Multicommodity Flows in Cycle Graphs[1)]

Young-Soo Myung
Department of Business Administration,
Dankook University, Cheonan, myung@dankook.ac.kr

Abstract

This paper considers the multicommodity flow problem and the integer multicommodity flow problem on cycle graphs. We present two linear time algorithms for solving each of the two problems.

1. Introduction

We are given an undirected graph G=(V,E) and a set of source-sink pairs (s_i,t_i), i=1,...,|K|. Let K={1,...,|K|} be the index set of source-sink pairs. For each edge e∈E, a nonnegative capacity c(e) is given and for each source-sink pair (s_i,t_i), i∈K, a nonnegative demand d(i) is given. A multicommodity flow is a function f on |K| where f_i is a flow from s_i to t_i for each i∈K. Given c and d, we will say a multicommodity flow f is feasible, if the value of f_i is d(i) for each i∈K and $\Sigma_{i\in K} f_i(e) \le c(e)$ for each e∈E. Then the multicommodity flow problem (MFP) is to find a feasible multicommodity flow. If we require an integer flow, the problem is called the integer multicommodity flow problem (IMFP). In the IMFP, we assume that the capacities and demands are all integers. Throughout, we use the terms multicommodity flow when no integrality is required. For more details on the definition of multicommodity flow problems, refer to Schrijver (2003).

The MFP and the IMFP are important from both the theoretical and practical viewpoints. Both problems are well known topics in combinatorial optimization and have good applications such as routing problems in telecommunication networks and the design of VLSI circuits. (See Schrijver (2003) and Suzuki et al. (1992)). In this paper, we consider the MFP and the IMFP on cycle graphs. Suzuki et al. (1992) have developed two algorithms, one for testing the feasibility of the multicommodity problem on cycle graphs and the other for finding a feasible flow in a feasible problem. Both algorithms run in linear time. It is known that if the capacities and demands are all integers and a feasible multicommodity flow exists, then a half-integer multicommodity flow always exists on cycle graphs (Okamura and Seymour (1981)). For the IMFP on cycles, Frank et al. (1992) have presented a linear time algorithm when all demands are equal to 1. A direct application of their algorithm results in a pseudopolynomial algorithm for a general instance of the IMFP on cycle graphs. From now on, if we do not specify a given graph, we assume it as a cycle graph.

Our objective of this study is to develop two linear time algorithms, one for solving the MFP and the other for solving the IMFP (with arbitrary demand). The paper is organized as follows. Section 2 introduces the notation and definitions. We present an algorithm that solves the MFP in Section 3 and an algorithm for the IMFP in Section 4. This paper is the conference version of the full paper in Myung (2006).

2. Notation and definitions

We mean a cycle graph as an undirected graph G=(V,E) with a node set V={1,2,...,n} and an edge set E={(1,2),(2,3),...,(n-1,n),(n,1)}. We use e_i to denote an edge (i,i+1) for i=1,...,n, where indices are counted modulo n. We order the edges such that $e_i<e_j$ if i<j. As previously defined, K is the index set of source-sink pairs. We assume that $s_i<t_i$ for each i∈K. The demand between s_i and t_i can be routed in either of the two directions, clockwise and counterclockwise. We say that a flow is routed in the *clockwise* (*counterclockwise*) direction if a flow passes through the node sequence

1) This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2005-041-B00167).

$\{s_i,s_i+1,...,t_i-1,t_i\}$ ($\{s_i,s_i-1, ..., 1,n,...,t_i+1,t_i\}$). For ease of description, we assume that nodes 1,2,...,n appear clockwise on G in this order. We also assume that $s_1 \le s_2 \le ... \le s_{|K|}$ and that each node is contained at least one source-sink pair, i.e., $2|K| \ge n$. Notice that we can make a given graph meet the condition of the second assumption by simply removing the nodes not contained in any source-sink pair.

For each (s_k,t_k), $k \in K$, let $E_k^+(E_k^-)$ denote the set of edges contained in the clockwise (counterclockwise) direction path from s_k to t_k. Note that edge e_n is contained in E_k^- for all $k \in K$. As a flow between a source-sink pair can be routed in only two directions, one variable is enough to represent the flow. For each $k \in K$, let's define variable $x(k)$ that denotes the amount of the total demand between s_k and t_k routed in the clockwise direction. Therefore, $d(k)-x(k)$ is the amount of the flow routed in the counterclockwise direction. Let $X=\{x \in R^{|K|} \mid 0 \le x(k) \le d(k) \forall k \in K\}$, and for a given multicommodity flow $x \in X$ and $e \in E$, let

$$g(x,e)=\sum\{x(k) \mid e \in E_k^+\}+\sum\{d(k)-x(k) \mid e \in E_k^-\}$$

Then g(x,e) denotes the sum of the flows routed through edge e. Therefore, $x \in X$ is a feasible multicommodity flow if $g(x,e) \le c(e)$ for each $e \in E$.

Any pair of distinct edges e and f constitute a cut and for each cut {e,f}, we define D(e,f) as

$$D(e,f)=\sum\{d(k): either\ e \in E_k^+, f \in E_k^-,\ or\ e \in E_k^-, f \in E_k^+\}.$$

Therefore, D(e,f) can be interpreted as the total demand across the cut {e,f}. If a multicommodity flow $x \in X$ is given, D(e,f) can be expressed with respect to x. Note that a flow between a source-sink pair whose demand is across the cut, is routed through either e or f and a flow between the remaining pairs, is routed through both e and f, or neither of the edges. Therefore, for a given $x \in X$, D(e,f) can be expressed as follows.

$$D(e,f)= g(x,e)+ g(x,f)-2\sum\{x(k) \mid e,f \in E_k^+\} -2\sum\{d(k)-x(k) \mid e,f \in E_k^-\}. \quad (1)$$

From (1), we can know the following facts.

Remark 1 (i) For any cut {e,f}, $g(x,e)+g(x,f) \ge D(e,f)$.
(ii) $D(e,f)=g(x,e)+g(x,f)$ if and only if e and f satisfy the following two conditions:
(C1) $x(k)=0$ for each $k \in K$ such that $e,f \in E_k^+$.
(C2) $x(k)=d(k)$ for each $k \in K$ such that $e,f \in E_k^-$.

The statement (i) of Remark 1 indicates that every feasible multicommodity flow $x \in X$ satisfies the following cut condition.

$$c(e)+c(f) \ge D(e,f), \quad \text{for each } e,f \in E. \quad (2)$$

Moreover, as a direct consequence of Okamura and Seymour's theorem (Okamura and Seymour (1981)), we know that in a cycle graph, the MFP has a solution if and only if the graph satisfies the cut condition.

3. Algorithm for the MFP

In this section, we present an algorithm of solving the MFP. For the MFP, Suzuki et al. (1992) have developed two linear time algorithms, one for testing the feasibility of the MFP and the other for finding a feasible flow in a feasible problem. Without a complex data structure, the first algorithm runs in $O(n|K|)$ time and the latter in $O(n|K|^2)$ time. Using Gabow and Tarjan's data structure (Gabow and Tarjan (1985)), both algorithms can be implemented in $O(|K|)$ time.

Here, we develop an algorithm that finds a feasible solution of the MFP, or verifies that the problem has no solution. Our algorithm is the modification of Myung, Kim, and Tcha's algorithm of solving the ring loading problem (RLP) on an undirected ring (cycle) network in Myung et al. (1997). In the RLP, each edge is required to have the same capacity and the objective is to find the smallest edge capacity that enables the existence of a feasible multicommodity flow. Myung et al.'s algorithm is very simple and can be implemented in linear time. Our algorithm is very similar to theirs and has the same computational complexity. In order to run in $O(|K|)$ time, our algorithm also rely on Gabow and Tarjan's data structure but without this, our algorithm runs in $O(n^2)$ time. The key feature of our algorithm is that it does not have to compute c(e)+c(f)-D(e,f) for each cut, that gives major computational burden to Suzuki et al.'s algorithm.

To present our algorithm, we define $m(x,e)=g(x,e)-c(e)$ and $mmax(x)=\max_{\{e \in E\}} m(x,e)$ for a given $x \in X$. Notice that a flow $x \in X$ is a feasible flow if $mmax(x) \le 0$. Our algorithm

finds a feasible flow for the MFP, if any, otherwise verifies that no such solution exists. Our algorithm initially set x=(x(1),x(2),...,x(|K|))= (d(1),d(2),...,d(|K|)), which means that all demands are routed in the clockwise direction. Then, for each source-sink pairs k∈K, the algorithm tries to reduce mmax(x) by rerouting all or a part of d(k) in the counter-clockwise direction. Note that if max{m(x,e)|e∈ E_k^+}> max{m(x,e)|e∈ E_k^-}, rerouting d(k) in the counterclockwise direction decreases mmax(x). The rerouting of demand k is done until all the demand is rerouted or the resulting flow satisfies max{m(x,e)|e∈ E_k^+}=max{m(x,e)|e∈ E_k^-}. We don't have to continue iteration, after we find a feasible flow, but we set our algorithm as one doing all of |K| iterations. This is simply for expositional convenience. Under this setting, the algorithm finds a flow x∈X with minimum mmax(x). Our algorithm is formally described as follows.

Algorithm FLOW

begin
 x←(d(1),d(2),...,d(|K|)).
 for each source-sink pairs (s_k,t_k), k=1,2,...,|K|,**do**
 begin
  m(E_k^+)← max{m(x,e)|e∈ E_k^+}
  m(E_k^-)← max{m(x,e)|e∈ E_k^-}
  if m(E_k^+)>m(E_k^-) **then**
  δ←min{(m(E_k^+)-m(E_k^-))/2,d(k)}
  else δ←0
  x(k)←d(k)-δ
 end
end

Now we show the correctness of the algorithm FLOW. It is straightforward that if mmax(x)≤0 for the returned solution x, it is a feasible flow. The remaining thing is to show that if mmax(x)>0, the given problem has no feasible flow. Let E(x)={e∈E|m(x,e)=mmax(x)} for a given x∈X. E(x) means the set of edges whose flow exceeds the capacity most with respect to x. We will call the edges in E(x) *the most violated edges* with respect to x. Let x^0={d(1),...,d(|K|)} and x^k denote the solution obtained after the rerouting step is performed for k∈K. For example, $x^{|K|}$ is the solution that the algorithm finally produces. We will show that if mmax($x^{|K|}$)>0, a given graph does not satisfy the cut condition, (2), which means that a given graph has no feasible flow. We formally state it.

Theorem 2 If mmax($x^{|K|}$)>0, there always exist a pair of most violated edges that satisfy (C1) and (C2) in Remark 1, i.e., a given graph does not satisfy the cut condition.

Proof. See Myung (2006).

The algorithm FLOW is a modification of Myung, Kim, and Tcha's algorithm that appeared in Myung et al. (1997) to solve the ring loading problem on undirected cycle graphs. The computational complexity of the algorithm FLOW is the computation of m(E_k^+) and m(E_k^-) for each k∈K. It is not difficult to construct an algorithm that carries out each rerouting step in O(n) time and thus the whole steps in O(n|K|) time. Moreover, Myung and Kim (2004) have shown that the whole rerouting steps can be done in O(n^2) time and Wang (2005) have developed an improved algorithm to complete the whole steps in O(|K|) time using Gabow and Tarjan's data structure. Therefore, using there algorithms we can achieve the same time bound for FLOW.

4. Algorithm for the IMFP

In this section, we present an algorithm of solving the IMFP. This algorithm is similar to the one presented in Myung (2001) for solving an integer version of the ring loading problem on undirected cycle graphs. In the first phase, our algorithm implements the algorithm FLOW. Notice that if a graph does not have a feasible (fractional) multicommodity flow, it never has an integer one, but not vice versa. The second phase of our algorithm starts with a feasible solution of the MFP produced by the algorithm FLOW. If the solution is an integer one, we are done. When the solution is a fractional one, our algorithm checks whether a given graph has an integer multicommodity flow, and finds one, if any.

Suppose that $x^{|K|}$, produced by the algorithm FLOW, is a fractional solution. Let K^f={k_1,...,k_f} be the index set of the source-sink pairs in K for which $x^{|K|}$(k) has fractional value. As $K^f \subseteq K_b$, $s_{k_1} < s_{k_2} < \cdots < s_{k_f} < t_{k_1} < \cdots < t_{k_f}$. Let's partition E into the following 2f+1 subsets as follows: $L_0 = \{e_i \mid 1 \le i < s_{k_1}\}$, $L_1 = \{e_i \mid s_{k_1} \le i < s_{k_2}\}$, ..., $L_f = \{e_i \mid s_{k_f} \le i < t_{k_1}\}$, $L_{2f} = \{e_i \mid t_{k_f} \le i \le n\}$. Let e_s=min E(x^0), e_{min}=min E($x^{|K|}$) and e_{max}=max E($x^{|K|}$). In other words, e_s is the edge having the smallest index among the most violated edges

with respect to the initial solution and $e_{min}(e_{max})$ is the edge having the smallest (largest) index among the most violated edges with respect to $x^{|K|}$. Note that e_s remains as a maximum load edge with respect to x^k for each k=1,2,...,|K|, but may or may not be e_{min}.

If we consider the rerouting procedure of the algorithm FLOW, it is not difficult to know that the fractional part of $x^{|K|}(k)$ for each $k \in K^f$ is equal to 0.5. So, if we reroute each demand $k \in K^f$ by 0.5 in either a clockwise or a counterclockwise direction, we can obtain an integer flow. We define the two different rerouting methods, *Method A* and *Method B*. Both methods reroute each demand $k \in K^f$ by the amount of 0.5, in the increasing order of $k \in K^f$ and in either of the two directions alternatingly, one after another. The difference of the two methods is that Method A starts iteration by rerouting the first flow in the clockwise direction while Method B starts rerouting in the counterclockwise direction. Let x* be the integer solution obtained by rerouting the fractional flows of $x^{|K|}$ using either Method A or Method B. Then the following observations are useful to develop an algorithm of solving the IMFP.

Remark 4 (a) If $|K^f|$ is odd, $g(x^{|K|},e)$, $e \in E$ has fractional value whose fractional part is 0.5 and if $|K^f|$ is even, $g(x^{|K|},e)$, $e \in E$ has integer value.

(b) If $|K^f|$ is odd and x* is the integer solution obtained by rerouting $x^{|K|}$ using either Method A or Method B, then for each $e \in E$,

$$g(x^{|K|},e)-0.5 \le g(x^*,e) \le g(x^{|K|},e)+0.5.$$

(c) If $|K^f|$ is even and x* is the result of Method A,

$$g(x^*,e)=\begin{cases} g(x^{|K|},e)+1, & e \in L_1 \cup L_3 \cup \cdots \cup L_{f-1}, \\ g(x^{|K|},e), & e \in L_0 \cup L_2 \cup \cdots \cup L_f \cup \cdots \cup L_{2f}, \\ g(x^{|K|},e)-1, & e \in L_{f+1} \cup L_{f+3} \cup \cdots \cup L_{2f-1}. \end{cases}$$

(d) If $|K^f|$ is even and x* is the result of Method B,

$$g(x^*,e)=\begin{cases} g(x^{|K|},e)+1, & e \in L_{f+1} \cup L_{f+3} \cup \cdots \cup L_{2f-1}, \\ g(x^{|K|},e), & e \in L_0 \cup L_2 \cup \cdots \cup L_f \cup \cdots \cup L_{2f}, \\ g(x^{|K|},e)-1, & e \in L_1 \cup L_3 \cup \cdots \cup L_{f-1}. \end{cases}$$

Now we present an algorithm that finds a feasible integer flow, or verifies that no such solution exists.

Algorithm INTEGER

(Step 1) Implement the algorithm FLOW. If $\mathrm{mmax}(x^{|K|})>0$, then the IMFP has no solution. If $\mathrm{mmax}(x^{|K|})\le 0$ and $K^f=\varnothing$, $x^{|K|}$ is a feasible solution. If $\mathrm{mmax}(x^{|K|})\le 0$ and $K^f\neq\varnothing$, go to Step 2.
(Step 2) If either $|K^f|$ is odd or $\mathrm{mmax}(x^{|K|}\})\le -1$, reroute $x^{|K|}$ using Method A. Otherwise, go to Step 3.
(Step 3) If there exists no most violated edge e such that $e<e_s$, then reroute $x^{|K|}$ using Method A. Otherwise, go to Step 4.
(Step 4) Depending on the situations, do the following.
(4-1) If every most violated edge e such that $e<e_s$, exists only in the even indexed sets, i.e., $e \in L_0 \cup L_2 \cup \cdots \cup L_f$, then reroute $x^{|K|}$ using Method A.
(4-2) If every most violated edge e with $e<e_s$, exists only in the odd indexed sets, i.e., $e \in L_1 \cup L_3 \cup \cdots \cup L_{f-1}$, then reroute $x^{|K|}$ using Method B. Select $k \in K$ such that e_s, $e_{max} \in E_k^+$ and $x^{|K|}(k)>0$. If such k exists, reroute one unit of demand k in the counterclockwise direction. If no such k exists, then the IMFP has no solution.
(4-3)} If there exist a pair of most violated edges {e,f} such that $e,f<e_s$ and that e belongs to the even indexed sets and f belongs to the odd indexed sets, then the IMFP has no solution.

Theorem 5 The algorithm INTEGER correctly solves the IMFP.

Proof. See Myung (2006).

As Myung (2001) and Wang (2005) have explained, the computational complexity of the algorithm INTEGER is not more than the algorithm FLOW. Therefore, INTEGER can be implemented in linear time.

References

A. Frank, T. Nishizeki, N.Saito, H. Suzuki and E. Tardos (1992), Algorithms for routing around a rectangle, Discrete Applied

Mathematics 40, 363-378.

H. N. Gabow and R. E. Tarjan (1985), A linear time algorithms for a special case of disjoint set union, J. Comput. System Sci. 30, 209-221.

Y.-S. Myung, H.-G. Kim and D.-W. Tcha (1997), Optimal load balancing on SONET bidirectional rings, Operations Research 45, 148-152.

Y.-S. Myung (2001), An efficient algorithm for the ring loading problem with integer demand splitting, SIAM J. Discrete Mathematics 14, 291-298.

Y.-S. Myung and H.-G. Kim (2004), On the ring loading problem with demand splitting, Operations Research Letters 32, 167-173.

Y.-S. Myung (2006), Multicommodity Flows in Cycle Graphs, To appear in Discrete Applied Mathematics.

H. Okamura and P. D. Seymour (1981), Multicommodity flows in planar graphs, Journal of Combinatorial Theory Series B 31, 75-81.

A. Schrijver, P. Seymour and P. Winkler (1998), The ring loading problem, SIAM J. Discrete Math., 11, 1-14.

A. Schrijver (2003), Combinatorial Optimization, Springer, Berlin.

H. Suzuki, A. Ishiguro, and T. Nishizeki (1992), Variable-priority queue and doghnut routing, Journal of Algorithms 13, 606-635.

B.-F. Wang (2005), Linear time algorithm for the ring loading problem with demand splitting, Journal of Algorithms 54, 45-57.