Discrete Applied Mathematics 10 (1985) 117–124
North-Holland

MINIMUM COST FLOW ALGORITHMS FOR SERIES-PARALLEL NETWORKS

Wolfgang W. BEIN, Peter BRUCKER
Fachbereich Mathematik/Informatik, Universität Osnabrück, Postfach 4469, 4500 Osnabrück, West Germany

Arie TAMIR
Department of Statistics, Tel-Aviv University, Ramat-Aviv, Tel-Aviv 69978, Israel

Received 15 January 1984

It is shown that an acyclic multigraph with a single source and a single sink is series-parallel if and only if for arbitrary linear cost functions and arbitrary capacities the corresponding minimum cost flow problem can be solved by a greedy algorithm. Furthermore, for networks of this type with m edges and n vertices, two $O(mn + m \log m)$-algorithms are presented. One of them is based on the greedy scheme.

1. Introduction

A directed *(multi) graph G* is given by a finite set E of *edges*, a *finite* set V of vertices and two mappings $h, t : E \to V$ which associate with each edge $e \in E$ the *head* $h(e)$ and the *tail* $t(e)$ of e. $h(e)$ is called a *successor* of $t(e)$ and $t(e)$ is called a *predecessor* of $h(e)$. A vertex without predecessors is called a *source*; a vertex without successors is called a *sink*. Two edges e and e' are called parallel if $h(e) = h(e')$, and $t(e) = t(e')$. For each vertex $v \in V$ we denote the set of edges e with $t(e) = v$ by $\mathrm{OUT}(v)$ and with $h(e) = v$ by $\mathrm{IN}(v)$. $\mathrm{OUT}(v)$ is the set of *outgoing edges* with respect to v, and $\mathrm{IN}(v)$ is the set of *ingoing edges* with respect to v. A (*two terminal*) *series-parallel graph* is a multigraph with exactly one source and one sink, which is defined recursively as follows:

(i) A single edge e together with $t(e)$ and $h(e)$ is a series-parallel graph.

(ii) If S_1 and S_2 are series-parallel graphs, so is the multigraph obtained by either of the following operations:

(a) *Parallel composition:* identify the source of S_1 with the source of S_2 and the sink of S_1 with the sink of S_2.

(b) *Series composition:* identify the sink of S_1 with the source of S_2.

Consider for a series-parallel graph $S = (E, V, h, t)$ the following parametric network flow problem $P(q)$ in which q is some nonnegative real parameter, a_e $(e \in E)$ are arbitrary real numbers and c_e are nonnegative integers for $e \in E$. Furthermore, the source and the sink of S are denoted by s and t respectively.

$P(q)$ Minimize $\sum_{e\in E} a_e x_e$, (1)

subject to

$$\sum_{e\in \mathrm{IN}(v)} x_e = \sum_{e\in \mathrm{OUT}(v)} x_e, \quad v\in V\setminus\{s,t\}, \tag{2}$$

$$\sum_{e\in \mathrm{OUT}(s)} x_e = \sum_{e\in \mathrm{IN}(t)} x_e = q, \tag{3}$$

$$0\le x_e\le c_e, \quad e\in E. \tag{4}$$

A vector $x=(x_e)$ is called a *feasible solution* for $P(q)$ if x satisfies the restrictions (2)–(4). The maximal integer value q for which $P(q)$ has a feasible solution is called *maximal flow value,* and denoted by $q_{\max}$.

In Section 2 we will show that an acyclic multigraph with a single source and a single sink is series-parallel if and only if for arbitrary linear cost functions $\{a_e\}$, $e\in E$ and arbitrary capacities, $\{c_e\}$, $e\in E$, the corresponding minimal cost flow problem $P(q)$, for $0\le q\le q_{\max}$, is solvable by a greedy algorithm.

Thus the greedy scheme is valid for series-parallel networks. Let $|V|=n$ and $|E|=m$. An implementation of this greedy scheme in an overall time of $O(mn+m\log m)$ as well as a second algorithm with the same complexity are presented in Section 3.

The following special case of the above problem has been dealt with by Brucker [2].

A multigraph G without parallel edges is called a *tree* if G has exactly one sink t and each vertex $v\ne t$ has exactly one successor. A tree may be transformed into a series-parallel graph by adding one source s and edges e with $t(e)=s$ and $h(e)=v$ for all *leaves* (i.e. vertices without predecessors) v of the tree. We also call series-parallel graphs constructed in such a way *trees.*

Brucker [2] has shown that if G is a tree and q is a fixed integer, problem $P(q)$ can be solved in $O(m\log m)$ steps. Special tree problems with convex cost functions have been discussed by Brucker [1] and Tamir [3], [4].

2. Minimum cost flows in series-parallel graphs

The construction process of series-parallel graphs along their recursive definition may be represented by binary trees which are called *decomposition trees.* In a decomposition tree sets of parallel edges of the graph are represented by the leaves of the tree. Vertices of the decomposition tree which are not leaves are labelled by S indicating a series composition, or P indicating parallel composition. In Fig. 1 an example of a series-parallel graph together with its composition tree is shown. Note that the sons of a vertex labeled with S are ordered.

Valdes, Tarjan and Lawler [5] gave an algorithm to check whether a given multigraph is series-parallel and to construct its decomposition tree in that case. The complexity of this algorithm is $O(|E|)$.

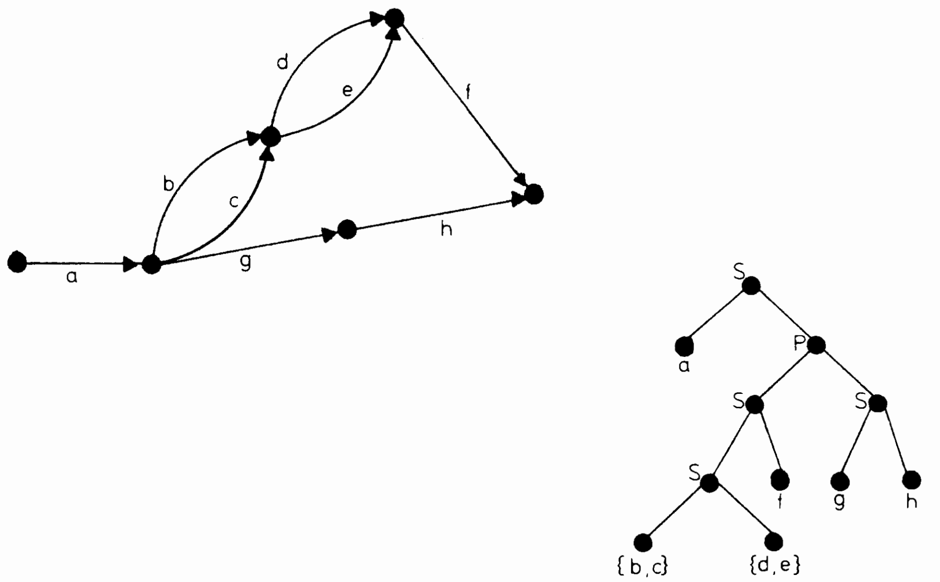

Fig. 1.

To solve problem $P(q)$ we assume that the series parallel graph S is given by a decomposition tree T with vertices $1, 2, \ldots, r$. Furthermore, let the vertices in T be *enumerated topologically,* i.e. we have $i<j$ if j is a father of i. r is the root of the tree. The subtree rooted in i is denoted by T_i. T_i corresponds to a series-parallel submultigraph S_i of S.

The solution of the parametric problem $P(q)$ may be described by the maximal flow value $q_{\max}$ and the optimal value $f(q)$ of the objective function of $P(q)$ for each q, $0 \le q \le q_{\max}$.

f is a piecewise linear convex function defined on the interval $[0, q_{\max}]$ with $f(0)=0$. A complete description of f is given by a partition of $[0, q_{\max}]$ into consecutive subintervals I_j $(j=1, \ldots, t)$ of length l_j, where the slope u_j of f does not change (see Fig. 2). Note that the sequence $u_1, u_2, \ldots, u_t$ is nondecreasing.

Furthermore, to each interval I_j, there corresponds a path p_j from s to t which has the property that the cost of one unit of flow along this path is equal to u_j. Thus, a complete solution of $P(q)$ may be characterized by

$$q_{\max} \quad \text{and} \quad (l_j, u_j, p_j) \quad \text{for } j=1, \ldots, t. \tag{6}$$

We also call (6) a *solution* of $P(q)$. Such a solution may be constructed using the following greedy algorithm:

Greedy Algorithm

1. **For all** $e \in E$ **do** $x_e := 0$; $j := 0$;
2. **While** there exists a path connecting s and t **do**
   Begin
3. $j := j+1$
4. Find a minimal cost path p_j and the corresponding u_j-value;
5. $l_j := \min\{c_e \mid e \in p_j\}$;

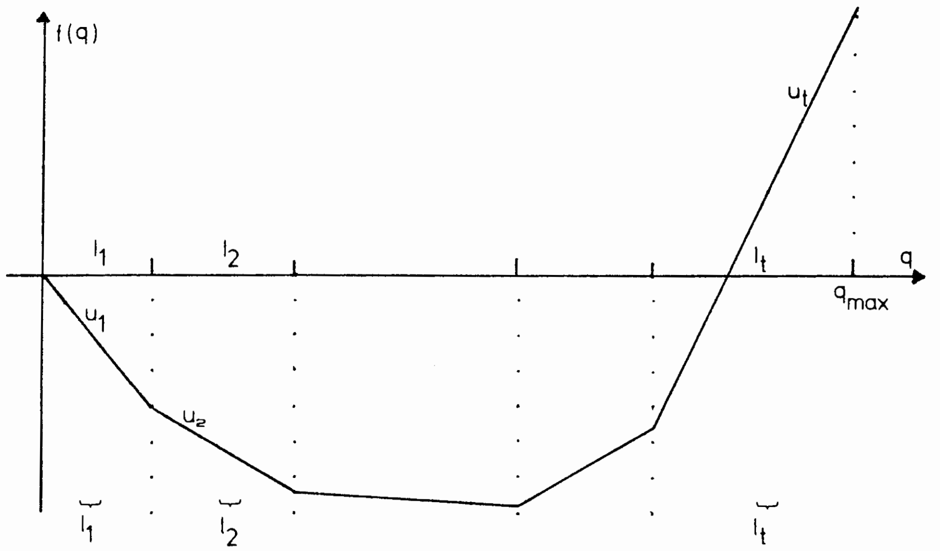

Fig. 2.

6. **For all** $e \in p_j$ **do**
   Begin
7. $c_e := c_e - l_j$; $x_e: x_e + l_j$
8. **If** $c_e = 0$ **then** $E := E \setminus \{e\}$
   End
End

Notice that the Greedy Algorithm is an augmenting path algorithm which does not use backward arcs.

Next, we will show that this algorithm solves $P(q)$ for all $0 \leq q \leq q_{max}$ and for arbitrary data if and only if the network is series-parallel.

Theorem. *Let G be a directed acyclic multigraph with a single source s and a single sink t. G is a (two terminal) series-parallel graph if and only if for every set of costs* $\{a_e\}$, $e \in E$, *and every set of nonnegative capacities* $\{c_e\}$, $e \in E$, *the above Greedy Algorithm solves the corresponding minimal cost flow problem* $P(q)$, *for* $0 \leq q \leq q_{max}$.

The proof of the theorem will use the following notation and definitions.

A directed path in G from vertex x to vertex y will be denoted by $P(x, y)$. x and y will then be called its *end vertices*.

We will say that two paths $P(x, y)$ and $P(u, v)$ are *vertex disjoint* if the fact that a vertex w is in both paths implies that w is an end vertex of $P(x, y)$ and $P(u, v)$.

Proof. Suppose first that G is series-parallel. The following result justifies the validity of the Greedy Algorithm:

Let q, $0 \leq q \leq q_{max}$, and let $P(s, t)$ be a minimum cost path connecting the source s and the sink t. Then there exists x^*, an optimal solution to problem $P(q)$, with

$$x_e^* \geq \min\left(q, \min_{e \in P(s,t)} \{c_e\}\right) \quad \text{for each edge } e \in P(s,t).$$

We prove the result by induction on the number of edges in G. Assume that G is obtained by a series composition of the series parallel graphs G_1 and G_2, where s_i, t_i are the terminals of G_i, $i=1,2$ and $t_1 = s_2$. Let $P_i(s,t)$ denote the restriction of $P(s,t)$ to G_i, $i=1,2$. Consider the problem $P(q)$ defined on G_i, $i=1,2$. By the induction hypothesis there exists an optimal solution x^i, $i=1,2$, to this problem such that $x_e^i \geq \min(q, \min_{e \in P_i(s,t)} \{c_e\})$. Since (x^1, x^2) optimally solves $P(q)$ on G, the proof for the series composition is complete.

Suppose now that G is obtained by a parallel composition of G_1 and G_2. Without loss of generality assume that $P(s,t)$ is contained in G_1. Thus, if q_1 units are flowing through G_1 in an optimal solution to $P(q)$ on G, we may assume without loss of generality that $q_1 \geq \min(q, \min_{e \in P(s,t)} \{c_e\})$. By the induction hypothesis on G_1 there exists an optimal solution to $P(q)$ on G such that the flow on each edge $e \in P(s,t)$ is at least $\min(q, \min_{e \in P(s,t)} \{c_e\})$.

For the second part of the theorem, let G be a directed acyclic multigraph with a single source and a single sink. Assume that the Greedy Algorithm is valid for $P(q)$, $0 \leq q \leq q_{\max}$. Suppose that G is not series parallel. It then follows from [5] that there exist in G four distinct vertices s', t', u, v and five (pairwise) vertex disjoint directed paths, $P(s',u)$, $P(s',v)$, $P(u,v)$, $P(u,t')$ and $P(v,t')$. Furthermore if $s' \neq s$ (i.e., s' is not a source), the properties of G imply the existence of a path $P(s',s)$, such that $P(s',s)$ and the above five paths are (pairwise) vertex disjoint. Similarly if $t' \neq t$ there exist a path $P(t',t)$ such that the seven paths $P(s,s')$, $P(s',u)$, $P(s',v)$, $P(u,v)$, $P(u,t')$, $P(v,t')$ and $P(t',t)$ are pairwise vertex disjoint (see Fig. 3).

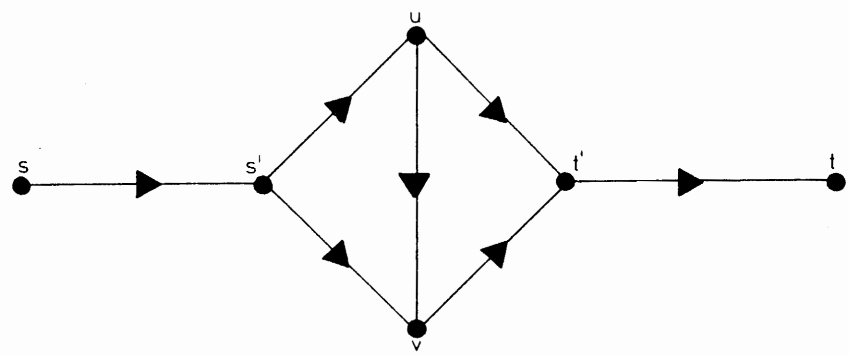

Fig. 3.

Next we define the capacities $\{c_e\}$ and the costs $\{a_e\}$.

$$c_e = \begin{cases} 2 & \text{if } e \text{ is on } P(s,s') \text{ or on } P(t',t), \\ 1 & \text{if } e \text{ is on } P(s',u) \text{ or on } P(s',v) \text{ or on} \\ & \quad P(u,v) \text{ or on } P(u,t') \text{ or on } P(v,t'), \\ 0 & \text{otherwise.} \end{cases}$$

$$a_e = \begin{cases} 0 & \text{if } e \text{ is on } P(s',u) \text{ or on } P(u,v) \text{ or on } P(v,t'), \\ 1 & \text{otherwise.} \end{cases}$$

If we set $q=2$, the optimum solution to $P(2)$ does not use the unique minimum cost path connecting s and t. Thus, the Greedy Algorithm does not solve $P(2)$ and the proof is complete.

In the next section we will show that for graphs with m edges and n vertices the Greedy Algorithm can be implemented in an overall time of $O(mn + m \log m)$.

3. Implementation. A bottom-up procedure

In this section we first discuss the implementation of step 4 of the Greedy Algorithm.

The minimal cost path p from s to t and the corresponding u-value can be calculated along the decomposition tree of the series-parallel network using the following algorithm.

Algorithm 1

```
1.  For i := 1 until r do
2.      If i is a leaf then
3.          INITIALIZE(i)
        else
            Begin
4.          Find the left son j and right son k of i;
5.          If i has label P then
                MERGE(j,k;i)
            else
6.              ADD(j,k;i)
            End
```

Notice that Algorithm 1 proceeds from the leaves of the decomposition tree to the root because the nodes of this tree are enumerated topologically. The procedure INITIALIZE(i) chooses among the set $E(i)$ of parallel edges e associated with leaf i one, say $\bar{e}$, with the smallest a_e-value and sets $p_i := \bar{e}, u_i := a_{\bar{e}}$. If $E(i)=\emptyset$, then it sets $p_i := \emptyset$, $u_i := \infty$.

The procedures MERGE$(j,k;i)$ and ADD$(j,k;i)$ are defined as follows.

```
MERGE(j,k;i)
  If u_j ≤ u_k then
      Begin u_i := u_j; p_i := p_j End
  else
      Begin u_i := u_k; p_i := p_k End
```

ADD$(j, k; i)$
$u_i := u_j + u_k$;
$p_i = p_j \circ p_k$

In the second procedure $p_j \circ p_k$ denotes the concatenation of p_j and p_k. $p_j \circ p_k = \emptyset$ if $p_j = \emptyset$ or $p_k = \emptyset$. Notice that if Algorithm 1 calculates $p_r = \emptyset$, then there exists no path connecting s and t.

For series-parallel graphs without parallel edges it can be shown by induction that the number of edges is at most $2n - 3$. Thus the decomposition tree has $O(n)$ vertices and, if we do not count the effort involved in step 3, the complexity of Algorithm 1 is $O(n)$.

Because of step 8 the number of iterations of the **while** loop of the Greedy Algorithm is $O(m)$. Thus if we do not count the calls of all INITIAL-procedures, the overall complexity of the Greedy Algorithm is $O(mn)$. For an efficient implementation of the INITIAL procedures we use heaps to represent the sets of parallel edges $E(i)$. Then, a minimal cost edge can be found in constant time. Furthermore, if in step 8 of the Greedy Algorithm an edge is eliminated, the corresponding heap can be updated in $O(\log m)$ steps. Thus the overall complexity of the Greedy Algorithm is $O(mn + m \log m)$.

We will now discuss an algorithm which solves $P(q)$ for all $0 \le q \le q_{\max}$ and has the same complexity as the Greedy Algorithm, but some computational advantages. Let i be a vertex of the decomposition tree and let S_i be the series parallel graph associated with T_i, the subtree rooted at i. Now let

$$q_{\max}^{(i)}, (l_j^{(i)}, u_j^{(i)}, p_j^{(i)}), \quad j = 1, \ldots, t^{(i)} \quad \text{with } u_1^{(i)} \le u_2^{(i)} \le \cdots \le u_{t^{(i)}}^{(i)} \tag{7}$$

be a solution of the corresponding subproblem $P^{(i)}(q)$.

The idea of the algorithm is to solve the problems $P^{(i)}(q)$ for $i = 1, \ldots, r$ recursively using Algorithm 1. All we have to do is to choose an appropriate data structure and replace the procedures INITIALIZE(i), MERGE$(j, k; i)$, and ADD$(j, k; i)$ by procedures INITIALIZE1(i), MERGE1$(j, k; i)$ and ADD1$(j, k; i)$ respectively. These new procedures may be described as follows:

(i) INITIALIZE1(i) creates a queue Q_i of data elements $(l_e^{(i)}, u_e^{(i)}, p_e^{(i)})$, $e \in E(i)$ with $l_e^{(i)} = c_e$; $u_e^{(i)} = a_e$, and $p_e^{(i)} = e$, sorted by $u_e^{(i)}$-values. Furthermore $q_{\max}^{(i)}$ is set equal to $\sum_{e \in E(i)} c_e$.

(ii) MERGE1$(j, k; i)$ merges the queues Q_j and Q_k into a new (sorted) queue Q_i and sets $q_{\max}^{(i)} = q_{\max}^{(j)} + q_{\max}^{(k)}$.

(iii) ADD1$(j, k; i)$ is more complicated. A detailed description is given below. In this description FIRST(Q), MAKENULL(Q), INSERT$((l, u, p); Q)$, and DELETE(Q) are the usual operations on the queue Q.

ADD1$(j, k; i)$
1. $q_{\max}^{(i)} := \min\{q_{\max}^{(j)}, q_{\max}^{(k)}\}$;

2. MAKENULL(Q_i);
3. $(l_j, u_j, p_j) := \text{FIRST}(Q_j)$; $(l_k, u_k, p_k) := \text{FIRST}(Q_k)$;
4. **While** $Q_j \neq \emptyset$ **and** $Q_k \neq \emptyset$ **do**
    Begin
5.   **If** $l_j < l_k$ **then**
        Begin
6.   INSERT($(l_j, u_j + u_k, p_j \circ p_k); Q_i$);
7. $l_k := l_k - l_j$;
8.   DELETE(Q_j);
9. $(l_j, u_j, p_j) := \text{FIRST}(Q_j)$
        End;
10.   **If** $l_k < l_j$ **then**
        Begin
11.   INSERT($(l_k, u_j + u_k, p_j \circ p_k); Q_i$);
12. $l_j := l_j - l_k$;
13.   DELETE(Q_k);
14. $(l_k, u_k, p_k) := \text{FIRST}(Q_k)$
        End;
15.   **If** $l_j = l_k$ **then**
        Begin
16.   INSERT($(l_j, u_j + u_k, p_j \circ p_k); Q_i$);
17.   DELETE(Q_j); DELETE(Q_k);
18. $(l_j, u_j, p_j) := \text{FIRST}(Q_j)$; $(l_k, u_k, p_k) := \text{FIRST}(Q_k)$
        End
    End

All sets of parallel edges can be sorted in an overall time of $O(m \log m)$. Furthermore, for each call of ADD1 and MERGE1 there are at most $O(m)$ steps. Thus, the second algorithm also has complexity $O(mn + m \log m)$.

Note that if we are interested only in the maximal flow values, these can be calculated in at most $O(m)$ steps doing only the $q^{(i)}_{\max}$ calculations.

References

[1] P. Brucker, Network flows in trees and knapsack problems with nested constraints, in: H.J. Schneider and H. Göttler, eds., Proc. 8th Conf. Graphtheoretic Concepts in Computer Science (Hanser, Müchen, 1982) 25–35.

[2] P. Brucker, An $O(n \log n)$-algorithm for the minimum cost flow problem in trees, in: G. Hammer and D. Pallaschke, eds., Topics in Operations Research and Mathematical Economics (Springer, Berlin, 1984) 299–306.

[3] A. Tamir, Efficient algorithm for a selection problem with nested constraints and its application to a production-sales planning model, SIAM J. Control Optimization 18 (1980) 282–287.

[4] A. Tamir, Further remarks on selection problems with nested constraints, Department of Statistics, Tel Aviv University (1979).

[5] J. Valdes, R.E. Tarjan and E.L. Lawler, The recognition of series-parallel diagraphs, SIAM J. Comput. 11 (1982) 298–313.