Available online at www.sciencedirect.com
ScienceDirect
Operations Research Letters 36 (2008) 408–413
Operations Research Letters
www.elsevier.com/locate/orl

A faster polynomial algorithm for the unbalanced Hitchcock transportation problem

Ulrich Brenner

Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, 53113 Bonn, Germany

Received 27 June 2007; accepted 21 January 2008
Available online 13 February 2008

Abstract

We present a new algorithm for the Hitchcock transportation problem. On instances with n sources and k sinks, our algorithm has a worst-case running time of $O(nk^2(\log n + k \log k))$. It closes a gap between algorithms with running time linear in n but exponential in k and a polynomial-time algorithm with running time $O(nk^2 \log^2 n)$.

Keywords: Hitchcock transportation problem; Minimum cost flow problem

1. Introduction

The Hitchcock transportation problem is an uncapacitated minimum cost flow problem on a complete bipartite graph where all edges are directed from a set X of sources to a set Y of sinks. In this paper, we are interested in instances where one side of the bipartition is significantly smaller than the other one. With $n := |X|$ and $k := |Y|$, we may assume w.l.o.g. that $n \geq k$. The main contribution of this paper is a new algorithm for the Hitchcock transportation problem with a worst-case running time of $O(nk^2(\log n + k \log k))$. Starting with a zero flow, the algorithm adds the sources one after the other to a flow instance. We can show that each new source requires to change the flow only on a small number of edges.

For constant k, linear-time algorithms have been developed, but their running time is exponential in k. For instances where $k = o\left(\frac{\log^2 n}{\log \log n}\right)$ but k is not bounded by a constant, our algorithm is the fastest strongly polynomial algorithm.

The Hitchcock transportation problem is a classical optimization problem with various applications (see [2] and [18]). An important application where n can be huge (up to several millions) while k is at most in the range of some dozens, is the placement of VLSI chips. Here a large number of chip modules has to be assigned to some chip regions (see e.g. [6]). In such applications we are interested in an algorithm whose running time grows slowly in n and is polynomial in k.

The remainder of this paper is organized as follows: In Section 2, we introduce the notation used in this paper and give a formal definition of the Hitchcock transportation problem. Section 3 contains an overview of the previous algorithms. In Section 4, we describe our algorithm in detail and analyze its running time.

2. Minimum cost flow and transportation problems

We will first summarize the general notation used in this paper. We assume that the reader is familiar with the standard terms in graph theory (see e.g. [14] for an introduction).

A *flow network* is a quadruple (G, b, u, c) where G is a directed graph with vertex set $V(G)$ and edge set $E(G)$, $b : V(G) \to \mathbb{R}$ is a mapping with $\sum_{v \in V(G)} b(v) = 0$, $u : E(G) \to \mathbb{R}_{>0} \cup \{\infty\}$ is a *capacity function*, and $c : E(G) \to \mathbb{R}_{\geq 0}$ is a *cost function*. If $u(e) = \infty$ for all $e \in E(G)$, the flow network is called *uncapacitated*. A *flow* in (G, b, u, c) is a function $f : E(G) \to \mathbb{R}_{\geq 0}$ with $f(e) \leq u(e)$ for $e \in E(G)$ and $\sum_{e \in \delta^+(v)} f(e) - \sum_{e \in \delta^-(v)} f(e) = b(v)$ for $v \in V(G)$ (where $\delta^+(v)$ ($\delta^-(v)$) is the set of edges leaving (entering) v). In the minimum cost flow problem we are given a flow network (G, b, u, c) and ask for a flow f in (G, b, u, c) minimizing $c(f) := \sum_{e \in E(G)} f(e)c(e)$.

Let (G, b, u, c) be a flow network, and let $f : E(G) \to \mathbb{R}_{\geq 0}$ be a mapping with $f(e) \leq u(e)$ for $e \in E(G)$. $G_{f,u}$ denotes

E-mail address: brenner@or.uni-bonn.de.

0167-6377/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.orl.2008.01.011

the *residual graph* of f, i.e. $V(G_{f,u}) := V(G)$ and $E(G_{f,u}) := \{e \in E(G) \mid f(e) < u(e)\} \dot\cup \{\overleftarrow{e} \mid e \in E(G) \text{ and } f(e) > 0\}$ where $\overleftarrow{e}$ is the *reverse edge* of e, i.e. if e is directed from node v to node w, then $\overleftarrow{e}$ is directed from w to v. We denote the *residual capacity* of an edge e by $u_f(e)$, so $u_f(e) := u(e) - f(e)$ for edges $e \in E(G) \cap E(G_{f,u})$ and and $u_f(\overleftarrow{e}) := f(e)$ for reverse edges. The function $c_f : E(G_{f,u}) \to \mathbb{R}$ describes the *residual edges' costs*, so we define $c_f(e) := c(e)$ for edges $e \in E(G) \cap E(G_{f,u})$ and $c_f(\overleftarrow{e}) := -c(e)$ for reverse edges.

The Hitchcock transportation problem is a special case of the uncapacitated minimum cost flow problem on a bipartite directed graph where one side of the bipartition contains all the sources and the other side contains all sinks. So a flow network (G, b, u, c) is an instance of the Hitchcock transportation problem if the following conditions are met:

- there are sets X and Y such that $V(G) = X \dot\cup Y$ and $E(G) = (X \times Y)$,
- $b(x) > 0$ for $x \in X$, $b(y) < 0$ for $y \in Y$,
- $u(e) = \infty$ for $e \in X \times Y$.

3. Previous approaches

The Hitchcock transportation problem is a well-studied special case of the minimum cost flow problem. The fastest known strongly polynomial algorithm for general uncapacitated minimum cost flow problems is described by Orlin [16] and can be implemented to run in time $O(|V(G)|(\log|V(G)|)(|E(G)| + |V(G)|\log(|V(G)|)))$. Applied to the Hitchcock transportation problem, the algorithm has a running time of $O((n^2k + nk^2 + (n+k)^2\log(n+k))\log(n+k))$ which is $O(n^2\log n(k+\log n))$ for $k \le n$ and $O(n^2\log^2 n)$ if k is constant.

Other algorithms exploit the special structure of the minimum cost flow instances in the Hitchcock transportation problem. In [13], a strongly polynomial algorithm with running time $O(n^2k\log n)$ (for $k < n$) is described which is slightly better than the direct application of the general minimum cost flow algorithm mentioned above (if $k = o(\log n)$), but is still quadratic in n. Tokuyama and Nakano [18] improve this result (for $k = o(n/\log n)$) by showing how the Hitchcock transportation problem can be solved in time $O(nk^2\log^2 n)$.

There are some algorithms whose running time is $O(n)$ for fixed k but grows exponentially in k. As the Hitchcock transportation problem can be seen as a special case of the k-dimensional multiple choice linear programming problem, the algorithms proposed in [7] and [21] can be applied to it. Their running time is linear in n if k is fixed. For general k, the running time is not analyzed in the papers and is hard to estimate exactly, but no better estimation than $O(nk^2 2^k)$ seems to be possible. In [15], this result has been generalized to a linear-time algorithm for quadratic transportation problems where the size of one side is fixed. However, for Hitchcock transportation instances in which k is not constant, no better upper bound on the running time than $O(n3^k)$ is possible for their algorithm. All these algorithms with linear running time for fixed k are quite complicated, and even for moderate values of k their worst-case running time is large.

If $k = 2$, then the problem reduces to the fractional knapsack problem. The unweighted version of this problem (i.e., $b(x) = 1$ for all $x \in X$) has been solved in [5] with a linear-time algorithm. The authors of [1,4,11] show how the algorithm for the unweighted version can be used as a subroutine for a linear-time algorithm of the fractional knapsack problem with weights. The algorithm in [5] can also be applied almost directly to weighted instances (see [14]).

Tokuyama and Nakano [17] describe an algorithm that solves the Hitchcock transportation problem in time $O(n(k!)^2)$ if $b(x) = 1$ for all $x \in X$ and $-b(y) \in \mathbb{N}$ for all $y \in Y$. Ahuja et al. [3] consider several flow algorithms and examine how fast they can solve flow problems on bipartite graphs. If all edge costs are integral, they show how the minimum cost flow algorithms presented in [9] can be modified for the Hitchcock transportation problem to run in time $O(nk^2 + k^3\log(k\max\{c((x,y)) \mid x \in X, y \in Y\}))$ or in time $O(nk^2\log(k\max\{d((x,y)) \mid x \in X, y \in Y\}))$ even if there are capacities on the graph edges. These algorithms are not strongly polynomial but if $\max\{c((x,y)) \mid x \in X, y \in Y\}$ is fixed, we get an algorithm with running time $O(nk^2)$, so the running time grows linearly in n. There is a linear-time algorithm described by Vygen [19,20] that solves the Hitchcock transportation problem for instances where $Y = \{y_1, y_2, y_3, y_4\}$ and $c((x,y_1)) + c((x,y_3)) = c((x,y_2)) + c((x,y_4))$. This is particularly the case if the elements of X represent points in the plane, the elements of Y represent the four quadrants, and c is the L_1-distance. These are very strong restrictions of the problem, but this special case has an application in VLSI placement.

For the bottleneck version of the Hitchcock transportation problem where the objective function to be minimized is $\max\{c((x,y)) \mid f((x,y)) > 0\}$, a linear-time algorithm for instances with constant k has been described in [10].

4. Our approach

We will show how the Hitchcock transportation problem can be solved in time $O(nk^2(\log n + k\log k))$. More precisely, our algorithm has a running time of $O(n(k^2\log n + k^3 + \mathrm{MCF}(k, k^2)))$ where $\mathrm{MCF}(k, k^2)$ is the running time that is needed to solve an uncapacitated minimum cost flow problem on a graph with $O(k)$ nodes and $O(k^2)$ edges. Together with Orlin's algorithm [16], this leads to the running time of $O(nk^2(\log n + k\log k))$.

4.1. Almost unsplitted solutions

It will be necessary to control the number of elements $x \in X$ with outgoing flow on more than one edge. Let f be a solution of the Hitchcock transportation problem. For $x \in X$, we define $\tau_f(x) := |\{y \in Y \mid f((x,y)) > 0\}|$. Let $F_f := \{x \in X \mid \tau_f(x) > 1\}$ be the set of elements $x \in X$ with outgoing flow into more than one edge.

Vygen [19,20] has shown that for constant k an optimum solution f can be transformed in linear time to an optimum

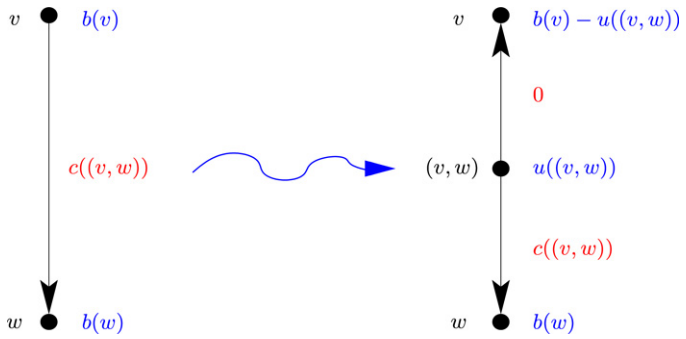

Fig. 1. Transformation from an edge (v, w) with capacity $u((v, w)) < \infty$ to two uncapacitated edges. For the nodes, supply and demand values are shown, and for the edges, edge costs are shown.

solution g with $|F_g| \leq k - 1$ (in fact, Vygen showed this for $k = 4$, but the construction is the same for any k).

For the analysis of our algorithm, we need a slightly stronger result as we have to control the numbers of edges with positive flow that leave nodes in F_g:

Lemma 1. *Given an instance (G, b, u, c) of the Hitchcock transportation problem, an optimum solution f, and the set F_f, we can transform f in time $O(k^2 \cdot |F_f|)$ into an optimum solution g of (G, b, u, c) such that*

(i) $|F_g| \leq k - 1$, *and*
(ii) $\sum_{x \in F_g} \tau_g(x) \leq 2k - 2$.

Proof. We start by setting $g := f$. During the transformation, g will always stay an optimum solution. We construct an undirected bipartite graph H_g on the vertex set $V(H_g) := F_g \cup Y$ and edge set $E(H_g) = \{\{x, y\} \mid g((x, y)) > 0\}$.

Claim: If H_g does not contain a cycle, g meets conditions (i) and (ii).

Proof of the claim: Assume that there is no cycle in H_g, so we have $|E(H_g)| \leq |V(H_g)| - 1$. We can conclude: $2|F_g| \leq \sum_{x \in F_g} \tau_g(x) = |E(H_g)| \leq |V(H_g)| - 1 = |F_g| + k - 1$. This shows that $|F_g| \leq k - 1$ and hence $\sum_{x \in F_g} \tau_g(x) \leq |F_g| + k - 1 \leq 2k - 2$. This proves the claim.

The claim shows that we are done if there is no cycle in H_g. Hence let us assume that H_g contains an undirected cycle consisting of the node set $\{y_1, x_1, y_2, x_2, \ldots, x_j, y_{j+1} = y_1\}$ and the edge set $\{\{x_i, y_i\} \mid i = 1, \ldots j\} \cup \{\{x_i, y_{i+1}\} \mid i = 1, \ldots j\}$. Then, for sufficiently small $\epsilon > 0$ also g' and g'' with $g'((x_i, y_i)) := g((x_i, y_i)) + \epsilon$, $g'((x_i, y_{i+1})) := g((x_i, y_{i+1})) - \epsilon$, $g''((x_i, y_i)) := g((x_i, y_i)) - \epsilon$, $g''((x_i, y_{i+1})) := g((x_i, y_{i+1})) + \epsilon$, for $i = \{1, \ldots, j\}$, and $g'((x, y)) := g''((x, y)) := g((x, y))$, for all other edges $(x, y) \in G$, are feasible solutions. Since g is optimum and the cost of g is the arithmetic mean of the costs of g' and g'', both g' and g'' are also optimum. If we choose ϵ as big as possible, then we have $\sum_{x \in F_{g'}} \tau_{g'}(x) < \sum_{x \in F_g} \tau_g(x)$ or $\sum_{x \in F_{g''}} \tau_{g''}(x) < \sum_{x \in F_g} \tau_g(x)$.

Then, we set $g := g'$ or $g := g''$, respectively, and iterate this method until $\sum_{x \in F_g} \tau_g(x) \leq 2k - 2$. As we have $\tau_g(x) \geq 2$ for each $x \in F_g$, then condition (i) will also be met. One iteration can be performed in time $O(k)$, and hence the whole computation takes time $O(k^2 \cdot |F_f|)$. □

4.2. Handling edges with capacities

In our approach, n minimum cost flow problems on graphs whose size depends only on k have to be solved, and the running time depends on how fast we can compute minimum cost flows on these small instances. In the small instances, most of the edges are uncapacitated but there will be $O(k)$ edges with finite capacities. In order to get rid of the capacitated edges, we apply the following trick due to Ford and Fulkerson [8] that demonstrates how minimum cost flow instances with capacities can be transformed equivalently into (larger) uncapacitated instances:

Lemma 2 (*[8]*). *Let (G, b, u, c) be a minimum cost flow instance with finite capacities on exactly σ edges. Then, there is an equivalent uncapacitated minimum cost flow instance with $|V(G)| + \sigma$ nodes and $|E(G)| + \sigma$ edges.*

Proof. Let E_{cap} be the set of edges with finite capacities (so we have $u(e) = \infty$ for $e \in E(G) \setminus E_{\text{cap}}$). We define a graph G' with $V(G') := V(G) \cup E_{\text{cap}}$ and

$$E(G') = (E(G) \setminus E_{\text{cap}}) \cup \{((v, w), v) \mid (v, w) \in E_{\text{cap}}\} \cup \{((v, w), w) \mid (v, w) \in E_{\text{cap}}\}.$$

For $(v, w) \in E_{\text{cap}}$, we set $c'(((v, w), w)) := c((v, w))$ and $c'(((v, w), v)) := 0$, and for $e \in E(G) \setminus E_{\text{cap}}$, we set $c'(e) := c(e)$. For $(v, w) \in E_{\text{cap}}$, we define $b'((v, w)) = u((v, w))$ and for $v \in V(G)$ we set $b'(v) = b(v) - \sum_{w:(v,w) \in E_{\text{cap}}} u((v, w))$. Of course, for all $e \in E(G')$, we define $u(e) := \infty$. Fig. 1 illustrates the transformation for a single edge (assuming that no other edges leaving v and w are replaced). Obviously, (G', b', u', c') is a flow network, and G' has $|V(G)| + \sigma$ nodes and $|E(G)| + \sigma$ edges. It remains to prove that the two instances are equivalent.

First, let f be a flow in (G, b, u, c). Then, we set $f'(((v, w), w)) := f((v, w))$ and $f'(((v, w), v)) := u((v, w)) - f((v, w))$ for $(v, w) \in E_{\text{cap}}$, and $f'(e) := f(e)$ for $e \in E(G) \setminus E_{\text{cap}}$. It is easy to see that f' is a flow in (G', b', u', c') with $c'(f') = c(f)$.

For the other direction, let f' be a flow in (G', b', u', c'). We set $f((v,w)) := f'(((v,w),w))$ for $(v,w) \in E_{\text{cap}}$, and $f(e) := f'(e)$ for $e \in E(G) \setminus E_{\text{cap}}$. Again, it is not difficult to see that f is a flow in (G,b,u,c) with $c(f) = c'(f')$. □

4.3. The algorithm

In our algorithm, we first sort the nodes in X such that $b(x_1) \geq b(x_2) \geq \cdots \geq b(x_n)$ and consider the elements of X in this order, one after the other. The algorithm consists of n phases, and at the end of phase i, we have an optimum flow f in $(G, \tilde{b}_i, u, c)$ where $\tilde{b}_i : V(G) \to \mathbb{R}$ is a mapping with $\tilde{b}_i(x_j) = b(x_j)$ for $j \in \{1,\dots,i\}$, $\tilde{b}_i(x_j) = 0$ for $j \in \{i+1,\dots,n\}$ and $\tilde{b}_i(y) \geq b(y)$ for $y \in Y$.

Consider a phase i (with $i \in \{1,\dots,n\}$) and assume that we are already given a minimum cost flow f in $(G, \tilde{b}_{i-1}, u, c)$. We modify f by computing a minimum cost flow in a network (G^i, b^i, u^i, c^i) whose size does not depend on n.

For $y \in Y$ let $M_i(y) := \{x \in X \mid f((x,y)) = b(x)\}$. We define $G^i = (V(G^i), E(G^i))$ by

- $V(G^i) := \{x_i\} \cup F_f \cup Y \cup \{t\}$.
- $E(G^i) := \big((F_f \cup \{x_i\}) \times Y\big)$
  $\cup \{(y,x) \in Y \times F_f \mid f((x,y)) > 0\}$
  $\cup (Y \times \{t\})$
  $\cup \{(y_1,y_2) \in Y \times Y : y_1 \neq y_2, M_i(y_1) \neq \emptyset\}$.

For $e = (v,w) \in E(G^i)$ we define edge capacities u^i and edge costs c^i by

- $u^i(e) := \begin{cases} f((w,v)) & \text{if } e \in Y \times F_f \\ -b(v) - \sum_{x \in X} f((x,v)) & \\ \quad \text{if } (v,w) \in Y \times \{t\} \\ \infty \quad \text{else} \end{cases}$
- $c^i(e) := \begin{cases} c((v,w)) & \text{if } e \in X \times Y \\ -c((w,v)) & \text{if } e \in Y \times X \\ 0 \quad \text{if } e \in Y \times \{t\} \\ \min\{c(x,w) - c(x,v) : x \in M_i(v)\} & \\ \quad \text{if } e \in Y \times Y. \end{cases}$

Finally, we define $b^i(v) := \begin{cases} b(x_i) & \text{if } v = x_i \\ -b(x_i) & \text{if } v = t \\ 0 & \text{else.} \end{cases}$

The size of G^i depends on the number of elements of F_f. As mentioned above, if $|F_f| > k-1$, we may apply Lemma 1 and find a flow f' of the same cost with $|F_{f'}| \leq k-1$. After each phase of the algorithm, we will call a subroutine ADJUST (f) that applies the algorithm in the proof of Lemma 1, so we always have $|F_f| \leq k-1$ and $\sum_{x \in F_f} \tau_f(x) \leq 2k-2$. Thus, we have $|V(G^i)| \leq k+2+(k-1) = 2k+1$ and $|E(G^i)| = O(k^2)$.

TRANSPORTATION FLOW ALGORITHM

Input: An instance (G,b,u,c) of the Hitchcock transportation problem.

Output: A minimum cost flow f in (G,b,u,c).

① Set $f(e) := 0$ for all $e \in E$.

② Sort the elements of X such that $X = \{x_1, x_2, \dots, x_n\}$ with $b(x_1) \geq b(x_2) \geq \cdots \geq b(x_n)$.

③ FOR($i = 1,\dots,n$)
{
  Compute a flow network (G^i, b^i, u^i, c^i) as described above.
  Compute a minimum cost flow g in (G^i, b^i, u^i, c^i) such that the edges with positive flow do not contain a directed cycle.
  FOR($e = (v,w) \in E(G^i)$ with $g(e) > 0$)
  {
    IF($e \in X \times Y$)
      Set $f(e) := f(e) + g(e)$.

    IF($e \in Y \times F_f$)
      Set $f((w,v)) := f((w,v)) - g(e)$.

    IF($e \in Y \times Y$)
      Choose $x \in M_i(v)$ with $c^i(e) = c((x,w)) - \text{cost}((x,v))$.
      Set $f((x,v)) := f((x,v)) - g(e)$.
      Set $f((x,w)) := f((x,y)) + g(e)$.

  }
  ADJUST(f)
}

We prove our main results:

Theorem 3. *The* TRANSPORTATION FLOW ALGORITHM *works correctly.*

Proof. We show by induction on i: At the end of iteration i of the loop in ③, f is a minimum cost flow in a network $(G, \tilde{b}_i, u, c)$ where $\tilde{b}_i : V(G) \to \mathbb{R}$ is a mapping with $\tilde{b}_i(x_j) = b(x_j)$ for $j \in \{1,\dots,i\}$, $\tilde{b}_i(x_j) = 0$ for $j \in \{i+1,\dots,n\}$ and $\tilde{b}_i(y) \geq b(y)$ for $y \in Y$.

Let $i \in \{1,\dots,n\}$, let $\tilde{f}$ be the flow before iteration i, and f at the end of iteration i. We can assume that $\tilde{f}$ is a minimum cost flow, and hence the residual graph of $\tilde{f}$ does not contain a cycle of negative cost.

G^i does not contain a negative cycle because any cycle in G^i corresponds to a cycle in the residual graph of $\tilde{f}$ of the same cost. Let h be any minimum cost flow in (G^i, b^i, u^i, c^i). The support of h (i.e. a subgraph of G^i whose edge set consists of all edges with positive h-flow) cannot contain a cycle of negative weight (because G^i does not contain such a cycle) nor a cycle of positive weight (because h is optimal). Hence, each cycle in the support of h has weight 0. Such cycles can be removed by reducing the flow value on the cycle edges without losing optimality. Therefore, we can compute the minimum cost flow g in (G^i, b^i, u^i, c^i).

Since there is no cycle in the support of g, we have $\sum_{e\in\delta^+(v)} g(e) \le \sum_{w\in V(G):b(w)<0} -b(w) = b(x_i)$ for each $v \in V(G^i)$.

We prove that f is nonnegative. During iteration i, the flow on edges $(w, v) \in F_{\tilde{f}} \times Y$ is decreased by $g((v, w))$. However, since $g((v, w)) \le u^i((v, w)) \le \tilde{f}((w, v))$, the flow will stay nonnegative on these edges. The only other edges where flow $\tilde{f}$ is decreased are of the type (x, v) with $x \in M_i(v)$. But if $x \in M_i(v)$, we have $\tilde{f}((x, v)) = b(x) \ge b((x_i)) \ge \sum_{e\in\delta^+_{G^i}} g(e)$, so f cannot be negative on these edges. This shows that f is a nonnegative mapping.

We can assume that $\tilde{f}$ is a flow network $(G, \tilde{b}_{i-1}, u, c)$ where $\tilde{b}_{i-1} : V(G) \to \mathbb{R}$ is a mapping with $\tilde{b}_{i-1}(x_j) = b(x_j)$ for $j \in \{1, \dots, i-1\}$, $\tilde{b}_i(x_j) = 0$ for $j \in \{i, \dots, n\}$ and $\tilde{b}_{i-1}(y) \ge b(y)$ for $y \in Y$. Now we define $\tilde{b}_i : V(G) \to \mathbb{R}$ by $\tilde{b}_i(x_j) := b(x_j)$ for $j \in \{1, \dots, i\}$, $\tilde{b}_i(x_j) := 0$ for $j \in \{i+1, \dots, n\}$ and $\tilde{b}_i(y) := \tilde{b}_{i-1}(y) - g((y, t))$ for $y \in Y$. Then, we have $\tilde{b}_i(y) = \tilde{b}_{i-1}(y) - g((y, t)) \ge \tilde{b}_{i-1}(y) - u^i((y, t)) = \tilde{b}_{i-1}(y) + b(y) + \sum_{x\in X} \tilde{f}((x, y)) = b(y)$ for $y \in Y$. Moreover, in step ③, the f-balance of nodes (i.e. the total outcoming f-flow minus the total incoming f-flow) is changed only for x_i (where it is increased by $\sum_{e\in\delta^+_{G^i}(x_i)} g(e) = b(x_i)$) and for nodes $y \in Y$ (where it is decreased by $g((y, t))$). Therefore, f is a flow in $(G, \tilde{b}_i, u, c)$.

Now assume that f is not a minimum cost flow. Then, there must be a directed cycle of negative cost in the residual graph of f (this is a well-known result, see [12]). Let C be such a cycle. We will show that we can find a negative cycle in the residual graph of g which is of course a contradiction to the optimality of g.

Let $x \in V(C) \cap X$ be a vertex. In C, vertex x has an incoming edge (y_1, x) and an outgoing edge (x, y_2). If $x \in F_{\tilde{f}}$, then both the (y_1, x) and the edge (x, y_2) belong to the residual graph of g.

If $x \notin F_{\tilde{f}}$, then there must be a vertex $y_3 \in Y$ with $\tilde{f}((x, y_3)) = b(x_3)$ (because x must have had positive outgoing flow before iteration i).

We distinguish two cases:

Case 1: $y_3 = y_1$. Then, we replace x and the edges (y_1, x) and (x, y_2) by the edge (y_1, y_2). Because $x \in M_i(y_1)$, we have $(y_1, y_3) \in E(G^i_{g,u^i})$. Moreover, we have $c^i_g((y_1, y_2)) = \min\{c((x', y_2)) - c((x', y_1)) \mid x' \in M_i(y_1)\} \le c((x, y_2)) - c((x, y_1)) = c_g((y_1, x)) + c_g((x, y_2))$.

Case 2: $y_3 \ne y_1$. In that case, we replace (y_1, x) and (x, y_2) by the two edges (y_1, y_3) and (y_3, y_2). We have $\tilde{f}((x, y_1)) = 0$ and $\tilde{f}((x, y_3)) = b(x)$ but $f((x, y_1)) > 0$, and hence by construction $g((y_3, y_1)) > 0$ which implies that $(y_1, y_3) \in E(G^i_{g,u^i})$. As $M_i(y_1) \ne \emptyset$, edge (y_3, y_2) is as well in the residual graph of g. In addition, we have $c^i_g((y_1, y_3)) + c^i_g((y_3, y_2)) = -c^i((y_3, y_1)) + c^i((y_3, y_2)) \le c((x, y_3)) - c((x, y_1)) - c((x, y_3)) + c((x, y_2)) = -c((x, y_1)) + c((x, y_2)) = c_g((y_1, x)) + c_g((x, y_2))$.

In both cases, we could replace the edges incident to x by edges in the residual graph of g without increasing the total cost. If we do this for every $x \in (V(C) \cap X) \setminus F_{\tilde{f}}$, then we get a closed walk of negative total cost in the residual graph of g. This closed walk must contain a cycle of negative cost which is a contradiction to the optimality of g. □

Theorem 4. *The* TRANSPORTATION FLOW ALGORITHM *can be implemented to run in time* $O(nk^2(\log n + k \log k))$.

Proof. Obviously, step ① can be done in time $O(nk)$ and step ② takes time $O(n \log n)$. The construction of G^i can be done in time $O(k^2 \log n)$ for each iteration if one stores each set $M_i(y)$ in $k-1$ heaps: For each ordered pair $y, y' \in Y$ with $y \ne y'$, we use a heap to store the elements x of $M_i(y)$ with $\text{key}_{y,y'}(x) = c((x, y')) - c((x, y))$. For the worst-case running time, it does not matter which kind of heaps we use as long as finding and deleting an element with smallest key and inserting an element can be done in time $O(\log n)$ for a heap with n elements.

Apart from the k edges from nodes in Y to the node t, the only edges in G^i with finite capacity are the edges in $(Y \times F_f) \cap E(G^i)$. According to Lemma 1(ii), the number of edges in $(Y \times F_f) \cap E(G^i)$ is (after an adjustment) bounded by $2k-2$, so there are $O(k)$ edges with finite capacities in graph G^i. Hence, we can apply Lemma 2 and replace them by $O(k)$ additional vertices and edges. Putting this together, we can compute g in ③ by solving a minimum cost flow problem on an uncapacitated instance with $O(k)$ vertices and $O(k^2)$ edges. Using Orlin's algorithm [16], this can be done in time $O(k^3 \log k)$.

Applying Lemma 1, the flow can be adjusted in time $O(k^2 \cdot |F_{f_{i-1}}|) = O(k^3)$ (where $\tilde{f}$ is the flow be the iteration). To update the heaps after a flow augmentation, we need $O(k^2)$ delete operations. The number of insert operations after a flow augmentation is as well $O(k^2)$: only the elements of $V(G^i)$ can be inserted into a heap, and for each heap that stores a set $M_i(y)$, at most one of the elements of $(X \cap V(G^i)) \setminus F_{\tilde{f}}$ can be inserted into $M_i(y)$. Since there are at most $k-1$ elements in $F_{\tilde{f}}$, and each of them can be added to at most $k-1$ heaps, we need at most $O(k^2)$ insert operations. As no heap contains more than n elements, each operation can be done in time $O(\log n)$. Therefore, the $k(k-1)$ heaps can be updated in time $O(k^2 \log n)$. □

It should be noted that our algorithm can be easily adapted to instances with finite capacities on the edges from X to Y. If there are σ edges with finite capacity, we only have to add $O(\sigma)$ nodes and $O(\sigma \cdot k)$ edges to the minimum cost flow instances that are solved in the iterations. This is not efficient if σ is big, but for constant $\sigma \in O(k)$ we would still get the running time $O(nk^2(\log n + k \log k))$.

Acknowledgements

The author would like to thank Prof. Satoru Fujishige and Prof. Jens Vygen for their helpful comments.

References

[1] D.L. Adolphson, G.N. Thomas, A linear time algorithm for a $2 \times n$ transportation problem, SIAM Journal on Computing 6 (1977) 481–486.

[2] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, Network Flows, Prentice-Hall, Englewood Cliffs, 1993.
[3] R.K. Ahuja, J.B. Orlin, C. Stein, R.E. Tarjan, Improved algorithms for bipartite network flow, SIAM Journal on Computing 23 (1994) 906–933.
[4] E. Balas, E. Zemel, An algorithm for large zero-one knapsack problems, Operations Research 28 (1980) 1130–1154.
[5] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, R.E. Tarjan, Time bounds for selection, Journal of Computer and System Science 7 (1973) 448–461.
[6] U. Brenner, M. Struzyna, 2005. Faster and better global placement by a new transportation algorithm, in: Proceedings of the 42nd Design Automation Conference, ACM 2005, pp. 591–596.
[7] M.E. Dyer, An $O(n)$ algorithm for the multiple-choice knapsack linear program, Mathematical Programming 29 (1984) 57–63.
[8] L.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.
[9] A.V. Goldberg, R.E. Tarjan, Finding minimum-cost circulations by successive approximation, Mathematics of Operations Research 15 (1990) 430–466.
[10] D.S. Hochbaum, G.J. Woeginger, An optimal algorithm for the bottleneck transportation problem with a fixed number of sources, Operations Research Letters 24 (1999) 25–28.
[11] D.B. Johnson, T. Mizoguchi, Selecting the Kth element in $X + Y$ and $X_1 + X_2 + \cdots + X_m$, SIAM Journal on Computing 7 (1978) 147–153.
[12] M. Klein, A primal method for minimum cost flows, with applications to the assignment and transportation problems, Management Science 14 (1967) 205–220.
[13] H.P. Kleinschmidt, H. Schannath, A strongly polynomial algorithm for the transportation problem, Mathematical Programming 68 (1995) 1–13.
[14] B. Korte, J. Vygen, Combinatorial Optimization: Theory and Algorithms, 4th edition, Springer, Berlin, 2008.
[15] N. Megiddo, A Tamir, Linear time algorithms for some separable quadratic programming problems, Operation Research Letters 13 (1993) 203–211.
[16] J.B. Orlin, A faster strongly polynomial minimum cost flow algorithm, Operations Research 41 (1993) 338–350.
[17] T. Tokuyama, J Nakano, Geometric algorithms for a minimum cost assignment problem, in: Proceedings of the Seventh Symposium on Computational Geometry, 1991. pp. 262–271.
[18] T. Tokuyama, J. Nakano, Efficient algorithms for the Hitchcock transportation problem, SIAM Journal on Computing 24 (1995) 563–578.
[19] J. Vygen, Plazierung in VLSI-Design und ein zweidimensionales Zerlegungsproblem, Doctoral Thesis, University of Bonn, 1996.
[20] J. Vygen, Geometric quadrisection in linear time with application to VLSI placement, Discrete Optimization 2 (2005) 362–390.
[21] E. Zemel, An $O(n)$ algorithm for the linear multiple choice knapsack problem and related problems, Information Processing Letters 18 (1984) 123–128.