
S4	1	15

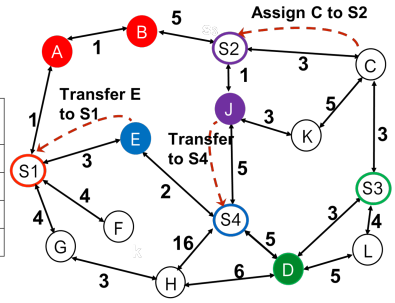

Figure 1: Figure 1: A partially constructed assignment on a sample input. Processed demand nodes are filled with the same color as their respective service center. (Best in color)

Algorithm 1 Assignment Subspace Re-organization based Approach for CCA

Input: (a) α : allowed breadth of the search in assignment subspace tree ($\alpha \leq \mid S \mid -1$);
Output: Assignment $\mathcal{A}$ with objective function value Δ

```
1: while ClosestSC heap is not empty do
2:   if s* has vacancy then
3:     Decrement capacity of s*
4:   else
5:     Increment Δ by penalty corresponding to s*
6:     Initialize the list of seeds Γ
7:     Γ ← Determine α seeds
8:     if α ≤ τ then /*allowed breadth of the search is less than #threads*/
9:       λ ← Create α threads
10:    else/*allowed breadth of the search is more than threads*/
11:      λ ← Create τ threads
12:    end if
13:    Create a job queue JQ with seeds present in Γ. /*Each seed is a job*/
14:    Barrier to ensure that all threads in Λ terminate
15:  end if
16: end while
```

References

[1] V. T. Chakaravarthy et al. Scalable single source shortest path algorithms for massively parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):2031–2045, 2016.

[2] A. Davidson et al. Work-efficient parallel gpu methods for single-source shortest paths. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 349–359. IEEE, 2014.

[3] Ugur Demiryurek et al. A case for time-dependent shortest path computation in spatial networks. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 474–477, 2010.

[4] V. Gunturi et al. A critical-time-point approach to all-departure-time lagrangian shortest paths. volume 27, pages 2591–2603. IEEE, 2015.

[5] Saeed Maleki et al. Dsmr: A parallel algorithm for single-source shortest path problem. In *Proceedings of the 2016 International Conference on Supercomputing*, pages 1–14, 2016.

[6] Hanke N et al. The age of analytics: competing in a data-driven world. *McKinsey Global Institute Research*, 2016.

[7] Y. Simmhan et al. Distributed programming over time-series graphs. In *2015 IEEE International Parallel and Distributed Processing Symposium*, pages 809–818. IEEE, 2015.