DEPARTMENT OF OPERATIONS RESEARCH
UNIVERSITY OF AARHUS

Working Paper no. 2006 / 5

2006 / 10 / 25

# Single-Sink Fixed-Charge Transportation: Applications and Exact Solution Algorithms

Andreas Klose

ISSN 1600-8987

# Single-Sink Fixed-Charge Transportation: Applications and Exact Solution Algorithms

Andreas Klose

*Department of Mathematical Sciences, University of Aarhus,*
*Ny Munkegade, Building 1530, 8000 Aarhus C, Denmark*
*aklose@imf.au.dk*

**Abstract**

The single-sink fixed-charge transportation problem (SSFCTP) consists in finding a minimum cost flow from a number of supplier nodes to a single demand node. Shipping costs comprise costs proportional to the amount shipped as well as a fixed charge. Although the SSFCTP is an important special case of the well known fixed-charge transportation problem, only a few methods for solving this problem have been proposed in the literature. This paper first summarizes applications of the single-sink fixed charge transportation problem in the fields of purchasing, manufacturing, and transportation. Exact solution methods based on dynamic programming and implicit enumeration are afterwards discussed and improved by means of problem reduction techniques and additional bounds. Finally, the performance of the various solution methods is compared in a series of computational experiments.

*Key words:* fixed-charge transportation problem, dynamic programming, implicit enumeration, branch-and-bound, penalties

## 1 Introduction

The single-sink fixed-charge transportation problem (SSFCTP) is a special case of the well-known fixed-charge transportation problem (FCTP); it is a FCTP with just one sink node or, alternatively, just one source node. The problem is to decide on the amounts $x_j \geq 0$ of shipments to be made from a given set of suppliers $j = 1, \ldots, n$ to a single sink in such a way that the suppliers' capacities $b_j$ are respected and the sink's demand $D$ is satisfied at minimum shipment cost. The cost of shipping $x_j > 0$ units from a supplier $j$ to the sink involves a fixed charge $f_j$ as well as costs $c_j x_j$ that are proportional to the quantity shipped. The mathematical formulation of the problem is:

$$Z = \min \sum_{j=1}^{n} \Big( c_j x_j + f_j y_j \Big) \tag{1}$$

$$\text{s.t.:} \sum_{j=1}^{n} x_j = D\,, \tag{2}$$

$$0 \le x_j \le b_j y_j\,, \quad j = 1, \dots, n\,, \tag{3}$$

$$y_j \in \{0,1\}\,, \qquad j = 1, \dots, n\,. \tag{4}$$

Without loss of generality, it can be assumed that $c_j \ge 0$, $f_j \ge 0$, and $0 < b_j \le D$ for $j = 1, \dots, n$. Furthermore, it is assumed that the capacities $b_j$ and the demand $D$ are integer-valued, and that $\sum_{j=1}^{n} b_j - b_k \ge D$ holds for each $k = 1, \dots, n$.

The SSFCTP is closely related to the binary knapsack problem (BKP). It is easy to see that the SSFCTP always has an optimal solution $x^*$ such that $0 < x_j^* < b_j$ holds for at most one supplier $j$. In case that $x_j^* \in \{0, b_j\}$ for $j = 1, \dots, n$, the problem can be reduced to a minimization knapsack problem (min-KP) by substituting $x_j$ with $b_j y_j$. Furthermore, in case of zero or constant unit transportation costs $c_j = 0$ for $j = 1, \dots, n$, the SSFCTP also reduces to a min-KP. The SSFCTP is thus NP-hard, and can be solved in pseudo-polynomial time by means of dynamic programming.

Despite its importance as a relaxation of the FCTP, the SSFCTP did not attract much attention in the literature. Herer et al. [13] describe a number of applications of the SSFCTP in manufacturing and transporation. For the purposes of computing optimal solutions, they propose an implicit enumeration algorithm that improves an older method of Haberl [12]. Recently, Alidaee and Kochenberger [1] introduced a dynamic programming method capable to solve the problem in $O(nD)$ time. These two methods are revisited in Sect. 6.2 and Sect. 6.1. Furthermore, a new implicit enumeration algorithm for the SSFCTP is introduced in Sect. 6.3. This method relies on ideas developed by Martello and Toth [16] (see also Martello and Toth [17, pp. 32 ff.]) for the binary knapsack problem (BKP). In addition, it is shown how reduced cost information can be exploited for the purposes of reducing the problem size (Sect. 5) as well as reducing the search space of the dynamic programming method (Sect. 6.1). To this end, computationally inexpensive methods for computing lower and upper bounds are required. These methods are briefly sketched in Sect. 3 and Sect. 4. A detailed computational comparison of the different solution methods is then given in Sect. 7, and Sect. 8 summarizes the findings. Beforehand, however, a number of applications of the SSFCTP are described in order to underpin its importance.

## 2 Applications

Herer et al. [13] mention a number of applications of the SSFCTP in the area of manufacturing and transportation. Especially, they describe applications of this problem in the fields of

- supplier selection,
- product distribution/fleet selection,
- process selection.

Moreover, the SSCTP is a relaxation of the FCTP. The FCTP is in turn a special case of the fixed-charge network flow problem, a class of optimisa- tion problems that have a huge number of applications in the area of tele- communication and supply chain planning.

## *2.1 Supplier selection*

Consider a firm that has a periodic demand of $D$ units for an item. The item can be procured from different suppliers $j = 1, \ldots, n$. A supplier $j$ cannot deliver more than $b_j$ units per period. The problem is to decide on the periodic quantity $x_j$ of the item to be bought from each of the suppliers $j = 1, \ldots, n$ in such a way that the total cost is minimised. The costs considered are the supplier management, total periodic purchasing, ordering and inventory holding costs:

- The supplier management cost $f_j$ is independent of the amount purchased and incurred whenever supplier $j$ is used.
- If $p_j$ denotes the unit purchasing cost, $p_j x_j$ gives the cost of purchasing $x_j$ units per period from supplier $j$.
- The ordering and inventory carrying costs are derived using the economic order quantity model. Let $F_j$ be the fixed cost associated with each order from supplier $j$, and let $Q_j$ denote the size of an order placed at supplier $j$ every $T_j$ units of time. If $x_j > 0$ units are bought every period, $T_j x_j = Q_j$ results and $F_j x_j / Q_j$ gives the periodic fixed ordering cost.
- If the inventory is always depleted before the next order is placed, $Q_j Q_j/(2D)$ is the inventory per cycle of items purchased from supplier $j$. Thus, if $h_j$ denotes the cost per period of holding one unit purchased from supplier $j$, $h_j Q_j x_j/(2D)$ gives the inventory cost per period for the units procured from supplier $j$.

Minimising the sum

$$\left(\frac{F_j}{Q_j} + hj\frac{Q_j}{2D}\right)x_j$$

of the ordering and purchasing cost then yields the optimal order quantity

$$Q_j^* = \begin{cases} \sqrt{2F_j D/h_j} & \text{, if } x_j > 0\,, \\ 0, \text{ if } x_j = 0\,. \end{cases}$$

The supplier selection problem that is to decide which suppliers should be used and how many units should be bought per period from each supplier, reduces then to a SSFCTP with the fixed-charges given by the supplier management costs $f_j$ and the unit supply cost $c_j$ computed from

$$c_j = p_j + \left(\frac{F_j}{Q_j^*} + h_j\frac{Q_j^*}{2D}\right) = p_j + \sqrt{2F_j h_j/D}\,.$$

### *2.2 Product distribution/fleet selection*

The next application described by Herer et al. [13] is a typical fixed-charge transportation problem. A firm may choose a set of trucks or carriers from a given set of available trucks in order to meet a number of daily delivery needs at minimum cost. Each truck may only be used once within the day and has a capacity of $b_j$ units. A fixed-charge of $f_j$ units is expended for each truck used, and a variable cost of $c_j$ is incurred for each unit assigned to a truck. The problem is then to select the trucks to be used and to allocate the required daily transport capacity of $D$ units among the trucks such that the sum of the variable and fixed costs is minimised.

### *2.3 Process selection*

As a third application of the SSFCTP, Herer et al. [13] mention the problem of process selection. A predetermined amount of $D$ units of a product can be produced by means of different processes $j = 1, \ldots, n$. Each process $j$ has a given capacity of $b_j$ units per period. The unit production cost of a process is denoted by $c_j$, and the fixed costs $f_j$ reflect the periodic discounted purchase and maintenance costs.

Chauhan and Proth [5] consider a related decision problem. A given set of providers feed a single manufacturing unit. However, the quantity a provider can deliver must lie between a minimum and a maximum value. The upper bound is a technical limit, whereas the lower bound stems from economical considerations. Furthermore, the costs incurred to feed the manufacturing unit are now assumed to be a concave function of the quantity received from a provider.

### *2.4 Relaxation of the FCTP*

The fixed-charge transportation problem (FCTP) extends the SSFCTP by considering $m > 1$ demand nodes. The problem can be formulated as the following linear mixed-integer program:

$$\begin{aligned}
Z = \min & \sum_{j=1}^{n}\sum_{k=1}^{m}\Big(c_{jk}x_{jk} + f_{jk}y_{jk}\Big) \\
\text{s.t.:} & \sum_{j=1}^{n} x_{jk} = d_k\,, \qquad k = 1, \ldots, m\,, \\
& \sum_{k=1}^{m} x_{jk} = s_j\,, \qquad j = 1, \ldots, n\,, \\
& 0 \le x_{jk} \le b_{jk}y_{jk}\,,\ j = 1, \ldots, n\,, k = 1, \ldots, m\,, \\
& y_{jk} \in \{0, 1\}\,, \qquad j = 1, \ldots, n\,, k = 1, \ldots, m\,,
\end{aligned}$$

where $d_k$ is customer $k$'s demand, $s_j$ is the amount supplied by supplier $j$, $c_{jk}$ is the cost of transporting one unit on arc $(j,k)$, $f_{jk}$ denotes the fixed cost on arc $(j,k)$, $x_{jk}$ is the amount to be transported on arc $(j,k)$ and $y_{jk}$ is a binary variable equalling 1 if $x_{jk} > 0$.

Consider the following Lagrangean decomposition of the above program:

(1) Create copy variables $x'_{jk}$ and $y'_{jk}$ for $j = 1, \ldots, n$ and $k = 1, \ldots, m$.
(2) Write the demand constraints in the original variables $x_{jk}$ and state the supply constraints in the copy variables $x'_{jk}$. All other constraints are restated in dependency of the orginal as well as the copy variables.
(3) Dualize the identity constraints $x_{jk} - x'_{jk} = 0$ and $y_{jk} - y'_{jk} = 0$ with Lagrangean multipliers $\lambda_{jk}$ and $\mu_{jk}$, respectively.

It is then easy to see that the resulting Lagrangean subproblem decomposes into $n$ single-source fixed-charge transporation problems

$$\begin{aligned}
\min\ & \sum_{k=1}^{m} \lambda_{jk} x_{jk} + \mu_{jk} y_{jk} \\
\text{s.t.:}\ & \sum_{k=1}^{m} x_{jk} = s_j \,, \\
& 0 \le x_{jk} \le b_{jk} y_{jk} \,, \quad k = 1, \ldots, m \,, \\
& y_{jk} \in \{0,1\} \,, \qquad\quad k = 1, \ldots, m \,,
\end{aligned}$$

for each supplier $j = 1, \ldots, n$, and into the $m$ single-sink fixed-charge transportation problems

$$\begin{aligned}
\min\ & \sum_{j=1}^{n} \Big( (c_{jk} - \lambda_{jk}) x_{jk} + (f_{jk} - \mu_{jk}) y_{jk} \Big) \\
\text{s.t.:}\ & \sum_{j=1}^{n} x_{jk} = d_{jk} \,, \\
& 0 \le x_{jk} \le b_{jk} y_{jk} \,, \; j = 1, \ldots, n \,, \\
& y_{jk} \in \{0,1\} \,, \qquad j = 1, \ldots, n \,,
\end{aligned}$$

for each customer $k = 1, \ldots, m$. Methods for (exactly) solving the SSFCTP are therefore crucial within such a Lagrangean decomposition approach to the FCTP.

The FCTP is itself a special network flow problem with fixed charges. Fixed-charge network flow problems have a large number of applications. They play, e.g., an important role in the field of logistics as well as telecommunications network design, see, e.g., Gendron et al. [10] and Costa [6]. Such optimization problems are also used to model different supply chain planning problems. Ekşioğlu [8] and Ekşioğlu et al. [9], e.g., consider a problem of integrating production, inventory and transportation decisions in a two-stage supply chain. The problem is reformulated as an uncapacitated fixed-charge network design problem and solved heuristically.

## 3 Lower Bounds

A lower bound on the optimal objective function value of the SSFCTP can easily be obtained by solving its linear relaxation. Since $f_j \geq 0$, $y_j = x_j/b_j$ will hold in an optimal solution to the LP relaxation, which thus reduces to the linear program

$$\begin{aligned} Z^{LP} = \min \sum_{j=1}^{n} e_j x_j & \\ \text{s.t.: } \sum_{j=1}^{n} x_j = D\,, & \\ 0 \leq x_j \leq b_j\,, j = 1, \ldots, n\,, & \end{aligned} \tag{5}$$

where $e_j := c_j + f_j/b_j$. Assume that $e_1 \leq e_2 \leq \cdots \leq e_n$, and let $s \in \{1, \ldots, n\}$ be such that

$$\sum_{j=1}^{s-1} b_j < D \quad \text{and} \quad \sum_{j=1}^{s} b_j \geq D\,.$$

An optimal solution to (5) is then given by

$$x_j = \begin{cases} b_j & \text{, for } j = 1, \ldots, s-1 \\ D - \sum_{j=1}^{s-1} b_j & \text{, for } j = s \\ 0 & \text{, for } j = s+1, \ldots, n\,. \end{cases} \tag{6}$$

Optimality of the above LP solution is easily derived by considering the dual of the linear program (5). Denoting the multipliers of the demand constraint (2) and the variable upper bounds (3) by $\sigma$ and $\eta_j$, $j = 1, \ldots, n$, respectively, the dual program is given by

$$\begin{aligned} Z^{LP} = \max \sigma D - \sum_{j=1}^{n} \eta_j b_j & \\ \text{s.t.: } \sigma - \eta_j \leq e_j \quad \text{for } j = 1, \ldots, n\,, & \\ \sigma \in \mathbb{R}\,,\ \eta_j \geq 0 \text{ for } j = 1, \ldots, n\,. & \end{aligned}$$

The solution $\sigma = e_s$ and $\eta_j = \max\{0, e_s - e_j\}$ for $j = 1, \ldots, n$ is a feasible dual solution and leads to the same objective function value as the primal solution (6).

The supplier $s$ plays the same role as the so-called split item (or critical item or break item) does for the BKP, and may therefore be called "split supplier". If $x_s = b_s$ holds in the LP solution, this solution is also optimal for the integer program. Sorting the suppliers can be done in $O(n \log n)$ time. Using similar procedures as those for the knapsack problem (Balas and Zemel [2]), the split supplier may however also be found without sorting in linear time. As in case of the binary min-knapsack problem, the linear programming bound $Z^{LP}$ can however be arbitrarily bad.

The LP bound can be improved by pegging the variable $y_s$ to 0 and 1, respectively, recomputing the resulting linear programming bound and taking the minimum of both values. At the expense of a deteriotation in the bound's quality, the computational effort can substantially be reduced by relaxing the constraints $x_{s+1} \leq b_{s+1}$ and $x_{s-1} \geq 0$ when the variable $y_s$ is temporarily fixed at 0 and 1, respectively. This way, Martello and Toth [16] improved the LP bound for the BKP. In case of the SSFCTP, setting $y_s = 0$ gives

$$L_0 = \sum_{j=1}^{s-1} e_j b_j + e_{s+1}\Big(D - \sum_{j=1}^{s-1} b_j\Big) = Z^{LP} + \Big(e_{s+1} - e_s\Big)\Big(D - \sum_{j=1}^{s-1} b_j\Big)$$

as a lower bound on the objective function value on this branch. A lower bound for the branch $y_s = 1$ is obtained as follows. If $e_{s-1} \leq c_s$, the LP bound under the additional constraint $y_s = 1$ is given by

$$L_1 = \sum_{j=1}^{s-1} e_j b_j + f_s + c_s\Big(D - \sum_{j=1}^{s-1} b_j\Big) = Z^{LP} + (c_s - e_s)(D - \sum_{j=1}^{s} b_j)\,.$$

In the case of $e_{s-1} > c_s$, dualizing the constraint $\sum_{j \neq s} x_j = D - b_s$ with a multiplier of $e_{s-1}$ gives the lower bound

$$\begin{aligned} L_1 &= c_s b_s + f_s + e_{s-1}(D - b_s) \\ &\quad + \min\Big\{\sum_{j \neq s}(e_j - e_{s-1})x_j : 0 \leq x_j \leq b_j \text{ for } j = 1, \ldots, n,\, j \neq s\Big\} \\ &= e_s b_s + e_{s-1}(D - b_s) + \sum_{j=1}^{s-1}(e_j - e_{s-1})b_j = \sum_{j=1}^{s} e_j b_j + e_{s-1}\Big(D - \sum_{j=1}^{s} b_j\Big) \\ &= Z^{LP} + (e_{s-1} - e_s)(D - \sum_{j=1}^{s} b_j) \end{aligned}$$

on the branch $y_s = 1$. Summarizing, one thus obtains

$$L_1 = Z^{LP} + \Big(\max\{c_s, e_{s-1}\} - e_s\Big)\Big(D - \sum_{j=1}^{s} b_j\Big)$$

and

$$L_2 = \min\{L_0, L_1\} \geq Z^{LP} \tag{7}$$

as an improved lower bound on the objective function value $Z$ of the SSFCTP.

Herer et al. [13] propose other lower bounds. Their lower bound $Lb_3$ is also based on the linear relaxation; they do, however, independently minimize the sum of the fixed costs and the sum of the transportation costs. This bound is thus dominated by the LP bound. Their second lower bound $Lb_2$ is based on minimizing the transportation cost in the LP relaxation and adding the sum of the $m^*$ smallest fixed costs, where $m^*$ is the minimum number of suppliers required to meet the sink's demand. Although this bound is not generally dominated by the LP bound, it should normally be much weaker.

## 4 Upper Bounds

A first feasible solution is directly available from the solution, say $x^{LP}$, of the LP relaxation (5) by setting $x = x^{LP}$ and $y_j = 1$ if $x_j^{LP} > 1$ and $y_j = 0$ otherwise. The cost of this solution is

$$Z^G = \sum_{j=1}^{s-1} e_j b_j + f_s + c_s x_s^{LP} = Z^{LP} + f_s + (c_s - e_s)x_s^{LP} = Z^{LP} + f_s\Big(1 - x_s^{LP}/b_s\Big)\,.$$

This heuristic solution approach closely resembles the greedy algorithm for the knapsack problem. As in the case of the minimization knapsack problem (see, e.g., Kellerer et al. [14, p. 413]) the upper bound $Z^G$ may, however, be arbitrarily bad.

After the first $s-1$ flow variables $x_j$ were set to the upper bound $b_j$, the remaining demand $\overline{D} = D - \sum_{j=1}^{s-1} b_j$ is usually smaller than the split supplier's capacity $b_s$ (otherwise the LP solution is also optimal for the integer program). In order to complete the solution, it is therefore reasonable to use the "effective capacity" $\min\{\overline{D}, b_j\}$ for the purposes of linearizing the fixed costs and selecting further suppliers that supply the remaining demand. This gives the following "adaptive" greedy algorithm that was already considered by Herer et al. [13]:

*Step* 1: For $j = 1, \ldots, s-1$ set $x_j = b_j$ and $y_j = 1$. Set $\overline{D} = D - \sum_{j=1}^{s-1} b_j$.

*Step* 2: Find $i$ with $c_i + f_i/\min\{\overline{D}, b_i\} = \min\limits_{j=s,\ldots,n}\Big\{c_j + f_j/\min\{\overline{D}, b_j\} : x_j = 0\Big\}$.

*Step* 3: Set $x_i = \min\{\overline{D}, b_i\}$ and $\overline{D} := \overline{D} - x_i$. If $\overline{D} > 0$ go back to Step 2.

Although the above procedure usually gives upper bounds of reasonable good quality, the procedure's solution can also be arbitrarily bad. Polynomial-time approximation algorithms with a guaranteed worst-case performance can however be obtained by adjusting approximation algorithms for the min-knapsack problem as those propsed by Csirik et al. [7] and Güntzer and Jungnickel [11] to the case of the SSFCTP. A deeper analysis of greedy-type heuristics for the SSFCTP and some approximation results for the SSFCTP will be the subject of an accompanying working paper.

The feasible solutions computed as above can be improved slightly by means of recomputing the flows on the selected arcs $j \in J_1 := \{j = 1, \ldots, n : y_j = 1\}$. This is easily accomplished by sorting the arcs $j \in J_1$ in non-decreasing order of the unit costs $c_j$.

## 5 Problem Reduction

Branch-and-bound methods for the fixed-charge transportation problem usually employ penalties for the purposes of pegging binary variables and improving the linear programming bound (Kennington and Unger [15], Cabot and Erenguc [4], Palekar et al. [19], Bell et al. [3]). Penalties are lower bounds on the change in the objective function value caused by setting a variable to its lower and upper bound, respectively. In the following such penalties are derived for the SSFCTP and used to peg binary variables to zero and one, respectively, as well as to reduce the arc capacities and to improve the lower bound on the flow $x_j$ in the case that $y_j$ is set to one.

## *5.1 Pegging binary variables to zero*

Consider the Lagrangean relaxation of the SSFCTP that is obtained from dualizing the demand constraint (2) with a multiplier value of $e_s = c_s + f_s/b_s$, where $s$ is again the index of the split supplier. Because the arc capacities $b_j$ are assumed to be integer, the constraints $x_j \geq y_j$ are additionally included. This yields the Lagrangean subproblem

$$\begin{aligned} Z^{LR} = e_s D + \min \sum_{j=1}^{n} \Big( (c_j - e_s)x_j + f_j y_j \Big) & \\ \text{s.t.: } y_j \leq x_j \leq b_j y_j \quad & \text{for } j = 1, \ldots, n\,, \\ y_j \in \{0,1\} \qquad & \text{for } j = 1, \ldots, n \end{aligned} \tag{8}$$

or, equivalently,

$$\begin{aligned} Z^{LR} = e_s D + \min \sum_{j=1}^{n} \Big( \max\{0, c_j - e_s\} + \min\{0, c_j - e_s\} b_j + f_j \Big) y_j & \\ \text{s.t.: } y_j \in \{0,1\} \text{ for } j = 1, \ldots, n\,. & \end{aligned}$$

Because $b_j \geq 1$,

$$\max\{0, c_j - e_s\} + \min\{0, c_j - e_s\} b_j + f_j = f_j + \min\{c_j - e_s, (c_j - e_s) b_j\} =: \overline{f}_j$$

results. Moreover, since $e_j := c_j + f_j/b_j \leq e_s$ and thus

$$\overline{f}_j = f_j + c_j b_j - e_s b_j = (e_j - e_s) b_j \leq 0$$

holds for $j \in \{1, \ldots, s\}$, one obtains

$$\begin{aligned} Z^{LR} &= e_s D + \sum_{j=1}^{s} (e_j - e_s) b_j \\ &\quad + \min \Big\{ \sum_{j=s+1}^{n} \overline{f}_j y_j : y_j \in \{0,1\} \text{ for } j = s+1, \ldots, n \Big\} \\ &= Z^{LP} + \min \Big\{ \sum_{j=s+1}^{n} \overline{f}_j y_j : y_j \in \{0,1\} \text{ for } j = s+1, \ldots, n \Big\}. \end{aligned}$$

Setting $y_j = 1$ for $j \in \{s+1, \ldots, n\}$ yields therefore an increase in the objective function value of at least $\overline{f}_j$ and the variable can be pegged to zero if $\overline{f}_j \geq Z^{UB} - Z^{LP}$, where $Z^{UB}$ is the objective function value of the incumbent solution.

## *5.2 Pegging binary variables to one*

Let the flow $x_k$ of a supplier $k \in \{1, \ldots, s-1\}$ with $e_k < e_s$ be limited by $\hat{x}_k < b_k$. A lower bound on the objective function value that results if the constraint $x_k \leq \hat{x}_k$ is included, can be obtained from the linear program

$$\begin{aligned} L(\hat{x}_k) := e_s D + \min \ & \sum_{j=1}^{n} (e_j - e_s) x_j \\ \text{s.t.: } & 0 \leq x_j \leq b_j \text{ for } j = 1, \ldots, n\,, \\ & x_k \leq \hat{x}_k \end{aligned} \tag{9}$$

that results from the LP relaxation (5) if the demand constraint (2) is dualized with a multiplier value of $e_s$. An optimal solution to this linear program is given by $x_j = b_j$ for $j = 1, \ldots, s$ and $j \neq k$, $x_k = \hat{x}_k$, and $x_j = 0$ for $j = s+1, \ldots, n$. Hence

$$L(\hat{x}_k) = Z^{LP} - (b_k - \hat{x}_k)(e_k - e_s)\,,$$

and any feasible solution that improves an incumbent solution of objective function value $Z^{UB}$ must obey the constraint

$$Z^{LP} - (b_k - x_k)(e_k - e_s) \leq Z^{UB} \quad \Rightarrow \quad x_k \geq \left\lceil b_k + \frac{Z^{UB} - Z^{LP}}{e_k - e_s} \right\rceil =: l_k\,,$$

where $\lceil a \rceil$ is the smallest integer larger than $a$ or equal to $a$. Each binary variable $y_j$ with $j \in J_1 := \{j = 1, \ldots, s-1 : l_j \geq 1\}$ can then be pegged to one. The SSFCTP has however at least one optimal solution such that at most one of the flow variables $x_j > 0$ is not at its upper bound $b_j$. Thus if $|J_1| > 1$, the flow on each arc $j \in J_1$ except on the arc $j' = \arg\max\{c_j : j \in J_1\}$ can therefore be fixed to its upper bound $b_j$.

## *5.3 Improving the bounds on the flow variables*

Consider the linear program (9) with the constraint $x_k \leq \hat{x}_k$ replaced by $x_k \geq \tilde{x}_k > 0$ for a supplier $k \in \{s+1, \ldots, n\}$ with $e_k > e_s$. It is then apparent that imposing the constraint $x_k \geq \tilde{x}_k$ increases the LP bound at least to the amount $Z^{LP} + (e_k - e_s)\tilde{x}_k$. Hence

$$x_k \leq \left\lfloor \frac{Z^{UB} - Z^{LP}}{e_k - e_s} \right\rfloor =: u_k$$

must hold in any feasible solution that improves the incument solution of value $Z^{UB}$ and the arc capacity $b_k$ can be reduced to $u_k$ if $u_k < b_k$.

Bounds on the flow $x_k$ of an arc $k \in \{s+1, \ldots, n\}$ may also be obtained from the Lagrangean relaxation (8). If for $k \in \{s+1, \ldots, n\}$ the flow $x_k$ is set to a value of $\tilde{x}_k > 0$, $\tilde{x}_k \leq b_k$, the objective function value of the Lagrangean subproblem (8) increases to

$$Z^{LP} + f_k + (c_k - e_s)\tilde{x}_k\,,$$

and a solution that improves the incument $Z^{UB}$ must satisfy the constraint

$$f_k + (c_k - e_s)x_k \le Z^{UB} - Z^{LP} \, .$$

Thus, in case of $c_k > e_s$ one obtains

$$x_k \le \tilde{u}_k, \text{ where } \tilde{u}_k = \left\lfloor \frac{Z^{UB} - Z^{LP} - f_k}{c_k - e_s} \right\rfloor ,$$

while

$$x_k \ge \left\lceil \frac{Z^{UB} - Z^{LP} - f_k}{c_k - e_s} \right\rceil =: \ell_k$$

results in the case that $c_k < e_s$. If $\ell_k \ge 1$, this bound can be used as a lower bound on the minimum amount of flow on arc $k$ provided that $y_k = 1$. The upper bound $\tilde{u}_k$ can be used instead of $u_k$ to reduce the arc capacity $b_k$ if $\tilde{u}_k < u_k$ and $\tilde{u}_k < b_k$.

## 6 Algorithms

In the sequel, three algorithms for solving the SSFCTP are described. The next section summarizes the dynamic programming method of Alidaee and Kochenberger [1] and shows how the search space might be reduced. Section 6.2 briefly recapitulates the enumeration algorithm of Herer et al. [13], and Section 6.3 adapts an algorithm of Martello and Toth [16] for the BKP to the SSFCTP.

### *6.1 An Improved Dynamic Programming Algorithm*

Let $G_i(S)$ be the minimum cost required for shipping $D - S$ units from suppliers $j = i, \ldots, n$ to the sink node given that suppliers $j = 1, \ldots, i-1$ supply $S \in \{S_i^{\min}, S_i^{\min} + 1, \ldots, S_i^{\max}\}$ units, where $S_i^{\min} = \max\Big\{0, D - \sum_{j=i}^{n} b_j\Big\}$ and $S_i^{\max} = \min\Big\{D, \sum_{j=1}^{i-1} b_j\Big\}$. That means

$$\begin{aligned} G_i(S) = \min \ & \sum_{j=i}^{n} (c_j x_j + f_j y_j) \\ \text{s.t.:} \ & \sum_{j=i}^{n} x_j = D - S \, , \\ & 0 \le x_j \le b_j y_j \, , \quad j = i, \ldots, n \, , \\ & y_j \in \{0, 1\} \, , \qquad j = i, \ldots, n \end{aligned}$$

for $S_i^{\min} \le S \le S_i^{\max}$, $G_i(S) = \infty$ for $S \notin [S_i^{\min}, S_i^{\max}]$, $G_i(D) = 0$, and $G_1(0) = Z$. The function $G_i(S)$ can be computed recursively by means of the recursion

$$G_i(S) = \min\Big\{G_{i+1}(S), \ \min\Big\{f_i + c_i x_i + G_{i+1}(S + x_i) : x_i = 1, \ldots, x_i^{\max}\Big\}\Big\} ,$$

where $x_i^{\max} = \min\{b_i,\ D - S\}$. The complexity of the resulting dynamic programming algorithm is then $O(nD \max_j b_j)$.

Alidaee and Kochenberger [1] now make use of the variable transformation $r := x_i + S$ yielding

$$G_i(S) = \min\Big\{G_{i+1}(S),\ \min\Big\{f_i - c_i S + H_i(r) : r = S+1, \dots, \min\{S + b_i, D\}\Big\}\Big\}\,,$$

where $H_i(r) = c_i r + G_{i+1}(r)$. If the function values $H_i(r)$ and their arguments $r$ are stored in a list *Hlist* $:= \{(r_1, H_i(r_1)), \dots, (r_q, H_i(r_q))\}$ such that $r_{l-1}$ is the largest $r < r_l$ with $H_i(r) < H_i(r_l)$, one obtains

$$G_i(S) = \min\Big\{G_{i+1}(S),\ f_i - c_i S + H_i(r_1)\Big\}\,.$$

The list *Hlist* is computed once for $S = S_i^{\min}$ and updated accordingly when moving from $S$ to $S+1$. The complexity of the resulting dynamic programming method is then reduced to $O(nD)$.

The computational effort required by this dynamic programming approach might be reduced if the range $[S_i^{\min}, S_i^{\max}]$ on which the function $G_i(S)$ must be evaluated for each $i = n, n-1, \dots, 1$ could be made smaller. For this purpose, consider the function

$$Z_i(S) = Z_{i1}(S) + Z_{i2}(S)\,,$$

where

$$\begin{aligned} Z_{i1}(S) = \min\ & \sum_{j=1}^{i-1} e_j x_j \\ \text{s.t.:}\ & \sum_{j=1}^{i-1} x_j = S\,, \\ & 0 \le x_j \le b_j \text{ for } j = 1, \dots, i-1 \end{aligned}$$

and

$$\begin{aligned} Z_{i2}(S) = \min\ & \sum_{j=i}^{n} e_j x_j \\ \text{s.t.:}\ & \sum_{j=i}^{n} x_j = D - S\,, \\ & 0 \le x_j \le b_j \text{ for } j = i, \dots, n\,. \end{aligned}$$

It is again assumed that $e_1 \le e_2 \le \cdots \le e_n$ and that $s$ is the index of the split supplier. The function $Z_i(S)$ is difficult to evaluate. The duals of the linear programs above are given by

$$
\begin{aligned}
Z_{i1}(S) = \max\ & \sigma_1 S - \sum_{j=1}^{i-1} \eta_{1j} b_j \\
\text{s.t.: } & \sigma_1 - \eta_{1j} \le e_j \qquad \text{for } j = 1, \dots, i-1\,, \\
& \sigma_1 \in \mathbb{R}\,,\ \eta_{1j} \ge 0 \quad \text{for } j = 1, \dots, i-1
\end{aligned}
\tag{10}
$$

and

$$
\begin{aligned}
Z_{i2}(S) = \max\ & \sigma_2 (D - S) - \sum_{j=i}^{n} \eta_{2j} b_j \\
\text{s.t.: } & \sigma_2 - \eta_{2j} \le e_j \qquad \text{for } j = i, \dots, n\,, \\
& \sigma_2 \in \mathbb{R}\,,\ \eta_{2j} \ge 0 \quad \text{for } j = i, \dots, n\,.
\end{aligned}
\tag{11}
$$

First assume that $i \ge s+1$ and $e_i > e_s$. A feasible solution to the dual program (10) is then given by $\sigma_1 = e_s$, $\eta_{1j} = e_s - e_j$ for $j = 1, \dots, s$, and $\eta_{1j} = 0$ for $j = s+1, \dots, i-1$. Hence

$$
\begin{aligned}
Z_{i1}(S) \ge e_s S - \sum_{j=1}^{s} (e_s - e_j) b_j &= e_s D - \sum_{j=1}^{s} (e_s - e_j) b_j - e_s (D - S) \\
&= Z^{LP} - e_s (D - S)\,.
\end{aligned}
$$

A feasible solution to (11) is given by $\sigma_2 = e_i$ and $\eta_{2j} = 0$ for $j = i, \dots, n$. Therefore, $Z_{i2}(S) \ge e_i (D - S)$ and in total

$$
Z_i(S) \ge Z^{LP} + (e_i - e_s)(D - S)\,.
$$

Thus, if $Z^{UB}$ denotes again the objective function value of an incumbent solution, the following lower bound is obtained on the amount $S$ that has to be supplied by nodes $j = 1, \dots, i-1$ in an improved solution:

$$
Z^{LP} + (e_i - e_s)(D - S) \le Z^{UB} \quad \Rightarrow \quad S \ge \left\lceil D - \frac{Z^{UB} - Z^{LP}}{e_i - e_s} \right\rceil . \tag{12}
$$

In the case of $i \le s$, the solution $\sigma_1 = e_{i-1}$ and $\eta_{1j} = e_{i-1} - e_j$ for $j = 1, \dots, i-1$ is feasible for (10), while a feasible solution to (11) is given by $\sigma_2 = e_s$, $\eta_{2j} = e_s - e_j$ for $j = i, \dots, s$, and $\eta_{2j} = 0$ for $j = s+1, \dots, n$. Evaluating the objective functions of the dual programs (10) and (11) at these solutions gives

$$
\begin{aligned}
Z_i(S) &\ge e_{i-1} S - \sum_{j=1}^{i-1} (e_{i-1} - e_j) b_j + e_s (D - S) - \sum_{j=i}^{s} (e_s - e_j) b_j \\
&= (e_{i-1} - e_s) S - \sum_{j=1}^{i-1} \Big( b_j (e_{i-1} - e_j) - b_j (e_s - e_j) \Big) + D e_s - \sum_{j=1}^{s} (e_s - e_j) b_j \\
&= (e_{i-1} - e_s) S - \sum_{j=1}^{i-1} (e_{i-1} - e_s) b_j + Z^{LP} \\
&= Z^{LP} + (e_s - e_{i-1}) \Big( \sum_{j=1}^{i-1} b_j - S \Big)\,.
\end{aligned}
$$

Thus if $e_s > e_{i-1}$, the condition $Z_i(S) \le Z^{UB}$ implies

$$S \ge \left\lceil \sum_{j=1}^{i-1} b_j - \frac{Z^{UB} - Z^{LP}}{e_s - e_{i-1}} \right\rceil . \tag{13}$$

When the lower bound $S_i^{\min}$ at stage $i$ is sharpened using expressions (12) and (13), respectively, one has to additionally take into account that suppliers $j = 1, \dots, i-1$ must at least supply $S_{i+1}^{\min} - b_i$ units.

## *6.2 The Implicit Enumeration Algorithm of Herer et al.*

Herer et al. [13] propose an implicit enumeration algorithm for solving the SSFCTP. The suppliers are reordered such that $c_1 \le c_2 \le \cdots \le c_n$. The method then enumerates all feasible solutions $y \in \{0,1\}^n$ in lexicographically descending order. Due to the ordering of the suppliers, the corresponding solution to the flow variables is easily determined by setting $x_j = b_j$ if $y_j = 1$ as long as $\sum_j x_j < D$, and to set the last $x_j$ with $y_j = 1$ equal to the remaining demand. In order to avoid that all feasible solutions have to be enumerated explicitly, Herer et al. [13] use lower bounds as well as domination rules. A backtracking step consists in setting the rightmost $y_j = 1$ to zero; the bounds $Lb_2$ and $Lb_3$ mentioned in Section 3 are then computed in order to check if this branch with $y_j = 0$ can be pruned. The domination rules proposed by Herer et al. [13] can be stated as follows.

**Definition 1.** A supplier $i$ is said to *strictly dominate* a supplier $j \ne i$ if

$$b_j \le b_i \quad \text{and} \quad f_i + c_i x_j \le f_j + c_j x_j \text{ for } x_j = 1, \dots, b_j .$$

Furthermore, a supplier $i$ is said to *weakly dominate* a supplier $j \ne i$ if

$$(b_i \ge b_j \text{ and } f_i + b_j c_i \le f_j + b_j c_j) \quad \text{or} \quad (b_i < b_j \text{ and } f_i + b_i c_i \le b_i c_j) .$$

If $i$ weakly dominates $j \ne i$, then there is an optimal solution to the SSFCTP such that $x_i = 0$ implies $x_j < b_j$ (see [13, Property 3] for a proof). If supplier $i$ strictly dominates $j \ne i$, then obviously $x_i = 0$ implies $x_j = 0$ in an optimal solution to the SSFCTP (see also Property 4 in [13]). Furthermore, if $c_i \le c_j$, then $i$ strictly dominates $j$ if and only if $f_i - f_j \le c_j - c_i$. In the case of $c_i > c_j$, $i$ strictly dominates $j$ if and only if $f_j \ge f_i + (c_i - c_j)b_j$.

Herer et al. [13] use the above notion of dominance in the following way.

(1) Let $k$ be the index of the last $y$-variable explicitly set to one. If $k$ dominates a supplier $i < k$ such that $x_i > 0$, then $x_k$ must not be set to zero.
(2) If on backtracking $x_k$ is set to zero, then $x_j = 0$ follows for all $j > k$ that are strictly dominated by $k$ and $x_j < b_j$ follows for all $j > k$ that are weakly dominated by $k$.

(3) Let $F$ be the set of all free variables $j > k$ that strictly dominate a fixed variable $i < k$ with $x_i > 0$. Then $x_j > 0 \; \forall j \in F$ follows, and one may thus set: $x_j = b_j \; \forall j \in F \setminus \{j'\}$ where $j' = \arg\max\{c_j : j \in F\}$.

## *6.3 An Alternative Implicit Enumeration Algorithm*

This method uses the lower bound (7) within an enumeration algorithm for the SSFCTP similar as Martello and Toth [16] did for the BKP. To this end, the suppliers are first sorted according to non-decreasing linearized costs per unit, that is $e_1 \leq e_2 \leq \cdots \leq e_n$. Based on this ordering of the suppliers, feasible solutions are then enumerated in lexicograhically decreasing order. A backtracking step sets the last variable $y_j$ that was explicitly set to one in a previous forward move to zero. The bound (7) is then computed in order to check if this branch can be pruned. Furthermore, to facilitate the computation of the flows $x_j$, a second ordering of the suppliers according to non-decreasing unit costs $c_j$ is kept. The overall algorithm can be described as follows:

*Step* 0: Sort the suppliers such that $e_1 \leq \cdots \leq e_n$.

*Step* 1: Solve the LP relaxation (5). Let $x^{LP}$ be the resulting solution and $Z^{LP}$ denote the LP bound. If $x_j^{LP} \in \{0, b_j\}$ for $j = 1, \ldots, n$, then $x^{LP}$ is optimal for the SSFCTP. Otherwise compute the bound $L_2 \geq Z^{LP}$ in (7). Set $L^* = L_2$.

*Step* 2: Apply the "adaptive" greedy algorithm of Sect. 4 in order to obtain a second feasible solution to the SSFCTP. Let $Z^{UB}$ denote the objective function value of the best feasible solution $x^*$ found this way. If $L_2 \geq Z^{UB}$ then terminate.

*Step* 3: Optionally try to reduce the problem as shown in Sect. 5. Determine a second ordering of the $n'$ remaining free variables according to non-decreasing unit costs $c_j$. Set $S = D$.

*Step* 4: If $x^{LP} = 0$ then terminate, $x^*$ with objective function value $Z^{UB}$ is an optimal solution. Otherwise, set $k = \max\{j = 1, \ldots, n' : x_j^{LP} > 0\}$, $S := S - x_k^{LP}$, $x_k^{LP} = 0$.

*Step* 5: Let $F_k^0$ and $F_k^<$ be the set of all suppliers $j > k$ that are strictly and weakly dominated by supplier $k$, respectively. Set $s = k$ and $S' = S$. As long as $S < D$ do the following:

- Let $s := s + 1$ and set $b'_s = 0$ if $s \in F_k^0$, $b'_s = b_s - 1$ if $s \in F_k^<$, and $b'_s = b_s$ otherwise.
- Set $x_s^{LP} = \min\{b'_s, D - S\}$, $Z^{LP} := Z^{LP} + e_s x_s^{LP}$, and $S := S + x_s^{LP}$.

*Step* 6: Let $j_0 = \min\{j > s : j \notin F_k^0\}$ and $j_1 = \max\{j < s : x_j^{LP} > 0\}$. Set $L_0 = \infty$ if $j_0 > n'$ and $L_0 = Z^{LP} + (e_{j_0} - e_s)x_s^{LP}$ otherwise. Set $L_1 = Z^{LP}$ if $j_1 = 0$ and $L_1 = Z^{LP} + (e_s - \max\{c_s, e_{j_1}\})(b_s - x_s^{LP})$ otherwise. If $L_2 := \min\{L_0, L_1\} \geq Z^{UB}$, set

$S = S'$ as well as $x_j^{LP} = 0$ for $j = k+1, \ldots, n'$ and goto Step 4. Otherwise continue with Step 7.

*Step* 7: Use the ordering of the suppliers according to non-decreasing unit costs $c_j$ to determine the best feasible solution $x^{cur}$ with $x_j^{cur} = 0$ if $x_j^{LP} = 0$. Let $Z^{cur}$ be the objective function value of this solution. If $Z^{cur} < Z^{UB}$ then set $Z^{UB} := Z^{cur}$ and $x^* = x^{cur}$. If $L^* \geq Z^{UB}$ terminate, $x^*$ with objective function value $Z^{UB}$ is an optimal solution. If $L_2 \geq Z^{UB}$ then set $S = S'$ and $x_j^{LP} = 0$ for $j = k+1, \ldots, n$. Return to Step 4.

The above algorithm does not make use of the third dominance test mentioned in Sect. 6.2. If supplier $j$ strictly dominates a supplier $i < j$ then, by definition, $b_j \geq b_i$ and $f_j + c_j b_i \leq f_i + c_i b_i$. Due to the ordering of the suppliers according to non-decreasing values of the $e_j$, we have $e_i \leq e_j$. Hence

$$\frac{f_j}{b_i} + c_j \leq \frac{f_i}{b_i} + c_i = e_i \leq e_j = \frac{f_j}{b_j} + c_j \,.$$

The above inequality in turn implies that $b_i \geq b_j$, and thus $b_i = b_j$. With $b_i = b_j$ one further obtains that $f_j/b_i + c_j = f_j/b_j + c_j = e_j \leq e_i$ and thereby $e_j = e_i$. A supplier $j$ can thus only strictly dominate a supplier $i < j$, if $e_i = e_j$ and $b_i = b_j$. Since this barely occurs, the third dominance rule will usually not be applicable.

## 7 Computational Results

The algorithms presented in the preceding section were coded in C, compiled with the Gnu C compiler (version 2.95.2) using the compiler option $-$O3, and run on a 750 MHz Pentium III PC equipped with 512 MB RAM and operated with a Linux system, kernel version 2.2.16. Sect. 7.1 describes how the test problems were generated, and Sect. 7.2 reports on the computional results.

### *7.1 Test Problems*

Two different groups of test problem instances were generated.

The first group of test problems consists of 60 problem instances for each problem size of $n$ = 500, 1000, 5000, and 10000 suppliers. According to the problem generation method proposed by Herer et al. [13], the problem structure is determined by two parameters.

- The $b$-ratio $B_r = 100 \cdot D / \sum_j b_j$ estimates the percentage number of suppliers that supply a positive amount in an optimal solution. The computational experiments are based on the four $b$-ratio values $B_r = 5\,\%$, $10\,\%$, $25\,\%$, and $50\,\%$. Herer et al.

[13] use in fact the absolute number $B'_r = B_r \cdot n/100$; utilizing the same expected number of employed suppliers for problems that largely differ in size is however not meaningful.

- The $F$-ratio $F_r = \overline{f}/(\overline{cb})$ is the ratio between the average fixed cost $\overline{f} = \frac{1}{n}\sum_j f_j$ and the average transportation cost $\overline{cb} = (\frac{1}{n}\sum_j c_j)(\frac{1}{n}\sum_j b_j)$ per supplier. Three different $F$-ratio values were used, 0.3, 0.6 and 1.

The sink's demand was set to $D = 100000$ and the other problem data were generated randomly such that $f_j \in U[75000, 125000]$, $c_j \in U[8, 12]$, and $b_j \in U[7500, 12500]$, where $U[a_1, a_2]$ denotes a uniformly distributed random number between $a_1$ and $a_2$. The fixed costs $f_j$ and the suppliers' capacities $b_j$ were then scaled to meet the desired $b$-ratio and $F$-ratio. Furthermore, the capacities as well as the fixed costs were rounded to the nearest integer. 5 problem instances were then generated for each problem size and combination of $b$-ratio and $F$-ratio.

In the second set of 20 test problems, 5 instances were generated for each of the problem sizes of $n = 500$, 1000, 5000 and 10000 suppliers. In this case, the suppliers' capacities $b_j$ are integer random numbers from $U[10, 100]$. The sink's demand was set to $D = 0.5\sum_j b_j$ and rounded to the nearest integer. The cost data are then generated such that a positive correlation between the capacity $b_j$ and a supplier $j$'s total cost $C_j := f_j + c_j b_j$ is achieved. To this end, $C_j$ is computed as $C_j = b_j + \beta_j$, where $\beta_j \in [0, b_j]$ is a random integer. The cost $C_j$ is then distributed between fixed and unit cost by setting $f_j = \phi C_j$ and $c_j = (1 - \phi)C_j/b_j$ with $\phi \in U[0.75, 1)$. The linearized costs $e_j$ are in this case given by $e_j = 1 + \beta_j/b_j$ and lie between 1 and 2 with an expected value of 1.5. These instances can be expected to be more difficult due to the correlation between capacities and costs and the similarities in the linearized costs.

## *7.2 Computational Comparison of the Proposed Methods*

For reasons of comparison, CPLEX's MIP solver (version 7.1) was additionally employed to solve the test problems. To this end, the LP relaxation (5) was first solved, and then the upper bounds of Sect. 4 were computed in order to reduce the problem by means of the reduction tests of Sect 5. The reduced problem and the upper bound was then passed to CPLEX's routine *CPXmipopt()* using default values for the MIP solver's optional parameters, except of the relative optimality tolerance that was set to $10^{-6}$. The same optimality tolerance was also used in case of the other two enumerative algorithms. Furthermore, the computation time was limited to 2100 seconds per problem instance.

Table 1 compares the computation times in seconds obtained with the dynamic programming algorithm, the implicit enumeration approach of Sect. 6.3 and CPLEX on the first group of test problems. Before one of the algorithms was invoked, the test problem instance was first reduced by means of the described reduction tests. In case of the dynamic programming method, the bounds on the amount $S_i^{\min}$ that

Table 1
CPU times of DP, enumeration algorithm and CPLEX for group 1 test problems

| | | dyn. prog. | | | alternative EA | | | CPLEX | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $F_r$ | | | $F_r$ | | | $F_r$ | |
| $n$ | $B_r$ | 0.30 | 0.60 | 1.00 | 0.30 | 0.60 | 1.00 | 0.30 | 0.60 | 1.00 |
| 500 | 5 | 3.94 | 7.31 | 7.47 | 0.01 | 0.15 | 0.32 | 0.39 | 3.73 | 276.18 |
| | 10 | 1.98 | 4.21 | 4.53 | 0.08 | 0.03 | 0.14 | 1.36 | 2.11 | 5.81 |
| | 25 | 0.79 | 1.05 | 1.93 | 0.05 | 0.07 | 0.14 | 2.53 | 1.60 | 6.58 |
| | 50 | 0.28 | 1.08 | 1.01 | 0.01 | 0.20 | 0.13 | 1.06 | 11.85 | 13.85 |
| 1000 | 5 | 3.80 | 7.85 | 6.41 | 0.12 | 0.04 | 0.54 | 8.80 | 4.38 | 32.09 |
| | 10 | 0.65 | 2.06 | 1.99 | 0.01 | 0.14 | 0.04 | 1.43 | 4.80 | 1.19 |
| | 25 | 0.28 | 1.64 | 1.45 | 0.08 | 0.44 | 0.17 | 8.19 | 19.14 | 23.86 |
| | 50 | 0.44 | 0.93 | 0.88 | 0.11 | 0.10 | 0.25 | 8.31 | 8.72 | 92.74 |
| 5000 | 5 | 1.29 | 2.92 | 6.21 | 0.18 | 0.75 | 40.17 | 21.78 | 215.71 | 577.29 |
| | 10 | 1.27 | 1.46 | 2.99 | 0.54 | 1.07 | 20.25 | 84.01 | 204.51 | 29.10 $^{(2)}$ |
| | 25 | 0.31 | 0.83 | 0.93 | 0.19 | 0.39 | 0.24 | 41.72 | 212.66 | 188.93 |
| | 50 | 0.39 | 0.32 | 0.81 | 0.12 | 0.12 | 0.49 | 9.15 | 53.61 | 51.97 |
| 10000 | 5 | 1.92 | 1.82 | 4.49 | 0.73 | 6.70 | 133.25 | 132.57 | 633.52 $^{(1)}$ | 499.27 $^{(3)}$ |
| | 10 | 1.41 | 2.07 | 2.69 | 5.70 | 12.21 | 16.42 | 132.22 | 834.66 | 153.62 $^{(1)}$ |
| | 25 | 0.81 | 1.46 | 1.22 | 2.05 | 0.79 | 0.88 | 282.31 | 36.72 | 10.85 $^{(1)}$ |
| | 50 | 0.36 | 0.93 | 0.55 | 0.13 | 1.31 | 0.61 | 12.24 | 53.36 | 21.87 $^{(1)}$ |

has at least to be supplied by the first $i-1$ suppliers were also determined in order to reduce the search space. The table shows the CPU times in dependence of the problem size $n$, the $b$-ratio $B_r$ and the $F$-ratio $F_r$. The computation times are averaged over the 5 instances per problem size and combination of $b$-ratio and $F$-ratio that could be solved within the time limit of 2100 seconds. The small superscripted number in brackets shows how many of the 5 instances were not terminated within the time limit. The dynamic programming algorithm required 517 seconds in total to solve all the 240 test problem instances, while the enumerative algorithm of Sect. 6.3 required 1243 seconds. The CPLEX routine was not able to solve every single instance within the time limit of 2100 seconds; in case of 9 instances the procedure stopped early due to the time limit; the total CPU time required for solving all other 231 instances amounted to 22796 seconds. The first two solution methods thus significantly outperform the CPLEX procedure. As already observed by Herer et al. [13], the test problems tend to get more difficult when the $F$-ratio approaches 1. In case, e.g., of the dynamic programming method, the computation time doubled on average when the $F$-ratio raises from 0.3 to 0.6; and an additional increase of $F_r$ from 0.6 to 1 further raised the running time by 20% on average. An even larger average increase in computation time could be observed in case of the enumeration algorithm. The impact of the $b$-ratio on the running time is different. As could be expected, the enumeration algorithm tend to consume more running time if about 50% of all suppliers need to supply a positive amount. The running time of the dynamic programming algorithm however decreases if $B_r$ increases. The computational effort required by this method strongly depends on the average supplier capacity, and this figure declines with increasing $B_r$. Because the sink's demand is the same for all problem instances, this is also true if the number $n$ of suppliers gets larger. Thus, the DP method required in tendency less time to solve the larger instances of

Table 2
CPU times of the algorithm of Herer et al. and CPLEX (without reduction)

| | | EA of Herer et al. | | | CPLEX without reduction | | |
|---|---|---|---|---|---|---|---|
| | | $F_r$ | | | $F_r$ | | |
| $n$ | $B_r$ | 0.30 | 0.60 | 1.00 | 0.30 | 0.60 | 1.00 |
| 500 | 5 | 0.22 | 8.59 | 4.97 | 3.53 | 268.02 | 420.33 |
| | 10 | 4.40 | 11.81 | 61.15 | 6.79 | 5.15 | 5.88 |
| | 25 | 19.70 | 134.67 | 226.17 | 20.42 | 4.37 | 21.05 |
| | 50 | 20.43 | 521.95 | 558.54 | 7.04 | 10.74 | 18.23 |
| 1000 | 5 | 17.32 | 395.62 | 472.75 | 31.20 | 47.19 | 134.10 |
| | 10 | 6.62 | 82.35 | 360.39 | 31.58 | 58.44 | 19.27 |
| | 25 | 59.67 | 1291.47 $^{(2)}$ | 824.72 $^{(1)}$ | 102.97 | 49.93 | 47.62 |
| | 50 | 386.67 $^{(1)}$ | 796.47 $^{(3)}$ | 9.83 $^{(3)}$ | 48.58 | 23.99 | 126.49 |

Table 1 than to solve the smaller ones.

The enumeration algorithm of Herer et al. [13] preceded by the problem reduction method as well as CPLEX's MIP solver without the use of such a reduction test were also applied to solve the test problems. Table 2 shows the results. Due to the large running times, these two procedures were however not applied to test problems of size larger than $n = 1000$. The enumeration algorithm of Herer et al. did not succeed to solve 10 of the instances of size $n = 1000$ within the time limit of 2100 seconds; the time required to solve all other 110 instances of size $n \leq 1000$ amounted to 25169 seconds in total, which is about 20 times as much as the enumeration procedure of Sect. 6.3 required to solve all 240 instances to optimality. Without a preceding application of the reduction tests, it took 7565 seconds in total to solve the instances of size $n \leq 1000$ to optimality by means of CPLEX's MIP solver, in contrast to 2703 seconds if the problem reduction tests were applied.

Table 3 compares the running times of the dynamic programming algorithm, the alternative enumeration method and CPLEX (with application of the reduction tests) on the 20 single instances of the second group of test problems. Also, on these test problem instances, the dynamic programming method as well as the enumerative algorithm both significantly outperformed CPLEX; and dynamic programming did also better than the implicit enumeration approach, though the latter was faster than dynamic programming on the smallest test problems of size $n = 500$.

In order to assess the value of the reduction tests of Sect. 5 and the bounds discussed in Sect. 6.1, the enumerative algorithm of Sect. 6.3 and the dynamic programming algorithm were applied with and without the use of these measures to solve the test problems of the first group. Table 4 compares the running times (summed over the 60 problem instances for each problem size) of the branch-and-bound method depending on whether the problem reduction tests were applied or not. The table shows that a preceding application of the reduction tests saved on average between 46% to 70% of the method's computation time for problems of size $n = 500$ and $n = 10000$, respectively. If the dynamic programming algorithm was applied to the group 1 test problems in a pure form, that is without a preceding problem reduction and computation of the bounds discussed in Sect. 6.1, the computation time did only depend on

Table 3
CPU times of DP, enumeration algorithm and CPLEX for group 2 test problems

| $n$ | instance | dyn. prog. | alternative EA | CPLEX |
|---|---|---|---|---|
| 500 | 1 | 0.04 | 0.01 | 6.06 |
| | 2 | 0.23 | 0.07 | 4.84 |
| | 3 | 0.20 | 0.11 | 9.84 |
| | 4 | 0.08 | 0.01 | 0.08 |
| | 5 | 0.17 | 0.04 | 8.96 |
| 1000 | 1 | 0.05 | 0.10 | 30.20 |
| | 2 | 0.04 | 0.50 | 34.42 |
| | 3 | 0.47 | 0.24 | 87.18 |
| | 4 | 0.05 | 0.06 | 6.91 |
| | 5 | 0.04 | 0.11 | 4.35 |
| 5000 | 1 | 0.70 | 1.64 | $*$ |
| | 2 | 0.40 | 98.66 | $*$ |
| | 3 | 4.95 | $*$ | $*$ |
| | 4 | 0.29 | 0.02 | 1060.74 |
| | 5 | 0.18 | 11.44 | $*$ |
| 10000 | 1 | 2.62 | 9.03 | $*$ |
| | 2 | 1.14 | $*$ | $*$ |
| | 3 | 2.86 | 3.71 | 299.58 |
| | 4 | 3.53 | 3.95 | 665.81 |
| | 5 | 3.67 | 0.68 | $*$ |

* Indicates that the time limit of 2100 seconds was exceeded.

Table 4
CPU time of the enumerative method of Sect. 6.3 with and without reduction test

| problem size $n$ | 500 | 1000 | 5000 | 10000 | sum |
|---|---|---|---|---|---|
| CPU time with reduction test | 6.59 | 10.18 | 322.55 | 903.99 | 1243.31 |
| CPU time without reduction test | 12.10 | 22.12 | 768.15 | 3030.02 | 3832.39 |

the $b$-ratio and the problem size $n$. The method required then 3355 seconds to solve all 60 problem instances of size $n = 500$ and 5832 seconds to solve the 60 instances of size $n = 1000$. If the reduction test and additional bounds were exploited, it took just 320 seconds to solve all these 120 test problems. Larger test problems could not be solved by means of the pure method, since the memory requirements exceeded then the available memory. Even if the reduction tests were carried out and just the computation of the bounds on the amounts $S_i^{\min}$ were skipped, a number of the larger problem instances could not be solved due to memory limitations. Table 5 shows the average running time per problem instance of the dynamic programming algorithm depending on whether the bounds on $S_i^{\min}$ were computed or not. This time the numbers in brackets give the number of problem instances that could not be solved due to insufficient memory. The table shows that the bounds on the amounts $S_i^{\min}$ were very effective and helped to considerably reduce the running time as well as the memory requirements of the dynamic programming algorithm.

Table 5
Average running times of DP with and without use of the bounds on $S_i^{\min}$

| problem size $n$ | 500 | 1000 | 5000 | 10000 |
|---|---|---|---|---|
| with bounds on $S_i^{\min}$ | 2.97 | 2.36 | 1.64 | 1.64 |
| without bounds on $S_i^{\min}$ | 20.22 | 32.81 | 63.70 $^{(17)}$ | 89.01 $^{(37)}$ |

## 8 Conclusions

In this paper, a number of applications of the SSFCTP were described and different algorithms for solving the problem were compared. A dynamic programming approach from the literature was considerably improved by means of a preceding problem reduction test and the incorporation of lower bounds on the quantities to be supplied by the first $i-1$ suppliers. Furthermore, an enumerative algorithm was introduced that is based on similar well-known approaches for the binary knapsack problem. Both methods were found to be quite effective in solving medium-sized and larger instances and outperformed other algorithms for the SSFCTP. Especially the dynamic programming method peformed well as long as the average supplier capacity is not too large. The enumerative algorithm was competetive on smaller instances and in the case of larger supplier capacities. Both solution methods might be suitable for repeatedly solving smaller and medium-sized instances of the SSFCTP, e.g., within Lagrangean relaxation and column generation procedures for fixed-charge transportation and network flow problems. In order to effectively solve very large-scale instances of the SSFCTP, advanced dynamic programming concepts similar to those proposed by Pisinger [20] and Martello et al. [18] for the BKP should be adapted to the case of the SSFCTP.

## References

[1] Alidaee, B. and Kochenberger, G. A. (2005). A note on a simple dynamic programming approach to the single-sink, fixed-charge transportation problem. *Transportation Science*, **39**, 140–143.

[2] Balas, E. and Zemel, E. (1980). An algorithm for large zero-one Knapsack problems. *Operations Research*, **28**, 1130–1154.

[3] Bell, G. J., Lamar, B. W., and Wallace, C. A. (1999). Capacity improvement, penalties, and the fixed charge transportation problem. *Naval Research Logistics*, **46**, 341–355.

[4] Cabot, A. V. and Erenguc, S. S. (1986). Improved penalties for fixed cost linear programs using Lagrangean relaxation. *Management Science*, **32**, 856–869.

[5] Chauhan, S. S. and Proth, J.-M. (2003). The concave cost supply problem. *European Journal of Operational Research*, **148**, 374–383.

[6] Costa, A. M. (2005). A survey on benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, **32**, 1429–1450.

[7] Csirik, J., Frenk, J. B. G., Labbé, M., and Zhang, S. (1991). Heuristics for the 0-1 min-knapsack problem. *Acta Cybernetica*, **10**, 15–20.

[8] Ekşioğlu, S. D. (2002). *Optimizing Integrated Production, Inventory and Distribution Problems in Supply Chains.* Ph.D. thesis, Graduate School of the University of Florida, University of Florida.

[9] Ekşioğlu, S. D., Romeijn, H. E., and Pardalos, P. M. (2006). Cross-facility management of production and transportation planning problem. *Computers & Operations Research*, **33**, 3231–3251.

[10] Gendron, B., Crainic, T. G., and Frangioni, A. (1999). Multicommodity capacitated network design. In B. Sansò and P. Soriano, editors, *Telecommunications Network Planning*, chapter 1, pages 1–19. Kluwer Academic Publishers Group, London, Dordrecht, Boston.

[11] Güntzer, M. M. and Jungnickel, D. (2000). Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. *Operations Research Letters*, **26**, 55–66.

[12] Haberl, J. (1991). Exact algorithm for solving a special fixed-charge linear programming problem. *Journal of Optimization Theory and Applications*, **69**, 489–529.

[13] Herer, Y. T., Rosenblatt, M. J., and Hefter, I. (1996). Fast algorithms for single-sink fixed charge transportation problems with applications to manufacturing and transportation. *Transportation Science*, **30**, 276–290.

[14] Kellerer, H., Pferschy, U., and Pisinger, D. (2004). *Knapsack Problems.* Springer-Verlag, Berlin, Heidelberg, New York.

[15] Kennington, J. L. and Unger, E. (1976). A new branch-and-bound algorithm for the fixed charge transportation problems. *Management Science*, **22**, 1116–1126.

[16] Martello, S. and Toth, P. (1977). An upper bound for the zero-one Knapsack problem and a branch and bound algorithm. *European Journal of Operational Research*, **1**, 169–175.

[17] Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore.

[18] Martello, S., Pisinger, D., and Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 Knapsack problem. *Management Science*, **45**, 414–424.

[19] Palekar, U. S., Karwan, M. K., and Zionts, S. (1990). A branch-and-bound method for the fixed charge transportation problem. *Management Science*, **36**, 1092–1105.

[20] Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, **45**, 758–767.