Università di Pisa

Dipartimento di Informatica

Technical Report : TR-97-17

A strongly polynomial algorithm for the Balanced network flow problem

Klinz Bettina, Maria Grazia Scutella'

October 01, 1999

ADDR: Corso Italia 40,56125 Pisa,Italy. TEL: +39-50-887111. FAX: +39-50-887226

A Strongly Polynomial Algorithm for the Balanced Network Flow Problem

Bettina Klinz
TU Graz, Austria

Maria Grazia Scutellà
University of Pisa, Italy

Abstract
We show that the Balanced Network Flow Problem in the case of general weights, i.e., the problem of finding a feasible flow on a given network which minimizes the difference between the maximum and the minimum weighted flow on single arcs, can be solved in strongly polynomial time. The proposed solution algorithm consists of extending Megiddo's approach for parametric programming.

Keywords:
Flows, parametric programming

September, 1999

1. Introduction

Balanced Optimization Problems model "equitable" distribution of resources, in the sense that, given weights for the variables of the problem, the difference between the maximum and the minimum weighted variable defining a feasible solution has to be minimized. Combinatorial problems, like the Balanced Assignment Problem and the Balanced Spanning Tree Problem, have been studied by Martello et al. [3] and by Camerini et al. [2]. Furthermore, a parametric simplex method has been proposed by Ahuja [1] for the general Balanced Linear Programming Problem, and a specialized version has been provided for the Balanced Network Flow Problem. This problem consists in finding a feasible flow on a given network such that the difference between the maximum and the minimum weighted flow on single arcs be minimized. Special "balanced flows" have been addressed in the literature; they are the so-called "optimal sharing flows", where weights are only associated with the arcs outgoing from a given source node [10, 11].

Recently, the Balanced Network Flow Problem in the uniform case, i.e. in the case of unit weights, named the Uniform Balanced Network Flow Problem, has been addressed [8]. This problem is a linear programming problem with (0,+1,-1)-constraint matrix, and thus it can be solved in strongly polynomial time, for instance by Tardos algorithm [7]. However, Tardos algorithm is principally theoretical and oriented to general linear programming problems, whereas the algorithm described in [8] is "ad hoc" for the problem, being based on max-flow computations. In particular, it has been shown that the uniform problem can be solved in a strongly polynomial number of iterations by means of the classical Newton's method for fractional optimization, by $O(n^2 \log^3 n)$ max-flow computations. On the other hand, no result about the existence of a strongly polynomial algorithm is known at the moment for the general case.

In this paper, we shall show that the Balanced Network Flow Problem in the case of general weights, too, can be solved in strongly polynomial time, by extending Megiddo's approach for parametric programming.

The plan of the paper is the following. In Section 2, the Balanced Network Flow Problem is presented. Then, a strongly polynomial algorithm is described in Section 3.

2. The Balanced Network Flow Problem

Let us given a graph $G=(V,E)$, where V is the set of nodes, of cardinality n, and E is the set of arcs, of cardinality m. Let $b(i)$ be the flow demand associated with node i, $\forall i \in V$, while $c(i,j)$ be a nonnegative weight for unit of flow associated with arc (i,j), $\forall (i,j) \in E$.

The Balanced Network Flow Problem (BNFP) consists in finding a feasible flow f on G such that the difference between the maximum and the minimum weighted flow along single arcs of G, i.e. $\max_{(i,j)\in E} \{c(i,j)f(i,j)\} - \min_{(i,j)\in E} \{c(i,j)f(i,j)\}$, be minimized. Such an optimum flow will be also referred to as a *balanced flow*. W.l.o.g., we shall assume $c(i,j) > 0$, $\forall (i,j) \in E$.

Special balanced flows are the *optimal sharing flows*. There, a weight is only associated with the arcs outgoing from a given source node s (hereafter referred to as s-arcs), and, among the maximum flows from s to a given sink node t, one has to be computed which minimizes the difference between

the maximum and the minimum weighted flow along the *s*-arcs. As known from the literature [10, 11], the optimal sharing flow, which is indeed a balanced flow restricted to the *s*-arcs, is simultaneously a *maximin flow*, i.e. it maximizes the minimum weighted arc flow, and a *minimax flow*, i.e. it minimizes the maximum weighted arc flow. Such an attractive property does not hold generally for BNFP: the example in figure 1 shows an instance of BNFP where no balanced flow can be a maximin and a minimax flow simultaneously. In the figure, 1 is the only source node, having a flow demand equal to 2, while 3 is the only sink node; moreover, it is $c(1,2)=1/4$, whereas all the other arc weights are equal to 1. It is easy to see that a minimax flow is the flow holding 1 on the arcs (1,2), (1,3) and (2,3), and 0 on (2,1): the minimum value of the maximum weighted arc flow is thus 1. On the other hand, a maximin flow is the one having 1 on the arcs (1,3) and (2,3), 4 on the arc (1,2) and 3 on (2,1): the maximum value that the minimum weighted arc flow can assume is 1 as well.

Observe that any maximin flow must be characterized by a flow 1 both on (1,3) and (2,3). Then, in order to guarantee that such a flow be a minimax flow as well, a unit of flow needs to be pushed along (2,1). By the flow conservation constraints, a flow 2 has thus to be imposed on the arc (1,2), with the consequence that the resulting flow is not a maximin flow (in fact, the minimum weighted arc flow is 1/2).

In the considered example, a balanced flow is the one pushing 1 unit of flow along the arcs (2,1), (1,3) and (2,3), and 2 units of flow along (1,2). Observe that such a flow is a minimax flow, whereas it is not a maximin flow.

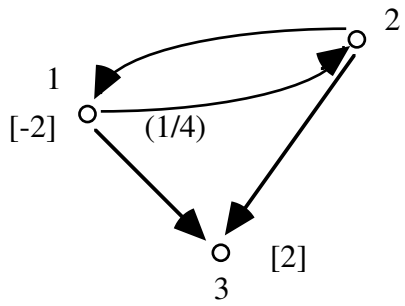

Figure 1

2.1. ***Problem formulations***

Let us introduce an upper capacity parameter z and a lower capacity parameter y; BNFP can then be mathematically formulated as follows:

$$\text{Min } z - y \tag{1}$$

$$\begin{aligned}
&\sum_{(j,i)\in BS(i)} f(j,i) - \sum_{(i,j)\in FS(i)} f(i,j) = b(i), && \forall i\in V,\\
&f(i,j) \le u(i,j)z, && \forall (i,j)\in E,\\
&f(i,j) \ge u(i,j)y, && \forall (i,j)\in E,\\
&f(i,j) \ge 0, && \forall (i,j)\in E,
\end{aligned}$$

where $u(i,j) = \dfrac{1}{c(i,j)}$ is the (positive) *capacity multiplier* associated with (i,j), $\forall (i,j)\in E$, and $BS(i)$ and $FS(i)$ denote the backward star and the forward star of i, respectively, $\forall i\in V$. By this rewriting, the problem consists in computing capacity parameters z and y such that a feasible flow f can be found on G when an upper capacity $u(i,j)z$ and a lower capacity $u(i,j)y$ are imposed on each arc (i,j), and such that the difference between the parameters, i.e. $(z - y)$, be minimized.

Let us set now $g(i,j) = f(i,j) - u(i,j)y$, $\forall (i,j)\in E$, and define $\lambda = z - y$; then the following, equivalent, linear programming formulation can be stated for BNFP:

$$\text{Min } \lambda \tag{2}$$

$$\begin{aligned}
&\sum_{(j,i)\in BS(i)} g(j,i) - \sum_{(i,j)\in FS(i)} g(i,j) = b(i) - k(i)y, && \forall i\in V,\\
&0 \le g(i,j) \le u(i,j)\lambda && \forall (i,j)\in E,\\
&y \ge 0,
\end{aligned}$$

where $k(i) = \sum_{(j,i)\in BS(i)} u(j,i) - \sum_{(i,j)\in FS(i)} u(i,j)$ is the difference between the sum of the capacity multipliers of the arcs in the backward star of i and the sum of the capacity multipliers of the arcs in the forward star of i, $\forall i\in V$.

If we fix parameter y in formulation (2), then BNFP reduces to a *parametric flow* (or, equivalently, a minimax flow) *problem* on a graph, say G^y, which differs from the input graph G since a flow demand $b(i) - k(i)y$ is associated with each node i of G. In fact, what we have to determine on G^y is the minimum value of the upper capacity parameter λ such that a flow g satisfying all the flow demands exists. Being in the case of general weights, each parametric flow problem so obtained can be solved by performing $O(m)$ max-flow computations on G^y by Newton's approach [6].

Let $h(y)$ denote the optimum objective value in (2) depending on parameter y. Function $h(y)$ is convex and piecewise linear. That provides an alternative formulation of problem BNFP: minimize function $h(y)$ on the half-space $\{y: y \ge 0\}$. What we have to do is to choose an efficient strategy for

minimizing $h(y)$, i.e. for finding the minimum value of λ for $y \geq 0$. This will be the subject of the following section.

3. A strongly polynomial algorithm for the Balanced Network Flow Problem

As observed before, if we fix parameter y in formulation (2), then the evaluation of function $h(y)$, i.e. the solution of the parametric flow problem on the graph G^y, can be approached via Newton's method in $O(m)$ max-flow computations.

In the following, firstly Newton's approach when applied to the evaluation of function $h(y)$ will be reviewed. Then, it will be shown how Megiddo's approach for fractional programming [4] can be extended for minimizing function $h(y)$, i.e. for solving BNFP.

3.1. *The evaluation of function h(y)*

Throughout this section, the assumption will be made that parameter y has been fixed, and function $h(y)$ needs to be evaluated. In all the notations below, y has thus to be interpreted as a constant value.

In order to solve the parametric flow problem on G^y, Newton's approach exploits the dual version of problem, the so called *maximum mean weight cut problem*. In our context, this problem consists in finding a cut (S,T) of G^y having the maximum mean weight surplus. The *surplus* of (S,T) in G^y is the difference between the total flow requirements of the nodes in T and the capacity of the cut without considering the parametric arc capacities $u(i,j)\lambda$:, $\forall (i,j) \in E$ (that cut capacity holds zero in the case of the graph G^y ; in more general parametric flow applications, the input graph may have capacitis of the form $q(i,j) + u(i,j)\lambda$:, $\forall (i,j) \in E$):

$$surplus_y(S,T) = \sum_{i \in T} (b(i) - k(i)y);$$

on the other hand, the *mean weight surplus* of (S,T) in G^y is defined as:

$$mean_y(S,T) = \frac{surplus_y(S,T)}{\sum_{i \in S} \sum_{j \in T} u(i,j)},$$

i.e. it is the surplus of (S,T) divided by the sum of the capacity multipliers of the directed arcs crossing the cut.

It can be proved, in fact, that the two problems have the same optimum values, i.e. the minimum value of parameter λ such that a feasible flow g can be found on G^y is equal to the maximum mean weight surplus of a cut of G^y. Then, $h(y)$ can be equivalently stated as:

$$h(y) = \max\{mean_y(S,T) : (S,T) \text{ cut in } G^y\}.$$

The set of all cuts of G^y can be described by means of a set of binary structures $X_{ST} \subseteq \{0,1\}^{n+m}$ as follows: each binary vector $x \in X_{ST}$ represents a cut (S,T) such that $x(i)=1$ if and only if $i \in T$, $i=1,\ldots, n$,

and $x(n+w)=1$ if and only if the w^{th} arc of G^y belongs to (S,T),$w=1,...,m$. Then, if b and k denote the $(n+m)$-vectors with components $b(i)$ and $k(i)$, respectively, $i=1,...,n$, and 0 otherwise, and if u denotes the $(n+m)$-vector with components 0, $i=1,...,n$, and 1 otherwise (i.e. in correspondance with the arcs of the graph), $h(y)$ can be also expressed as:

$$h(y) = \max\{ \frac{bx}{ux} - \frac{kx}{ux} y : x \in X_{ST}\}.$$

Now, let us introduce the λ-function:

$$H^y(\lambda) = \max\{ bx - ykx - \lambda ux : x \in X_{ST}\}.$$

As shown in [6], solving the parametric flow problem on G^y, or equivalently finding the maximum mean weight surplus of a cut of G^y, is equivalent to compute the root of the function $H^y(\lambda)$. That is, the evaluation of function $h(y)$ once the argument y has been fixed can be performed by calculating the root of function $H^y(\lambda)$.

In the following, Newton's approach when specialized to such a root computation will be described.

Newton's approach for computing h(y)

Let λ' denote the current approximation of parameter λ as computed by Newton's approach when applied to the finding of the root of function $H^y(\lambda)$ (initially, $\lambda'=0$). A each algorithm iteration, λ' is updated in an incremental way as follows until λ' is the optimum value, i.e. $\lambda' = h(y)$.

Typical Newton's iteration

Consider the graph, say $G^y(\lambda')$, obtained from the graph G^y by fixing the arc capacities to the values $\lambda' u(i,j)$, $\forall (i,j) \in E$ (that is, the parametric arc capacities grow up during Newton's iterations, until reaching the required flow feasibility).

Step 1- Compute a *maximum* surplus *cut* in $G^y(\lambda')$, where the *surplus* of a cut (S,T) in $G^y(\lambda')$ is the total flow demand in T less the cut capacity of (S,T) in G^y when λ has been set to value λ':

$$surplus_{y,\ \lambda'}(S,T) = \sum_{i \in T} (b(i) - k(i)y) \ - \lambda' \sum_{i \in S\ j \in T} u(i,j) \ .$$

Such a computation, which corresponds to the evaluation of function $H^y(\lambda)$ for $\lambda = \lambda'$, can be performed as follows.

Let us enlarge $G^y(\lambda')$ by introducing a super-source s (connected to each source i in $G^y(\lambda')$ by an arc (s,i) having capacity $-(b(i) - k(i)y)$), and a super-destination t (connected to each sink i in $G^y(\lambda')$ by an arc (i,t) having capacity $(b(i) - k(i)y)$).

A maximum surplus cut can then be found by computing a maximum flow $f_{\lambda'}$ from s to t in the extended network ($f_{\lambda'}$ can be computed starting from the maximum flow, if any, computed at the previous iteration of Newton's approach).

If flow $f_{\lambda'}$ is feasible for $G^y(\lambda')$, i.e. it satisfies all the flow demands of the nodes of $G^y(\lambda')$, then the maximum surplus of the cuts in $G^y(\lambda')$ holds 0: being $H^y(\lambda')=0$, the root of function $H^y(\lambda)$ has been found, and the algorithm stops (in particular, it is $\lambda'=h(y)$).

Otherwise, i.e. $f_{\lambda'}$ is not feasible, a maximum surplus cut is the cut $(S_{\lambda'},T_{\lambda'})$ having in $S_{\lambda'}$ all the sources with a negative residual demand with respect to $f_{\lambda'}$, and in $T_{\lambda'}$ all the sinks with a positive residual demand with respect to $f_{\lambda'}$ (observe that each directed arc crossing this cut has a zero residual capacity). Go to Step 2.

Step 2- Compute the mean weight surplus of $(S_{\lambda'},T_{\lambda'})$ in $G^y(\lambda')$, i.e. the surplus of $(S_{\lambda'},T_{\lambda'})$ in $G^y(\lambda')$ divided by the sum of the capacity multipliers of the directed arcs crossing the cut. As proved in [6], the surplus of $(S_{\lambda'},T_{\lambda'})$ can be equivalently computed in the residual graph relative to $f_{\lambda'}$, and so it is given by the sum of the (positive) residual demands of the nodes in $T_{\lambda'}$. Let ε denote such a mean weight surplus.

Step 3- Update the current approximation λ' according to $\lambda' := \lambda' + \varepsilon$, and go to the next iteration.

Informally speaking, at each iteration additional capacity is provided to the arcs of G^y via parameter λ, in a such a way that $surplus_y(S_{\lambda'},T_{\lambda'})=0$ after the parameter updating.

3.2. *The minimization of function h(y)*

Let y^* denote the argmin of function $h(y)$. In order to solve BNFP we should compute $h(y^*)$, i.e. firstly find out y^*, and then solve the parametric flow problem for $y=y^*$, by the Newton's approach previously reviewed; unfortunately, we do not know y^*. The idea is then to perform Newton's iterations independently of parameter y, when this is possible, i.e. by extending Newton's approach for the parametric flow problem to the problem with the parametric flow demands $b(i) - k(i)y$ at each node i of G. From this purpose, we need to extend the operations involving flows and capacities from the set of the real numbers to the set of the affine-linear functions (with respect to y).

As shown by Megiddo in [4], in order such a kind of extension works, the only operations involving affine-linear functions must be additions and subtractions, multiplications with a real number, and pairwise comparisons, as happens in the BNFP case. However, with respect to the applications of Megiddo's approach appeared in the literature so far, in the case of BNFP the tricky point is that there are two nested levels of parameters, i.e. y and λ, which must be carefully managed in order to get the approach applicable, as shown below.

A strongly polynomial algorithm

As the initialization phase, let us partition the feasible set $y \geq 0$ into a minimum set of intervals $\{I\}$ such that, for all the values y belonging to any interval I of the partition, the sign of $b(i) - k(i)y$ does not change for any $i \in V$ (in other words, i can be interpreted as a source or as a sink in any max-flow computation independently of the value of parameter y in I). Clearly, the cardinality of the above partition is at most n, since the change of the sign of $b(i) - k(i)y$ (and so, the determination of a different interval) may happen at most once for each node i of G.

Then, by performing $O(n)$ parametric flow computations, one for each end of the so found intervals, let us determine the slope of function $h(y)$ in correspondence with these ends, and thus the interval, say I^*, containing the argmin value y^* (assume, for the sake of simplicity, that I^* is unique).

Consider now the Newton's iteration as described in 3.1., and try to perform it independently of $y \in I^*$.

The parametric iteration

As before, let λ' denote the current approximation of parameter λ as computed by Newton's approach.

In Step 1, we have to compute a *maximum* surplus *cut* in $G^y(\lambda')$ along the lines described in 3.1.

At this regard, observe that it is possible to generalize any classical max-flow algorithm to work also in the presence of parameter y. Consider for instance a max-flow algorithm which pushes flow along shortest augmenting paths. At each augmentation, we have to perform $O(n)$ comparisons among the residual capacities of the arcs composing the selected path in order to determine the amount of flow to be pushed. Since the flow demands are affine-linear functions of y, at each augmentation the residual capacities are affine-linear functions of y too. We have thus to perform pairwise comparisons between affine-linear functions of y to compute the amount of flow to be pushed.

In order to decide the result of the comparison between two functions of this type within the considered interval I^*, we have first to find their intersection point, and then evaluate function $h(y)$ at this intersection point (by means of one parametric flow computation): depending on the slope of $h(y)$ at the intersection point, we can properly shrink the current interval I^* (I^* reduces to the subinterval at the left or at the rigth of the intersection point), and then take the decision about the comparison. We thus need $O(n)$ parametric flow computations per augmentation, and $O(n^2m)$ parametric flow computations for calculating a maximum flow if, for example, we use a simple max-flow algorithm like Dinic's one [9].

Once a maximum flow $f_{\lambda'}$ has been found in Step 1, one can easily compute a maximum surplus cut $(S_{\lambda'}, T_{\lambda'})$ and its mean weight surplus, as required in Step 2. Observe that, being the flow along each arc a linear function of y, the residual flow demands of the nodes in $T_{\lambda'}$ are generally linear functions of y too and, as a consequence, the computed mean weight surplus is generally a linear function of y. This linear function provides the increment ε needed to compute the next approximation of parameter λ in Step 3 (which is, generally, a linear function of y, too): $\lambda' := \lambda' + \varepsilon$.

At termination, the final set I^* gives the set of the optimum values of parameter y.

The algorithm time complexity

Let $T_{par\text{-}flow}$ *denote the time complexity for finding a parametric flow*. The following result holds true:

Theorem 1 - (Computational time complexity) *If Dinic's algorithm is used to "simulate" the maximum flow computations*, then BNFP *can be solved in* $O(n^2m^2T_{par\text{-}flow})$ *time*.

Proof: Each algorithm iteration requires $O(n^2m)$ parametric flow computations. Due to the fact that Newton's approach for the parametric flow problem has been "simulated", i.e. performed to solve BNFP independently of parameter y, the same number of iterations as the ones of Newton's approach when applied to the parametric flow problem is performed, i.e. $O(m)$. The resulting time complexity is thus $O(n^2m^2T_{par\text{-}flow})$.

Corollary - (Computational time complexity) *If Newton's approach is used for the parametric flow computations*, BNFP *can be solved by* $O(n^2m^3)$ *max-flow computations*.

Conclusions and future work

The time complexity of Newton's approach when applied to the parametric flow problem does not depend on the particular max-flow algorithm used as its subroutine. The time complexity of the proposed algorithm for BNFP can thus be made parametric with respect to the one of the selected max-flow algorithm.

Let $T_{maxflow}$ denote the time complexity of the used max-flow algorithm. At each iteration of the "simulated" parametric flow algorithm, a max-flow computation on a graph depending on parameter y is performed. This computation requires $O(T_{maxflow})$ comparisons between affine-linear functions of y, and so $O(T_{maxflow})$ parametric flow computations. Since each parametric flow computation requires $O(m)$ max-flow computations, and $O(m)$ algorithm iterations are performed, the following parametric time complexity can be derived:

Theorem 2 - BNFP *can be solved in* $O(m^2T^2_{maxflow})$ *time*.

In particular observe that, if Dinic's algorithm is chosen, as in the section before, being $T_{maxflow}=O(n^2m)$, the resulting time complexity is $O(n^2m^3T_{maxflow})$, according to the previous Corollary.

A further refinement could be the following one. In Radzik's paper, in the case of capacities functions of type $d(i,j)+u(i,j)\lambda$, a parametric flow can be computed by means of $O(m)$ max-flow computations, by using Newton's approach. In the balanced flow case, we have parametric capacities of type $u(i,j)\lambda$

(i.e. $d(i,j)=0$ for each original arc (i,j)). In this case, $O(n)$ max-flow computations could be sufficient. Then, the overall time complexity for BNFP should be $O(n^2T^2_{maxflow})$.

References

[1] R.K. Ahuja, The balanced linear programming problem, European Journal of Operational Research 101 (1997), 29-38.

[2] P.M. Camerini, F. Maffioli, S. Martello and P. Toth, Most and Least Uniform Spanning Trees, Discrete Applied Mathematics 15 (1986) 181-197.

[3] S. Martello, W.R. Pulleyblank, P. Toth and D. de Werra, Balanced Optimization Problems, Operations Research Letters 5, vol. 3 (1984) 275-278.

[4] N. Megiddo, Combinatorial optimization with rational objective functions, Mathematics of Operations Research 4 (1979), 414-424.

[5] A.J. Hoffman, A generalization of max-flow min-cut, Mathematical Programming 6 (1974) 352-359.

[6] T. Radzik, Parametric Flows, Weighted Means of Cuts and Fractional Combinatorial Optimization, in: Pardalos, ed., Numerical Optimization (World Scientific, 1993) 351-386.

[7] É. Tardos, A strongly polynomial algorithm for solving combinatorial linear programs, Operations Research 34 (1986) 250-256.

[8] M.G. Scutellà, A strongly polynomial algorithm for the Uniform Balanced Network Flow Problem, Discrete Applied Mathematics 81 (1998) 123-131.

[9] R.E. Tarjan, Data Structures and Network Algorithms, SIAM, Philadelphia (1983).

[10] U. Zimmermann, Linear and combinatorial sharing problems, Discrete Applied Mathematics 15 (1986) 85-104.

[11] U. Zimmermann, Sharing problems, Optimization 17 (1986) 31-47.