Upright Stiff – Improved Conjugate Frank-Wolfe methods through subproblem solution updating.

Johan Holmgren

School of Computing,

Blekinge Institute of Technology, Karlskrona, Sweden

P O Lindberg*

Depts. Transp. Science and Numerical Analysis, and Centre for Transp. Studies

KTH Royal Inst. Technology, Stockholm, Sweden

*Email: polin@kth.se

1 Introduction

In recent papers, e.g. [1,2,3], CFW, *Conjugate Frank-Wolfe Methods*, have shown to be competitive with other state of the art algorithms for Traffic Assignment, in particular when using several processors. CFW methods speed up the computations by finding better search directions. In the current paper we achieve even better improvements by updating the shortest path solutions of the subproblems, rather than computing them from scratch.

It is classical that the FW (Frank-Wolfe) subproblem essentially entails finding shortest path (SP) trees for all origins, and subsequently sending all demand along the SP trees, giving an AON (All-Or Nothing) solution, the *FW point*. See e.g. [4] for a general description of the *Traffic Assignment Problem* (TAP) and some methods for it. In the main routine, one then performs a line search towards the FW point, or in CFW, towards other feasible points, obtained by conjugation. It further is well known, that the main computational burden in FW methods is the determination of the SP trees, see e.g. [3]. In the current paper, we suggest saving workload by updating the SP trees.

Updating the subproblem solution, the main computational burden is to generate new SP trees. In an SP tree, all nodes have a *predecessor* label, which points to the next node in the shortest path to the root node. We use *thread* labels to handle the shortest path trees in the subproblem solutions. A *thread* is a set of node pointers giving a depth first traversal of the tree, i.e. such that all successors of a given node are traversed before any non-successor node. In particular, we mainly use thread, predecessor, and depth labels. During scanning and updating, we might use other labels, such as back-thread. At updating, we of course have to update the labels too.

Scanning for new arcs to enter the tree, during subproblem solution, one in principle has to scan the list of links several times in order to find all links to enter the SP tree. Moreover one needs one final complete scan of all the links, to guarantee, that one has found all links that should enter a given SP tree.

We suggest two improvements to this standard approach:

1. *Thread Following link scan*, and
2. *Bucketed link scan*

In Thread Following link scan, we scan the links by the nodes, following the thread. When we pivot, i.e. change the SP tree, the root node of the part of the tree that is moved, is placed in the thread after the current successors of the current node.

Using thread following link scan, we can prove that we only need a single traversal of the thread to find all links that need to enter the SP tree.

In the Bucketed link scan, we do a sort of partial pricing. Instead of scanning all arcs, for each origin, we keep and update a "bucket" of "promising" links. For several iterations we only scan the links in the buckets. This gives suboptimal subproblem solutions, but speeds up the convergence. Every now and then, we do a complete scan, and then add and delete links to/from the buckets.

The results presented below, emanate to a large extent from [6].

2 Computational Experience

We have applied our methods to a set of Classical test Cases, Sioux Falls, Winnipeg, Barcelona, Chicago Sketch and Chicago Regional, see e.g. [3] for problem data.
As is classical for TAP, a complete subproblem solution, whether using FW or CFW, gives in each iteration *k* an *upper bound* UBD_k as well as a *lower bound* LBD_k on the optimal objective value. These can be used to compute a *relative gap*, $\gamma_k = \frac{UBD_k - BLB_k}{BLB_k}$ between the current and the optimal solution. Here, BLB_k is the *best lower bound*, i.e. the maximum of all LBD_k up to and including iteration *k*.

We have run codes for our methods on said problems, and accumulated statistics concerning number of iterations and running times, as well as other data.

2.1. *Node Scanning Overhead.*

It is interesting to study the *Node Scanning Overhead* (NSO), i.e. the number of nodes scanned in a complete scan of the thread, as compared to the total number of nodes. It natural that the NSO is high in the early iterations, when there are many non-optimal links in the SP trees. But apart from that, as soon as the situation stabilizes, the NSO is astonishingly low, typically below 10%, except for the first few iterations.

2.2. *Reductions in running times for Thread Following*

It turns out that our savings in running time are lower in the early phases of the algorithm. Therefore, iteration-wise extremely easy problems, such as Chicago Sketch, get less savings. This problem only takes around 20 iterations to arrive at a relative gap of 10^{-4}.

For the other problems, the computation time reductions for thread following ranges from around 30% for the relatively easy Winnipeg up to around 60% for the more difficult Sioux Falls. These results hold true for plain FW, as well as the singly conjugated CFW, and the doubly conjugated BFW.

2.3. *Reductions in running times for Bucketed scan.*

In similarity to Thread Following, Bucketed scan need the problem to stabilize to reap its benefits. But in the later iterations, at relative gaps on the order of 10^{-6}, the extra savings, on top of those for Thread Following are generally of the order of 20-60%.

The total savings, using both approaches, vary between 60 and 90%, for the smaller relative gaps.

References

[1] Caliper Corporation (2010), "What TransCAD Users Should Know about New Static Traffic Assignment Methods," Caliper Corporation Communication to Users.

[2] Zhou, Z., and M. Martimo (2010), "Computational Study of Alternative Methods for Static Traffic Equilibrium Assignment," *Proceedings of the World Conference on Transport Research Society (WCTRS)* , Lisbon, Portugal.

[3] Mitradjieva, M. and P.O. Lindberg, (2011), The Stiff is Moving - Conjugate Direction Frank Wolfe Methods with Applications to Traffic Assignment, to appear in *Transportation Science.*

[4] Patriksson, M. (1994), *The Traffic Assignment Problem - Models and Methods*, VSP, Utrecht.

[5] Glover, F., D. Klingman, and J. Stutz (1974), Augmented Threaded Index Method for Network Optimization, *INFOR*, **12**, pp. 293-298.

[6] Holmgren, J. (2004), Efficient Updating Shortest Path Calculations for Traffic Assignment, Masters Thesis, Dept Mathematics Linköping University.