A Simultaneous Maximum Flow Algorithm for the Selection Model

Bin Zhang, Julie Ward, Annabelle Feng
Intelligent Enterprise Technologies Laboratory
HP Laboratories Palo Alto
HPL-2005-91
May 13, 2005*

Maximum Flow, parametric flow networks, graphs, optimization, selection, sequencing

A new algorithm, $SPMF^{simple}$, for finding the complete chain of solutions of the product selection model is presented in this report. λ-directed simple residual path is identified to the only kind of residual path necessary for the new algorithm. By augmenting the right amount of flows along λ-directed simple residual paths, the new algorithm is monotone convergent.

A Simultaneous Maximum Flow Algorithm for the Selection Model

Bin Zhang, Julie Ward, Annabelle Feng

IETL/HPL

Abstract. A new algorithm, SPMF^{simple}, for finding the complete chain of solutions of the product selection model is presented in this report. λ-directed simple residual path is identified to the only kind of residual path necessary for the new algorithm. By augmenting the right amount of flows along λ-directed simple residual paths, the new algorithm is monotone convergent.

(Note: This work was first presented at the INFORMS Annual Meeting in Denver, October 2004.)

1. Introduction

The *Selection Problem* is stated as follows: There are two sets of entities $P=\{p_i\}_{i=1}^{n_1}$ and $O=\{o_j\}_{j=1}^{n_2}$. Each $o \in O$ depends on a subset of *p's*, denoted by $P_o \subset P$. Let x_i and y_j be the indicator variables, taking values in *{0,1}*, associated with p_i and o_j respectively. A value $R_o \geq 0$ is associated with each $o \in O$. The selection problem is defined by the following integer programming model:

IP(n): $$\max \sum\nolimits_o R_o y_o \text{ s.t. } \sum\nolimits_p x_p \leq S,\ y_o \leq x_p \text{ for all } p \in P_o \text{ and } x_p, y_o \in \{0,1\}$$

Solving this integer program directly can be very difficult and costly. Replacing first constraint with a penalty term in the objective and relaxing the integrality requirement give us a linear program, the Langrangian Relaxation LR(λ), which depends on the penalty λ:

LR(λ): $$\max \sum\nolimits_o R_o y_o - \lambda \sum\nolimits_p x_p \text{ s.t. } y_o \leq x_p \text{ for all } p \in P_o \text{ and } 0 \leq x_p, y_o \leq 1$$

The problem LR(λ) for fixed λ is an example of a *selection problem* introduced independently by Balinski [1970] and Rhys [1970]. Balinski (1970) showed that a selection problem is equivalent to the problem of finding a minimum cut in a particular bipartite network, illustrated in Figure 1. There is a source node *s* at the far left and a sink node *t* at the far right. Adjacent to the source node is a set of *p*-nodes. Adjacent to the sink node is a set of *o*-nodes. The capacity of the arcs adjacent to *s* is λ. The capacity of the arc from *o* to *t* is R_o. The capacity of arcs between a *p*-node and a *o*-node is infinite.

A *st*-cut is a partition of the nodes into two subsets – the *s*-partition containing *s* and the *t*-partition containing *t*. The capacity of a *st*-cut is the sum of the capacities of arcs going from nodes in the *s*-partition to nodes in *t*-partition. A minimum cut is a *st*-cut with minimum capacity.

The equivalence is established by observing that $\min \sum\nolimits_o R_o (1-y_o) + \lambda \sum\nolimits_p x_p$ is the same as $\max \sum_o R_o y_o - \lambda \sum_p x_p$.

It is a well-known result of Ford and Fulkerson that the value of a maximal flow equals the value of a minimum cut. Moreover, the minimum cut can be obtained by finding a maximal flow.

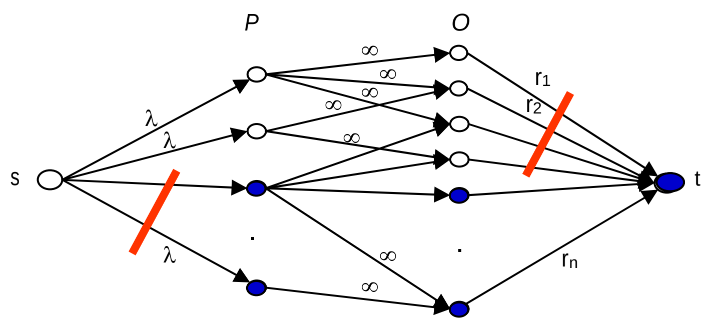

Figure 1. A bipartite minimum-cut/maximum flow problem corresponding to the Lagrangian relaxation LR(λ).

As we allow λ to vary, the problem LR(λ) becomes a parametric maximum flow problem, since the arc capacities depend on a parameter. There are several known algorithms for parametric maximum flow problems, including that of Gallo, Grigoriadis and Tarjan [1986]. In most of these algorithms, a series of maximum flow problems is solved, and the algorithm makes use of the previous problem's solution to speed up the solution at the next parameter value. By comparison, the algorithm presented in the next section finds the maximum flow in the network for all *breakpoints* of the parameter values simultaneously.

2. A Parametric Bipartite Maximum Flow Algorithm

A simple version of the simultaneous parametric maximum flow ($SPMF^{simple}$) algorithm is presented in this section. A more general version of this algorithm that applies to more general capacity constraints and more general parametric functions of parameter-dependent capacities can be found in (Zhang at el 2004).

$SPMF^{simple}$ works with a non-parametric network derived from the original parametric network. For the special case, the derivation is simply removing all the λ dependent capacities on the arcs incident to the source by positive infinity, which is the first step of the algorithm.

The second step initializes the flows in the derived network. The initial flows in the derived network are set to fill up all arcs incident to the target. Since all other arcs not incident to the target has +infinity capacity, this step is straightforward and does not require any special algorithm. After initialization, the total flow through the network remains fixed forever, which is always equal to the total flow to the target right after the initialization. Different initializations may have an impact on the amount of time the algorithm will take but it will not have any impact on the correctness of the algorithm.

The third step is the main body of the $SPMF^{simple}$ algorithm. It redistributes the flows through the arcs incident to the source in a regulated way which in the end will result in a *special state* of the flows in the derived network, from which all minimum cuts and their associated maximum flows at all breakpoints of the parameter λ, in the original network, can be found from one linear scan of the vertices and the arcs in the derived network.. Figure 2 shows the derived network.

In the derived network, we define $\lambda_i \triangleq f_{s,p_i}$. A residual path is called a λ-directed simple residual path if it starts from the source, containing a simple loop as $s \rightarrow p_i \rightarrow o \rightarrow p_j \rightarrow s$ with $\lambda_i = f_{s,p_i} < \lambda_j = f_{s,p_j}$ and $f_{p_j,o} > 0$.

The rules for redistributing the flows are

a) flows are augmented to only λ-directed simple residual paths

b) the amount of flow to augment to a λ-directed simple residual path is the minimum of the residue capacity of the path and $(f_{s,p_j} - f_{s,p_i})/2$.

The redistribution of the flow continues as long as there are λ-directed simple residual paths in the network. From the rules, it is obvious that the order of the two λ-values involved in the residual path of the current operation is never reversed after augmenting the flow.

Three results are proven: 1. The algorithm is monotone convergent. 2. The converged flows gives a special state of the flows in the derive network which allows us to read all minimum cuts and their associated maximum flows in the original network under any breakpoint λ-value in a linear scan of the vertices and arcs.

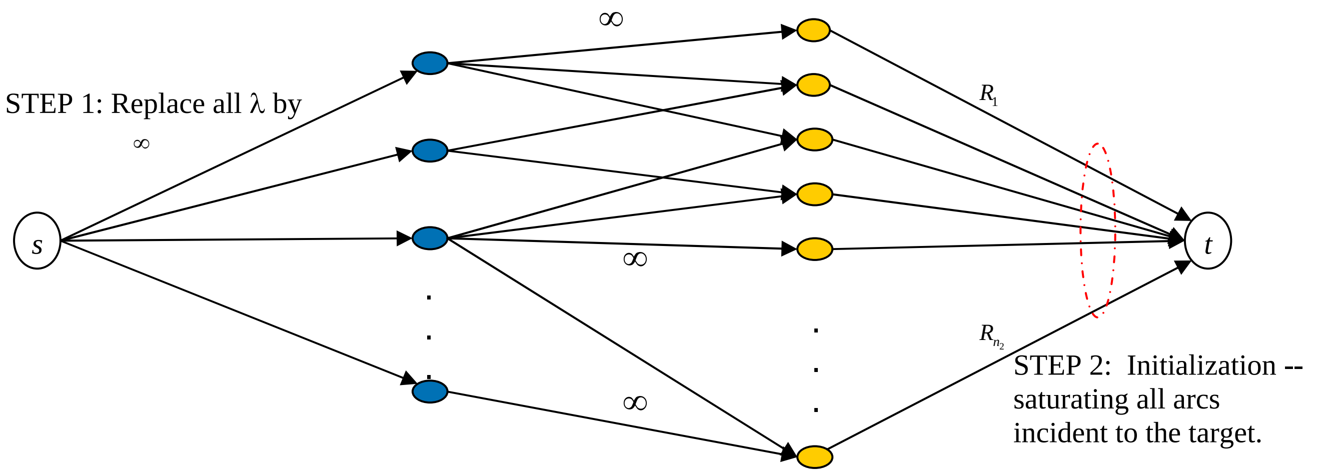

Figure 2. The derived non-parametric network.

3. Proof of Correctness

First we prove the monotone convergence property.

Theorem 1: The value $\sum_{i=1}^{n_1} \lambda_i^2$ decreases after each redistribution operation.

Proof: Each redistribution operation changes only two λ-values involved with the λ-directed simple residual path, $\lambda_i = f_{s,p_i}$ and $\lambda_j = f_{s,p_j}$. All other λ-values remain unchanged. The sum of all λ-values, which is equal to the total flow through the network, also remains unchanged. Therefore the sum $\lambda_i + \lambda_j$ remains unchanged but their difference $\lambda_j - \lambda_i$ becomes strictly smaller or zero after the redistribution operation (following rule b). $\lambda_i^2 + \lambda_j^2 = [(\lambda_i + \lambda_j)^2 + (\lambda_j - \lambda_i)^2]/2$ becomes strictly smaller. •

Theorem 2: For any λ, $P_{\lambda,t} = \{p \mid f_{s \to p} \geq \lambda\} \cup O_\lambda = \{o \mid (p \to o) \Rightarrow p \in P_\lambda\} \cup \{t\}$ gives the *t*-partition of the minimum cut of the original network Ω_λ.

Proof: Putting all the capacity bounds λ back to the arcs incident to the source in the derived network and reduce the flows that violate the bound and rebalance the flows at all the vertices where the conservation of the flows are broken by this reduction. The rebalancing cascades through the network from *p*-vertices to *o*-vertices.

When the rebalancing is done, the original network is recovered with a maximum flow and minimum cut.

All the arcs from the *o*-vertices in the *t*-partition to the *p*-vertices in the *s*-partition have zero flow guaranteed by the SPMF algorithm (otherwise SPMF would not have stopped). Figure 3 shows the recovered original network, which clearly shows that no augmenting path from the source to the target is left, therefore the flow is a maximum flow. •

(Note: The flow reduction and rebalancing in the last proof is only for the proof. Such steps are not needed in the implementation of $\text{SPMF}^{\text{simple}}$.)

By definition, *t*-partitions, under all breakpoint values of λ, for a monotone sequence of sets. A single scan of the *p*-vertices in either increasing or decreasing order of their associated λ-values will give all the minimum cuts. The associated maximum flows are calculated from the capacity of the minimum cuts shown in Figure 3, which is done by incremental computing along with the single scan of the *p*-vertices.

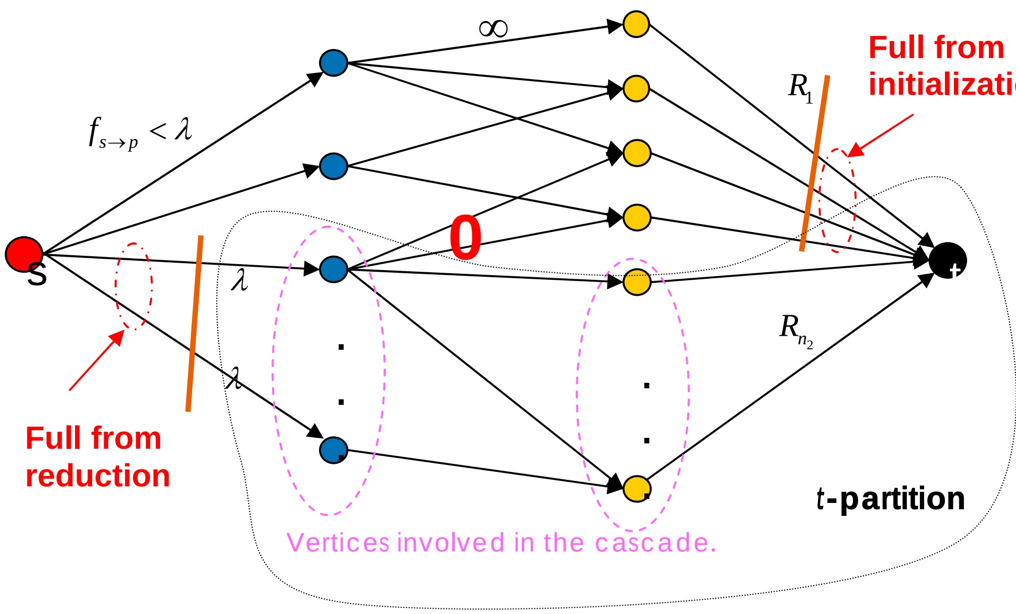

Figure 3. Recovering the original network with a maximum flow and a minimum cut from the derived network.

4. Implementation Details

After initialization, all *p*-vertices are queued in a FIFO queue. A procedure named process_one_p() is called on the *p* at the head of the queue. If any redistribution of the flows happened in the call, *p* is put back to the end of the queue, otherwise the *p* is deactivated. This process goes on until there is no more *p*'s in the queue. A deactivated *p* is reactivated when it is visited by process_one_p() in a redistribution operation.

process_one_p(*p*):

for each arcs *a* incident to *p*, get the *o* at the other end of *a*,

for each arc *b* incident to *o*, if *b* is not the same as *a*, get the p_1 at the other end of *b*,

if $s \to p \to o \to p_1 \to s$ is a λ-directed simple residual path, call redistribute(p,p_1).

Return.

5. Conclusions

The SPMF algorithm was implemented in C++. Its performance is significantly better than the improved algorithms for bipartite network maximum flow by Ahuja at el (implemented in C++ by the third author of this report). The amount of time $SPMF^{simple}$ takes to find all the minimum cuts was shown to be less than the average amount of time Ahuja's algorithm finds a single minimum cut. However, without access to an implementation of the parametric maximum flow algorithm by Gallo at el, experimental comparison with their algorithm is still missing.

Another advantage of SPMF is its simplicity which made its implementation very easy as shown in Section 4.

A generalized version of SPMF algorithm has been documented in a HP Technical report HPL-2004-189.

References

R. Ahuja, J. Orlin, C. Stein and R. Tarjan 1994. Improved algorithms for bipartite network flow. *SIAM Journal on Computing,* **23**, pp. 903-933

Balinksi, M. L. 1970. On a selection problem. *Management Science,* 17:3, 230-231.

Ford & Fulkerson 1956. Maximum Flow Through a Network, *Canadian Journal of Mathematics* 8, 339-404.

Ford, L.R. & Fulkerson, D.R., Flows in Networks, Princeton University Press, Princeton, NJ, 1962.

G. Gallo, M. D. Grigoriadis and R. E. Tarjan 1989. A fast parametric maximum flow algorithm and applications, SIAM Journal on Computing, **18**, pp. 30-55.

Goldberg, A.V. & Tarjan, R.E., 1986 and 1988. A New Approach to the Maximum Flow Problem, Proc. 18th Annual ACM Symposium on Theory of Computing, (1986), pp. 136-146; J. Assoc. Comput. Mach., 35 (1988).

Rhys, J. M. W. 1970. A selection problem of shared fixed costs and network flows. *Management Science,* 17:3, 200-207.

Zhang, B., Ward, J. and Feng, Q., A Simultaneous Maximum Flow Algorithm, Hewlett-Packard Research Laboratories Technical Report – HPL-2004-189.