Local Search Algorithms for Reserved Delivery Subnetwork Configuration Problems with Cycle and Bicycle Reduction

Ruibiao Qiu Jonathan S. Turner
{ruibiao,jst}@arl.wustl.edu
Applied Research Laboratory, Department of Computer Science and Engineering
Washington University, St. Louis, MO 63130, USA

***Abstract*— A reserved delivery subnetwork (RDS) is a semi-private network infrastructure to enable an information service provider to deliver more consistent service to its customers. In our previous study of the RDS configuration problem, we formulated the problem as a minimum concave cost network flow problem (MCCNFP), and presented an efficient approximation algorithm, largest demand first or LDF, to provide approximate solutions to this NP-hard problem in practice. In this paper, we apply local search algorithms based on negative cost cycle and bicycle reduction to further improve the quality of the results obtained from LDF. Our simulation results show that the local search algorithms improve the quality of LDF solutions, but with added computational complexity. They also further indicate that LDF produces solutions reasonably close to the local optimum without added complexity, and is suitable for large networks in practice.**

I. Introduction

We proposed the *Reserved Delivery Subnetwork* (RDS) as a means to enable an information service provider to deliver more consistent service to its customers in today's Internet [1]. In brief, an RDS is a semi-private network infrastructure used by an information service provider, connecting it to its customers at different metropolitan areas. The *endpoints* of an RDS include a *source node* and a potentially large number of *sink nodes* distributed within a fixed network infrastructure. Sink nodes are typically routers within metropolitan areas where customers of the information service are found. A network provider selects a set of links within the network and provisions bandwidth reservations on those links in order to accommodate expected traffic flows from the server to the various sink nodes. This allows traffic from the source node to flow through to the sinks without contention from other traffic sources, improving quality of service.

To allow for variability in the traffic volume at sink nodes, reserved bandwidth is provisioned based on the mean and variance of the expected traffic. Links that carry large traffic volumes are generally more efficient than links that carry small traffic volumes, since the amount of bandwidth that must be reserved to accommodate traffic variability becomes a smaller fraction of the total as traffic volume grows. This effect makes it beneficial to group together flows going from the source to sinks that are close to one another. This benefit of flow aggregation is reflected in a concave edge cost function that grows more slowly as the average aggregated traffic on a link increases.

In our previous study [1], we showed that the RDS configuration problem can be formulated as a minimum concave cost network flow problem [2], a well-known NP-hard problem. We proposed an efficient approximation algorithm, largest demand first or LDF, for solving the RDS configuration problem in practice. In this paper, we apply local search algorithms based on negative cycle and bicycle reduction to improve the quality of solutions obtained from LDF. We first apply the cycle reduction algorithm proposed by Gallo and Sodini in [3]. In addition, we observe that the existence of negative cost bicycles can also lead to lower cost flows from an existing one. We introduce bicycle reduction to further improve the solution quality after applying cycle reduction to the LDF solution. Our experimental results show that both cycle and bicycle reduction algorithms improve the LDF solution quality with added computational complexity, while LDF produces results that are usually within a small constant factor of an easily computed lower bound, and are close to local optimal solutions with much lower computational complexity.

The rest of this paper is organized as follows: in Section II, we recap the formulation of the RDS configuration as a minimum cost flow problem, and briefly describe the LDF algorithm. Section III describes the local search algorithms based on negative cycle and bicycle reduction. Experimental results are given in section IV and concluding remarks in Section V.

II. Largest Demand First Algorithm for RDS Configuration problem

A. Problem Formulation

Given a directed graph $G = (V, E)$, define two real-valued functions: length $l(\cdot)$ and bandwidth $b(\cdot)$ defined on E, which are the physical distance and bandwidth capacity of an edge, respectively. We are also given a *source node* $r \in V$ and a set of *sink nodes* $S \subseteq V$, with each sink node s having a mean demand $\mu(s)$. The minimum cost reserved delivery network that satisfies the mean demands can be found by solving a minimum cost flow problem, in which the flow into each sink is given by its mean demand, and the total flow on each link e is bounded by its link capacity $c(e)$. The cost of

a flow x on an edge e is defined to be $l(e)(x + \gamma x^{1/2})$. The second factor in this expression corresponds to the amount of bandwidth that must be reserved to accommodate a flow of magnitude x. The "extra" reservation of $\gamma x^{1/2}$ allows for traffic variability. If the underlying traffic consists of many statistically independent flows, this variability grows as the square root of the traffic volume. Note that the cost function is concave. Given a minimum cost flow that satisfies the demand, the optimal RDS is the subgraph of G defined by the edges with non-zero flows. The cost of the RDS is the sum of the costs of the flows on its edges.

B. Largest Demand First Algorithm

In our previous study of the RDS configuration problem [1], we proposed the largest demand first (LDF) algorithm as an approximation algorithm for the RDS configuration problem. It is a variant of the classical minimum cost augmenting path method for solving minimum cost flow problems in a network with linear edge costs. As with the original algorithm, we seek a minimum cost augmenting path at each step. However, as we showed in our study, the choice of such a path is complicated by the fact that the relative cost of different paths depends on the amount of flow sent on them. After investigating the implications of this problem, we devised LDF to resolve the problem.

For any edge e in the original graph, the cost of carrying x units of flow on e is $l(e)(x + \gamma x^{1/2})$. We let $\delta_f(e, \Delta)$, be the change in cost caused by adding Δ units of flow on the edge e in the residual graph. We refer to $\delta_f(e, \Delta)$ as the *incremental cost* of the edge e, with respect to the increment Δ. The incremental cost of a path, with respect to an increment Δ, is the sum of the incremental costs of its edges. For any flow f and flow increment Δ, we can define a shortest path tree $T_f(\Delta)$ in the subgraph of the residual graph, in which all edges have residual capacity no less than Δ. The root of $T_f(\Delta)$ is the source node, and the edge lengths are defined with respect to the incremental costs, $\delta_f(e, \Delta)$. As Δ is increased from zero, we get a finite sequence of trees $T_0, T_1, \ldots, T_m$. For each tree T_i in this sequence, there is a corresponding range R_i of values of Δ. The *incremental cost per unit flow* of an augmenting path p is $\delta_f(p, \Delta)/\Delta$, where Δ is the amount of flow needed to saturate p. To apply the minimum cost augmentation strategy to the RDS problem, we seek an augmenting path from the source to a sink that has the smallest *incremental cost per unit flow* among all augmenting paths. An efficient way to instantiate this strategy is to choose a small set of increments, construct the tree corresponding to each increment, and find the best augmenting path from among this smaller set of trees. We have found that in practice, the best path is usually found in the tree corresponding to the largest increment. This observation has led us to the LDF algorithm below.

$f := 0$;
while there is unmet demand at some sink
  Let Δ be the smaller of the largest unmet demand and the largest residual capacity among all augmenting paths.
  Let p be the augmenting path in $T_f(\Delta)$ with the smallest incremental cost per unit flow.
  Modify f by saturating p.
end

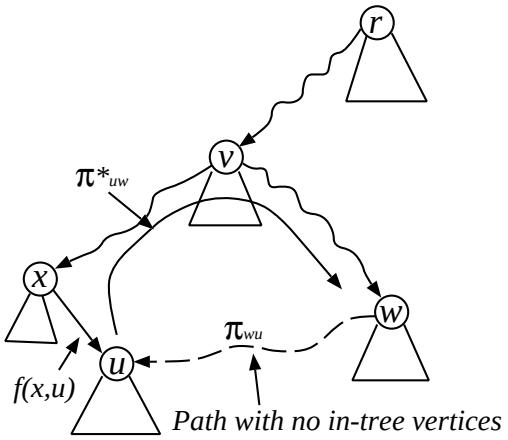

Fig. 1. Original fbw.

III. Local Search Improvements

Local search [4] is a well-known approximation method applicable to almost all combinatorial optimization problems. To study the performance of LDF, we apply local search algorithms to further improve the quality of LDF solutions. The key of a good local search algorithm is to find a viable simple operation that obtains a feasible flow from an existing one. The negative cost cycle reduction method is a natural candidate for such an operation. It works by "pushing" flows along a negative cost cycle to transform an existing feasible flow to another feasible flow with lower total cost.

A. Cycle Reduction Algorithm

The cycle reduction algorithm in [3] provides an effective way to implement local search with negative cost cycle reduction. The algorithm is based on the observation that a flow x' is adjacent to an existing flow x if and only if all edges that are in x' but not in x constitute a path connecting only two vertices in x.

Fig. 1 through Fig. 3 illustrate the basic operations of the Gallo-Sodini algorithm for finding a neighboring flow (Fig. 2) from an existing extreme flow (Fig. 1). For every vertex u with an incoming flow of $f(x, u)$ in the tree defined by the existing flow, let π^*_{uv} be the undirectional path from u to any other tree vertex v that is not in the subtree rooted at u. As shown in Fig. 1, π^*_{uv} normally has two directed parts, namely, the path from v to u and the path from v to w, where v is the nearest common ancestor of u and w. Compute the incremental cost c^u_v of adding $f(x, u)$ units of flow on π^*_{uv}. c^u_v has two component: one corresponds to the cost decrements on the path from v to u due to the flow reduction of $f(x, u)$, and the

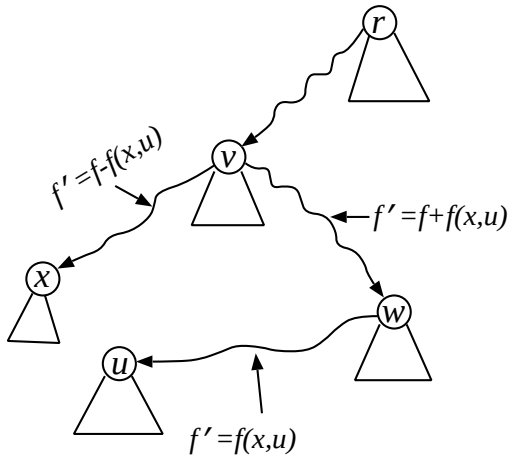

Fig. 2. The neighboring fbw obtained with the cycle reduction algorithm.

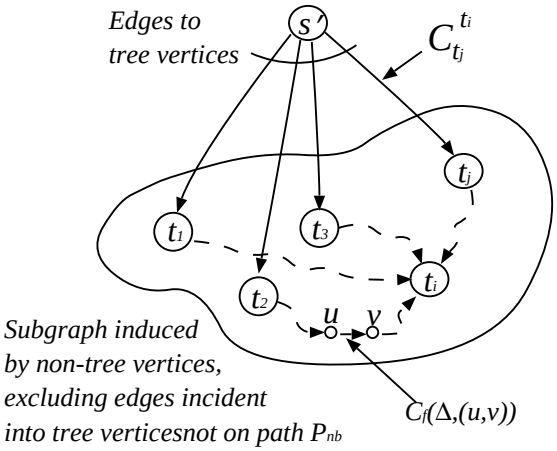

Fig. 3. Finding the best solution for "target" t_i.

other one corresponds to the cost increments on the path from v to w due to the flow increment of $f(x,u)$.

Next, a new network is derived for a specific tree vertex t_i as shown in Fig. 3. In this transformed network, a pseudo source vertex s' is added to the subgraph induced by the non-tree edges, and s' is connected to every tree vertex t_j with a direct edge (s', t_j), except for t_i. All edges incident to any tree vertex except t_i are removed too. If we define Δ to be the flow into t_i, an edge (s', t_j) is assigned a length of $c_{t_j}^{t_i}$ as defined in the last step. A non-tree edge (u, v) is assigned a length of the same as the incremental cost of adding Δ units of flow on that edge, $C_f(\Delta,(u,v))$. We find the shortest path from s' to t_i in the transformed network. The last vertex w on the shortest path to t_i is identified, and Δ units of flow are redirected along the undirectional path $\pi^*_{t_i w}$ and then along the directed path π_{wt_i}. The result is a neighboring flow to the original flow (Fig. 2). If the modified flow has a lower cost, the above procedure is repeated; otherwise, the original flow is restored, and the algorithm halts.

It is easy to see that the cycle reduction algorithm reduces

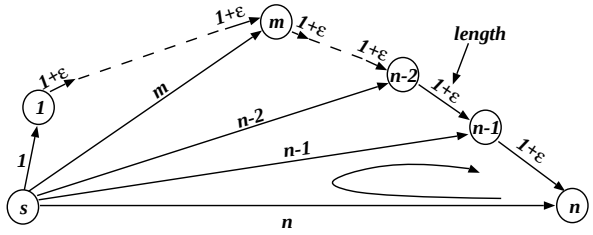

Fig. 4. A simple network that will benefi t from the cycle reduction algorithm.

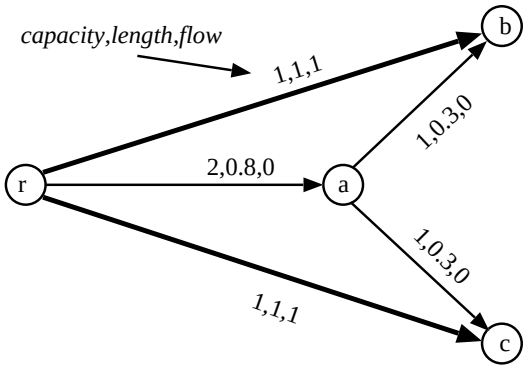

Fig. 5. A simple negative cost bicycle example.

the cost by redirecting flow along negative cost cycles. This local search algorithm can greatly improve the quality of solutions obtained from LDF. Take the network in Fig. 4 for example. The source s connects to all n sinks with unit demand. LDF picks only the direct links from s to all sinks, resulting in a suboptimal solution with no bandwidth sharing. The cycle reduction algorithm identifies the negative cycles in the LDF solution, redirects the flow along these cycles, and eventually finds the single path $s \to 1 \cdots \to n-1 \to n$ as the solution, which is the optimal solution in this case. With a small ϵ, the improved solution is $O(n)$ times better than the original one.

B. Bicycle Reduction Algorithm

We observe that the result from the cycle reduction algorithm does not necessarily include all possible neighboring flows with lower costs, and can be sub-optimal in a network with concave edge costs. Consider the example network shown in Fig. 5(a), where r is the source, b and c are the sinks with unit demands. Currently, flows go directly from r to the sinks as it is the shortest path, with a total cost of 8. There is clearly no negative cost cycle in the residual graph, and the cycle reduction can not improve this flow. However, if we push 2 units of flow on (r, a), and a unit of flow along (a, b), (a, c), (b, r), and (c, r), we will get a flow with a lower cost, 7.36. We observe that these edges we use constitute a subnetwork with special structure that results in flows with lower costs.

We identify such a subgraph structure as a *negative cost bicycle*. In general, a negative cost bicycle is defined as a pair of directed cycles that share a common segment, with

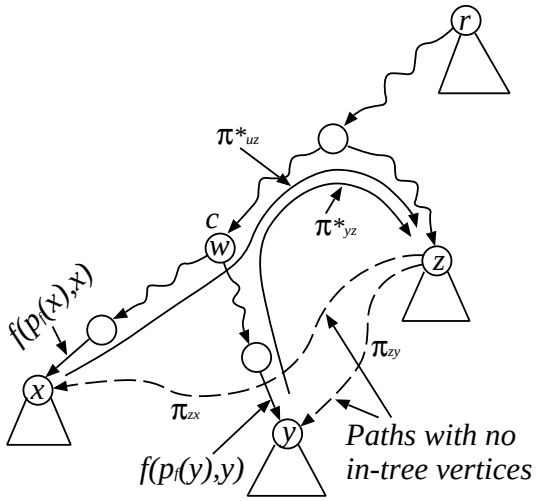

Fig. 6. Original fbw.

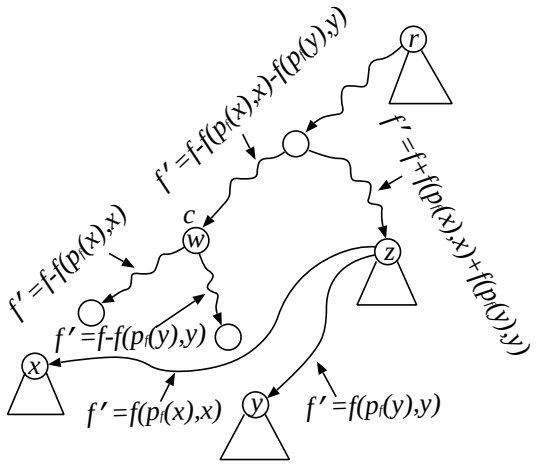

Fig. 7. The neighboring fbw obtained by the bicycle reduction.

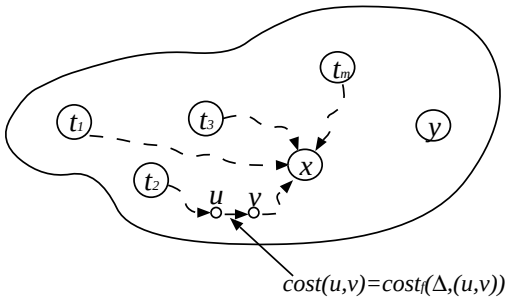

Fig. 8. G_x.

the remainder of the cycles edge disjoint. When we add flow along the two cycles, the total cost of the resulting flow is lower than the original flow. Let P_0 be the common path, P_1 and P_2 be the disjoint segments, and d_i be the incremental cost for a unit of flow increment on P_i. The incremental cost of adding a unit flow along the bicycle could be expressed as $(1+\epsilon)d_0 + d_1 + d_2$. For $0 < \epsilon < 1$, it results in a lower cost than the sum of costs on all paths, showing the benefits of path sharing.

Based on the observation of negative cost bicycles, we present a local search algorithm using bicycle reduction to improve the solutions from the cycle reduction algorithm. We start with two vertices, x and y, in the tree defined by the existing flow, and search for a *optimal split point* z, through which there is a pair of directed non-tree paths π_{zx} and π_{zy}. In addition, there is a *optimal merge point* w on both undirectional tree paths π^*_{xz} and π^*_{yz}. The tree path π^*_{wz} is the common segment of the bicycle, and the two cycles are $(\pi^*_{xw}, \pi^*_{wz}, \pi_{zx})$ and $(\pi^*_{yw}, \pi^*_{wz}, \pi_{zy})$. For example, let f_x and f_y be the flow into x and y respectively, after redirecting f_x along $(\pi^*_{xw}, \pi^*_{wz}, \pi_{zx})$ and f_y along $(\pi^*_{yw}, \pi^*_{wz}, \pi_{zy})$, we can obtain a neighboring flow (Fig. 7) from an existing flow (Fig. 6) with the bicycle reduction algorithm.

First, compute the incremental cost of redirecting f_x units of flow from x and f_y units of flow from y to all potential split points along undirectional paths in the existing flow. These vertices include all tree vertices except for those in the subtree rooted at x or y. The total incremental cost c^*_u for such a vertex u is the sum of the incremental costs from x and y to u.

Next, make two copies G_x (Fig.8) and G_y (Fig.9) of the subgraph induced by the zero-flow edges in G. In G_x, remove edges originating from the subtrees rooted at x or y, and edges incident into tree vertices except x. Compute the incremental cost c_{uv} for an edge (u, v) in G_x as adding f_x units of flow on (u, v). The shortest paths from all vertices in G_x are computed using c_{uv} as the length of edge (u, v). Similarly, G_y is generated, and the shortest paths from all vertices to y are computed using flow increment f_y. For each vertex u, record the shortest paths π_{ux} in G_x and π_{uy} in G_y as well as the shortest distance c_{ux} and c_{uy}. For any tree vertex u other than x and y, define the total cost $C_u = c_{ux} + c_{uy} + c^*_u$, and find the vertex z with the minimum total cost as the optimal split point. Redirect f_x flow along π_{zx} and π^*_{xz}, and f_y flow along π_{zy} and π^*_{yz}. If the updated flow has a lower cost, repeat the above procedure until no improvements can be made.

IV. Simulation Studies

Both the local search algorithms with cycle and bicycle reduction have greater computational complexity than LDF, because they could have exponential number of iterations. To study the solution quality improvements by these two local search algorithms, we evaluated both local search algorithms on a number of network topologies. We only present some representative results on random networks due to the limited space. More results of these local search algorithms can be found in [5]. The vertices and edges of the network are uniform randomly generated inside a unit square, with the source node located in the corner. The sinks are randomly chosen in the unit square with uniform random sink demands drawn from the range of $[1, 10]$.

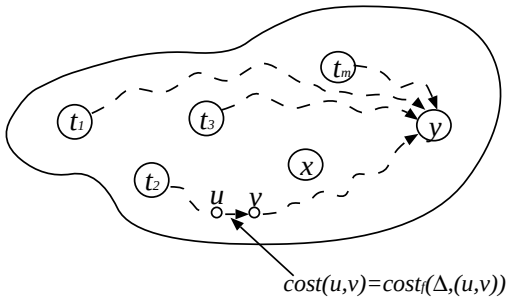

Fig. 9. G_y.

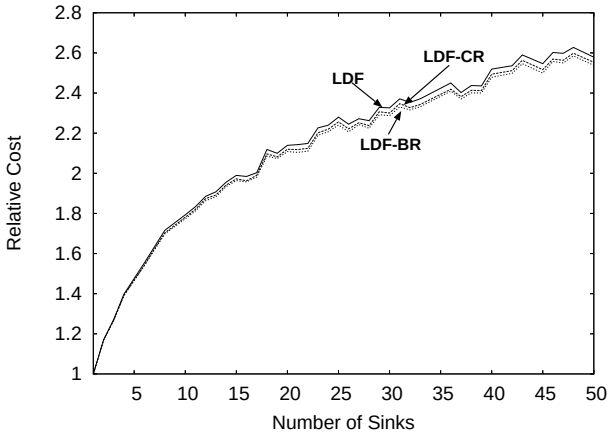

Fig. 10. Relative cost comparison on random networks.

We measure the relative costs of flows generated by different algorithms with an estimated lower bound as defined in [1]. We first obtain an initial solution with LDF, and apply the cycle reduction algorithm as well as the bicycle reduction algorithm to the flow produced by LDF. The number of sinks is varied to show how the results scale. We also measure the percentage improvements made to the LDF solutions.

The relative cost results (Fig. 10) show that LDF produces results up to 2.6 times the estimated lower bound, while the cycle reduction (LDF-CR) and bicycle reduction algorithm (LDF-BR) both improve the quality of the LDF results, and LDF-BR provides more consistent improvement to the LDF-CR solutions. However, the degrees of improvements are not significant as expected. As Fig. 11 shows more clearly, the cycle reduction algorithm (CR) improves the solution quality by up to 1.2% over LDF, and bicycle reduction (BR) improves the solution quality by up to 1.85% over LDF. In general, the bicycle reduction algorithm offers up to twice the improvement of the cycle reduction algorithm in most cases, but they are relatively small improvements for the additional complexity in the cycle and bicycle reduction algorithms.

Our analysis in the previous sections illustrates that the cycle and bicycle reduction algorithms can greatly improve the quality of LDF solutions in certain example networks. However, our simulation results show a small improvement

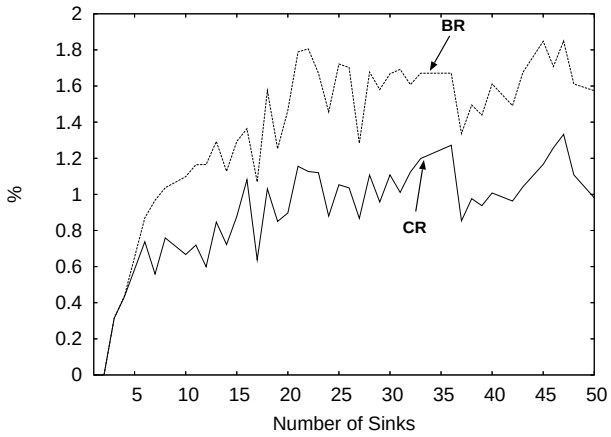

Fig. 11. Percentage cost improvement on random networks.

for average cases of the random networks generated. These results indicate that LDF offers good solutions without added computational complexity from the cycle and bicycle reduction algorithms. Although the improvements by cycle and bicycle reduction algorithms can be substantial in certain networks, and can not be ruled out entirely, they offer little improvements in average cases. This indicates that the special negative cycle and bicycle subnetwork structures do not occur frequently in the LDF solutions from the network topologies we simulated. While LDF is a polynomial approximation algorithm, both cycle and bicycle reduction algorithms run in exponential time. Thus, these results further prove that LDF can serve as a good approximation algorithm for the configuration of an RDS with large number of sink nodes in practice.

V. Conclusions

In this paper, we apply local search algorithms with cycle and bicycle reduction to improve the solution quality of the LDF algorithm for the RDS configuration problem. Our experimental results show that the both local search algorithms can improve the quality of results, but it would reach a point of diminishing return as the quality improvement is limited but the computational complexity grows. Meanwhile, LDF produces solutions that are close to the local optima, without added computational complexity.

References

[1] R. Qiu and J. S. Turner, "Confi guration of reserved delivery subnetworks," in *Proceedings of IEEE Globecom*, (Taipei, Taiwan), Novmber 2002.

[2] G. M. Guisewite and P. M. Pardalos, "Minimum concave-cost network fbw problems: Applications, complexity, and algorithms," *Annals of Operations Research*, vol. 25, pp. 75–99, 1990.

[3] G. Gallo and C. Sodini, "Adjacent extreme fbws and application to min concave cost fbw problems," *Networks*, vol. 9, pp. 95–121, 1979.

[4] E. Aarts and J. K. Lentra, *Local Search in Combinatorial Optimization*. John Wiley & Sons, 1997.

[5] R. Qiu and J. S. Turner, "Improved local search algorithm with multi-cycle reduction for minimum concave cost network fbw problems," tech. rep., WUCS-04-74, Washington University at St. Louis, Department of Computer Science and Engineering, 2004.