Speeding up the Frank Wolfe Algorithm — Implementation changes, Contraction Hierarchies and Bi-conjugate direction

Debojjal Bagchi

March 19, 2024

Four changes were tried on the original link flow based traffic assignment. Three of these changes lead to an improvement in run times while one of them did not.

1 Implementation changes (First Improvement)

Algorithm changes: We observe while calculating the Average Excess Cost (AEC) or Relative Gap (RG), there is no need to calculate the shortest paths again. The shortest paths are readily available from the All-or-Nothing (AON) of the current iteration and can be used directly to calculate AEC or RG.

Code changes: The shortest paths computed in the AON function (`Network.allOrNothing()`) function is returned in the `Network.userEquilibrium()` function and is passed as input to `Network.averageExcessCost()` and `Network.relativeGap()`. Further the net demand value is stored `Network.netdemand` attribute of `Network` class as soon as it is initialised. Thus the net demand values need not be calculated in each iteration while computing the AEC

Implication: This is a minor change but slashes the run time by halfves.

2 All-or-Nothing Assignment

Algorithm changes: A quick profiling of the existing Frank Wolfe Algorithm shows the All-or-Nothing (AON) is the bottleneck of the whole algorithm. There are two parts essentially in the AON step.

a. Calculate all the shortest paths between all the OD pairs

b. Load all flows on the shortest paths

2.1 Contraction Hierarchies (Second Improvement)

Algorithm changes: The first step is tackled using a label setting algorithm. The algorithm is called number of origin times each time a AON assignment is required. However the network does not change for each of the AON assignments. Hence, for large networks contraction hierarchies should be used. The "shortcuts" can be computed once the AON assignment is required. One this new network the shortest paths between all the OD pairs can be computed very fast.

Code changes: A python library `Pandana`[1] is used to compute the contraction hierarchies. The function `Network.create_ch()` uses the current network link costs (stored in `cost` attribute of class `link`) to create a `Pandana.Network` object. The creation of `Pandana.Network` object forms the contraction hierarchies in parallel threads. Each time `Network.allOrNothing()` is called a new `Pandana.Network` object is created and all pair shortest paths are computed. The output of the shortest paths are the full shortest paths and not the backtrack labels.

[1] https://udst.github.io/pandana/

Implication: This is a thread dependent function. Running on a machine with more number of threads will fasten up the process. In general for smaller networks using contraction hierarchies is counter-productive as prepossessing the shortcuts are not necessary and a simple label setting algorithm performs better.

2.2 Shortest Path Tree based AON assignment (No Improvement)

Algorithm changes: It is observed that one can us the shortest path tree structure and add flow in the reverse topological order. Although this process seem faster, in all test cases directly adding the flow on the paths proved faster. This is essentially because while using contraction hierarchies we no longer have access to backtrack labels. Hence the tree has to be computed and then a topological order has to be found. This is slower than readily adding the flow to each path.

3 Bi-conjugate direction (Third Improvement)

Algorithm changes: Another reason why Frank Wolfe is slow is that it takes large amount of time to converge. Rather using new all or nothing assignment to calculate the direction isn't a great idea near to equilibrium. Thus, Bi-conjugate directions can be used (Mitradjieva and Lindberg, 2013). This process uses the current AON assignment as well as AON assignment from last two iterations to compute the new direction. Once the direction is found, it uses Newton's method to find the step size.

Code changes: A Julia implementation[2] of Bi conjugate direction is used for reference. The direction is computed in `Network.BCFWStepSize()` function. The function requires the diagonal vector of the Hessian of the Beckmann Function. The components of the diagonal is stored is computed in `Link.calculateHessianComponent()` function which is $\frac{\alpha\beta t_0 . x_{ij}^{\beta-1}}{c^\beta}$ (symbols have usual meanings as that of BPR function). The function `Network.hessian_diag()` computes the diaginal vector of Hessian using `Link.calculateHessianComponent()`. `Network.BCFWStepSize()` also requires the gradient of Beckmann function which is essentially the travel costs available at `cost` attribute of `link` class. `Network.BCFWStepSize()` compute the new flow and returns the earlier two AON assignemnt. Specifically it takes three AON assignment as input and returns the new flow and the last two AON assignments. The variable names and code is kept similar to the Julia implementation for better understanding which in turn uses similar notations from Mitradjieva and Lindberg (2013). The first iteration is exactly same as Frank-Wolfe and second iteration is conjugate Frank Wolfe which uses the current and past AON assignment. From the third iteration Bi-conjuagte direction is used. Since, the new flow is found in `Network.BCFWStepSize()` directly `Network.shiftFlows` is not required. The function `Network.shiftFlowsBCFW()` just updates the link flows and travel costs of each link.

Implication: The Bi-conjuagte direction is extremely efficient in finding a good direction. In fact studies (Dial, 1999) by the original authors of Algorithm B has shown Bi-conjugate Frank Wolfe to be faster than Algorithm B upto a Relative Gap if 10^{-4} in the Chicago Regional network [3].

4 Results

Three conditions were used to terminate the algorithm in the given results:

(a) If the number of iterations crossed 1000

(b) If the time taken crossed 1000s

(c) If the Average Excess Cost reduced below 10^{-4}

If conditions (a) or (b) terminated the algorithm, we say the algorithm d*id not converge*; else we say the algorithm *converged*. As expected the first improvement improves computation by reducing run times by halves. The second improvement with contraction hierarchies also improves runt times. However, as expected contraction hierarchies are only better for large networks. Finally Bi-conjugate direction makes the traffic assignment remarkably fast, although

[2] Available at https://github.com/chkwon/TrafficAssignment.jl

[3] My implementation is nowhere close to this!

it still suffers from "tailing". **Based on the results the final code submitted uses label setting algorithm if number of Zones are less than 120 and Contraction Hierarchies otherwise.**

Table 1: Summary of results (*Gap*: Avergae Excess Cost, *Time*: Time in seconds, *It.*: Iteration count. Times in bold means the algorithm converged in lesser than 1000s and 1000 itterations)

Network	Unchanged FW Algorithm			With 1st improvement			With 1st & 2nd Improvements			With all 3 Improvement		
	Gap	Time	It.	Gap	Time	It.	Gap	Time	It.	Gap	Time	It.
Anaheim	9×10^{-5}	**18.60**	62	9×10^{-5}	**10.82**	62	7×10^{-5}	**16.73**	288	2×10^{-5}	**1.69**	39
SiouxFalls	0.00224	9.29	1000	0.00224	5.91	1000	0.00226	9.26	1000	8×10^{-5}	**1.71**	230
Eastern Massachusetts	10^{-4}	**48.52**	322	10^{-4}	**27.15**	322	7×10^{-5}	**11.01**	246	10^{-4}	**1.31**	34
ChicagoSketch	0.003032	1000	53	0.003032	1000	53	0.000192	1000	520	10^{-4}	**244.7**	132

References

Dial, R. B. (1999). Algorithm b: Accurate traffic equilibrium (and how to bobtail frank-wolfe. *Volpe National Transportation Systems Center, Cambridge, MA*.

Mitradjieva, M. and P. O. Lindberg (2013). The stiff is moving—conjugate direction frank-wolfe methods with applications to traffic assignment. *Transportation Science 47*(2), 280–293.