

An $O(n \log n)$ -Algorithm for Solving a Special Class of Linear Programs*

W. Bein, Albuquerque, P. Brucker, Osnabrück

Received June 7, 1988; revised February 13, 1989

Abstract — Zusammenfassung

An $O(n \log n)$ -Algorithm for Solving a Special Class of Linear Programs. An $O(nm + n \log n)$ -algorithm was developed by Kern [1986] to solve linear programs of the form $\max \{cx \mid l \leq Ax \leq b, L \leq x \leq U\}$ where l, b, L, U are nonnegative and A is a 0-1-matrix of dimension $m \times n$ with the property that the support of each row is contained in the support of every subsequent row. We will show that a more general class of linear programs can be solved in $O(n \log n)$ -time.

AMS Subject Classifications: 90C05, 90C35.

Key words: Linear programming, network flows, tree.

Ein $O(n \log n)$ -Algorithmus zur Lösung einer speziellen Klasse von linearen Programmen. Kern [1986] entwickelte einen $O(nm + n \log n)$ -Algorithmus zur Lösung von linearen Programmen der Form $\max \{cx \mid l \leq Ax \leq b, L \leq x \leq U\}$, wobei l, b, L, U nichtnegativ sind und A eine 0-1-Matrix der Dimension $m \times n$ mit der Eigenschaft, daß der Träger jeder Zeile von A im Träger jeder der nachfolgenden Zeilen enthalten ist, darstellt. Wir zeigen, daß eine allgemeinere Klasse von linearen Programmen sich mit einem Aufwand von $O(n \log n)$ lösen läßt.

1. Introduction

Let $A = (a_{ij})$ be a binary matrix with m rows and n columns. For each $i = 1, \dots, m$ denote by $\text{sup}(i)$ the support of row i , i.e.

$$\text{sup}(i) = \{j \mid a_{ij} = 1\}.$$

A has the “Manhattan Skyline” property (MSP) if $\text{sup}(i) \subseteq \text{sup}(j)$ for all i, j with $i \leq j$. Kern [1986] developed an $O(mn + n \log n)$ -algorithm for linear programs of the form

$$\max cx \quad \text{subject to} \quad \begin{cases} l \leq Ax \leq b \\ L \leq x \leq U \end{cases} \quad (1)$$

where A is a binary $n \times m$ -matrix with the MSP. A similar result can be found in Erenguc [1986].

* Supported in part by the Deutsche Forschungsgemeinschaft (Project CODiS) and by UCR grant 453-2810.

We will present an $O(n \log n)$ -algorithm for the larger class of problems (1) in which A satisfies the property

$$\sup(i) \cap \sup(j) \neq \emptyset \text{ implies } \sup(i) \subseteq \sup(j) \text{ or } \sup(j) \subseteq \sup(i). \tag{2}$$

We incorporate the n restrictions $L \leq x \leq U$ into the system $l \leq Ax \leq b$ and add a restriction

$$\sum_{i=1}^n l_i \leq \sum_{i=1}^n x_i \leq \sum_{i=1}^n b_i$$

if A does not contain a row $1 = (1, \dots, 1)$. We also assume that all rows in the extended matrix are different. Then we get a problem of the general form

$$\max cx \quad \text{subject to} \quad l \leq Ax \leq b \tag{3}$$

where A satisfies (2) and contains all unit vectors as rows as well as the row $1 = (1, \dots, 1)$. In addition to this all rows are different.

Problem (3) corresponds to a network flow problem in a tree. The rows of A correspond to the vertices of the tree. Vertex i is a son of vertex j if and only if $\sup(i) \subset \sup(j)$ and there exists no $k \neq i, j$ such that $\sup(i) \subset \sup(k) \subset \sup(j)$. Rows i with $|\sup(i)| = 1$ correspond with the leaves of the tree. Row 1 corresponds with the root. All arcs are directed towards the root (i.e. we have an intree). For each node i there is a lower capacity l_i and an upper capacity b_i for the flow passing i . We have to send flows x_i from the leaves i to the root of the tree such that the sum $\sum c_i x_i$ is maximized.

In Brucker [1984] it is shown that this network flow problem can be solved in $O(n \log n)$ steps if $l=0$. We will extend this result by developing an $O(n \log n)$ -algorithm for the general problem.

2. Flows in Treelike Networks with Lower Capacities

Let T be an intree with nodes $1, \dots, n$. Associated with the nodes there are lower and upper capacities l_i and b_i with $0 \leq l_i \leq b_i$. We assume that the nodes of T are enumerated topologically. Thus, n is the root of T . Furthermore $L(i)$ is used to denote the set of leaves of the subtree rooted at node i ($i = 1, \dots, n$). For all leaves $i \in L(n)$ we have "profit"-values c_i .

We consider the following flow problem with lower bounds

$$\max \sum_{i \in L(n)} c_i y_i \quad \text{subject to} \quad l_i \leq y_i := \sum_{j \in L(i)} y_j \leq b_i \quad i = 1, \dots, n. \tag{4}$$

To solve (4) we denote the set of all predecessors of node $i \notin L(n)$ by $P(i)$ and assume that

$$\sum_{j \in P(i)} l_j \leq l_i \quad \text{for all } i \notin L(n). \tag{5}$$

This can always be accomplished in linear time by going up the tree and increasing lower capacities of fathers if necessary.

An algorithm which solves (4) can be outlined as follows:

We first create a feasible solution y , such that there exists an optimal solution x^* with $y \leq x^*$. Then by replacing b by $b - y$ and l by 0 we transform the problem (4) into a problem (4') without lower bounds. A solution y^* of (4') yields a solution $y + y^*$ of (4). The simpler problem (4') can be solved in $O(n \log n)$ time as shown in Brucker [1984].

The idea for creating such a feasible solution y is to satisfy the lower bounds in an optimal way while going up the tree. More specifically before the i -th iteration we have $y_j = l_j$ for all $j \in P(i)$. During the i -th iteration we send additional flow from the leaves up to the root until $y_i = l_i$. We choose leaves with highest profit first. Furthermore in this increase we have to observe the upper bounds. If $y_i = l_i$ cannot be achieved the problem is infeasible.

To observe the upper bounds b we keep track of residual capacities $r_i = b_i - y_i$ ($i = 1, \dots, n$). Initially we have $y = 0$ and $r = b$. In the general step the flow that can be sent along the path from the chosen leaf to the root of the tree is bounded by the minimum r_k -value along that path. After this increase we update the r_k values.

The details of this procedure are given in the following algorithm.

Algorithm

1. $y := 0; r := b;$
2. FOR $j = 1$ TO n DO
 BEGIN
3. IF j is a leaf then $L(j) := \{j\}$ ELSE $L(j) := \bigcup_{i \in P(j)} L(i);$
4. WHILE $y_j < l_j$ AND $L(j) \neq \emptyset$ DO
 BEGIN
5. Find $k \in L(j)$ with c_k maximal;
6. $\Delta := \text{FINDMINVALUE}(k);$
7. IF $\Delta \leq l_j - y_j$ THEN DELETE $(k, L(j))$ ELSE $\Delta := l_j - y_j;$
8. ADDVALUE (k, Δ)
- END;
9. IF $y_j < l_j$ THEN STOP (*Problem is infeasible*)
- END

Here FINDMINVALUE and ADDVALUE are operations on the tree with node values r_i and y_i which are defined as follows:

FINDMINVALUE(v): Return the value of the node with minimum r -value on the path from v to the root.

ADDVALUE(v, Δ): Adds $-\Delta$ to the r -values and Δ to the y -values of every node on the path from v to the root.

The operation DELETE($k, L(j)$) deletes k from $L(j)$.

Each of the operations `FINDMINVALUE` and `ADDVALUE` are applied at most $O(n)$ times. This follows from the fact that in step 7, either a leaf k is deleted or $l_j - y_j$ is reduced to 0 in step 8. Using a data structure by Sleator and Tarjan [1987] each of the operations `FINDMINVALUE` and `ADDVALUE` can be done in $O(\log n)$ time yielding an overall $O(n \log n)$ time bound for steps 6. and 8. Note that in procedure `ADDVALUE` adding to the values of all nodes along a path does not require to traverse that path, since those values are kept as differences of values between childnode and parentnode. For details, see Sleator and Tarjan [1987].

For the same reasons the `FINDMAX`-operation of step 5. and the `DELETE`-operation in step 7. are applied at most $O(n)$ times. Furthermore the sets $L(j)$ are constructed using at most $O(n)$ `UNION`-operations and union can be done in a way that allows an effective search in step 5. of the algorithm. Again each of these operations can be done in $O(\log n)$ using mergeable heaps as additional data structure (see Aho, Hopcroft, Ullman [1974]). Thus, we have an $O(n \log n)$ algorithm.

Finally it is not difficult to prove by induction on n that the algorithm is correct.

If the coefficient matrix A in (3) has no row repetitions the corresponding tree has the property that each vertex different from a leaf has at least two sons. This implies that the number of non-leaf-nodes is smaller or equal than the number of leaves in the tree. Thus, we have a tree with $O(n)$ nodes where n is the number of variables in (3). This shows that (3) also can be solved in $O(n \log n)$ time.

3. Concluding Remarks

We have shown that min-cost flow problems with lower and upper capacities in trees with n nodes can be solved in $O(n \log n)$ time. This can be applied to solve special structured linear programs with n variables with the same time bound, if the corresponding tree is available and need not to be constructed. However this is the case in most applications where the structure is known when formulating the problem. If the tree is not available but matrix A has the MSP then the tree can be constructed in $O(n \log m)$ time by scanning the skyline while adding the leaves to the chain of nonleaves of the tree. To recognize that a matrix after a permutation of rows has MSP takes $O(mn)$ time. The steps of such a recognition algorithm are:

1. Calculate the number of ones in each row.
2. Order the rows according to nonincreasing row numbers.
3. Check inclusion property for consecutive rows.

Note that the sorting step can be done in $O(n + m)$ time by a hashing method because all row numbers are bounded by n .

Also property (2) can be recognized in $O(mn)$ steps but the method is more complicated.

It is a challenging question whether the time bound $O(n \log n)$ still can be improved. The fact, that knapsack problems with generalized upper bounds can be solved in $O(n)$ time suggest that such an improvement is not unlikely.

Acknowledgement

The authors would like to acknowledge the two anonymous referees for their valuable comments on an earlier version of this paper. This research was supported in part by the Deutsche Forschungsgemeinschaft (project COdiS) and by URC grant 453-2810.

References

- Aho, A. V., Hopcroft, J. E., Ullman, J. D.: The Design and Analysis of Computer Algorithms. Reading, Mass.: Addison-Wesley 1974.
- Brucker, P.: An $O(n \log n)$ -algorithm for the minimum cost flow problem in trees. In: Selected Topics in Operations Research and Mathematical Economics (G. Hammer, D. Pallaschke, eds.), pp. 299–306. Berlin: Springer 1984.
- Erenguc, S. S.: An algorithm for solving a structured class of linear programming problems. Operations Research Letters 6, 293–299 (1986).
- Kern, W.: An efficient algorithm for solving a special class of LP's. Computing 37, 219–226 (1986).
- Sleator, D. D., Tarjan, R. E.: Self-adjusting binary search trees. Journal of the ACM 32, 652–686 (1985).

Wolfgang W. Bein
University of New Mexico
Department of Computer Science
Albuquerque, NM 87 131
U.S.A.

Peter Brucker
Fachbereich Mathematik/Informatik
Universität Osnabrück
Albrechtstrasse 28
D-4500 Osnabrück
Federal Republic of Germany