An improved cut–and–solve algorithm for the single–source capacitated facility location problem*

Sune Lauth Gadegaard† and Lars Relund Nielsen

Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark.

Andreas Klose

Department of Mathematics, Aarhus University, Ny Munkegade 118, DK-8000 Aarhus C, Denmark.

January, 2017

Abstract: In this paper we present an improved cut–and–solve algorithm for the single–source capacitated facility location problem. The algorithm consists of three phases. The first phase strengthens the integer program by a cutting plane algorithm to obtain a tight lower bound. The second phase uses a two level local branching heuristic to find an upper bound and, if optimality has not yet been established, the third phase uses the cut-and-solve framework to close the optimality gap. Extensive computational results are reported, showing that the proposed algorithm runs 10 to 80 times faster on average compared to state–of–the–art problem specific algorithms.
Keywords: Facility location; capacitated facility location; single–sourcing; cutting planes; local branching; cut–and–solve.

1 Introduction

The *single–source capacitated facility location problem* (*SSCFLP*) is the problem of opening a set of facilities and assigning each customer to exactly one open facility while respecting the capacity of each facility. An optimal solution to the SSCFLP is an allocation which minimizes fixed opening and allocation costs. The difference between the SSCFLP and the more widely studied *capacitated facility location problem* (*CFLP*) is that each customer must be serviced by only one facility. The SSCFLP is a significant problem in the area of location science, as it exhibits many of the challenging aspects and features of general location problems. Furthermore, the SSCFLP often arises as a subproblem in more complex

*Preprint of S.L. Gadegaard, A. Klose and L.R. Nielsen, *An improved cut–and–solve algorithm for the single–source capacitated facility location problem* in EURO Journal on Computational Optimization 2017, doi:10.1007/s13675-017-0084-4

†Corresponding author, email: sgadegaard@econ.au.dk.

problems such as hierarchical location problems with capacities and capacitated facility problems with piecewise linear costs.

It is well known that the SSCFLP can be formulated as an *integer linear program* (*ILP*) with a set of demand constraints and a set of capacity constraints. Since the SSCFLP is a strongly $\mathcal{NP}$-hard problem (the SSCFLP polynomially reduces to the node cover problem, which was one of the original 21 $\mathcal{NP}$–complete problems listed in Karp (1972)), most research has been focused on heuristic solution approaches.

Since the *linear programming* (*LP*) lower bound of the SSCFLP stated above is known to be weak, most of the existing literature focuses on a Lagrangean relaxation of one or more sets of the constraints. These Lagrangean based heuristics primarily differ in the way constraints are relaxed and in the way a primal solution is obtained from the Lagrangean subproblem. Klincewicz and Luss (1986) relax the capacity constraints in a Lagrangean manner where the Lagrangean subproblem becomes an uncapacitated facility location problem. A heuristic primal solution is obtained by an add heuristic and a refinement heuristic that improves primal feasible solutions to the Lagrangean subproblem. If the demand constraints are relaxed in a Lagrangean manner instead, Bitran, Chandru, Sempolinski, and Shapiro (1981) show that the Lagrangean subproblem decomposes into an independent knapsack problem for each facility site. This is utilized by Barceló and Casanovas (1984), Pirkul (1987), Sridharan (1993), and Chen and Ting (2008). Barceló, Fernández, and Jörnsten (1991) propose a Lagrangean decomposition approach that separates the demand constraints from all other constraints. The Lagrangean subproblem thus decomposes into two subproblems; one being identical to the one obtained when relaxing the demand constraints and one being a simple semi-assignment problem. Taking a different approach Barceló, Hallefjord, Fernández, and Jörnsten (1990) suggest, to relax the capacity constraints, including a total demand constraint (stating that the open facilities' capacity needs to cover the total demand). Other types of heuristics have been developed as well. Some of the more recent approaches count a repeated matching heuristic proposed by Rönnqvist, Tragantalerngsak, and Holt (1999), a scatter search strategy developed by Contreras and Díaz (2008), and a multi–exchange heuristic presented in Ahuja, Orlin, Pallottino, Scaparra, and Scutellà (2004).

Less research has been devoted to exact solution methods primarily due to the large number of integer variables. Neebe and Rao (1983) reformulate the SSCFLP as a set partitioning problem and column generation is used to solve the resulting model. Holmberg, Rönnqvist, and Yuan (1999) relax the demand constraints in a Lagrangean manner in order to obtain lower bounds. Upper bounds are generated using a repeated matching heuristic. These bounds are used in a branch–and–bound algorithm to find a proven optimal solution to the SSCFLP. A branch–and–price algorithm is developed by Díaz and Fernández (2002). Ceselli and Righini (2005) consider the capacitated p-median problem, which is closely related to the SSCFLP; instead of including fixed facility costs, the number of open facilities may not exceed a number p. They also propose a branch–and–price algorithm based on a set–covering like formulation of the problem. The lower bound obtained from the LP-relaxation of the reformulated problem is easily shown to be the same as the one obtainable from the Lagrangean relaxation of the demand constraints.

Over the years, various methods have been successfully used to strengthen the lower bound on general ILPs, e.g. knapsack separation using lifted cover inequalities (Balas and Zemel (1978) and Kaparis and Letchford (2010)), weight inequalities (Weismantel (1997)) and exact separation (Boyd (1993)). Avella, Boccia, and Salerno (2011) proposed a reformulation of the SSCFLP based on dicut inequalities where exact knapsack separation is used to strengthen the formulation. Furthermore, in Yang, Chu, and Chen (2012) a cutting plane algorithm based on exact knapsack separation is used to strengthen the formulation of the SSCFLP and a cut–and–solve framework is used to solve the strengthened formulation.

We propose an improved cut-and-solve algorithm running in three phases and significantly faster on known test beds than the best solution methods currently known for this problem. The four main contributions of this paper are:

1. We characterize the facets of the knapsack structures arising from the capacity constraints and exploit the result when strengthening the lower bound of the problem.

2. We propose a new simple local branching heuristic that takes advantage of the two decision levels in the SSCFLP.

3. We suggest a straightforward way of choosing piercing cuts and a number of speed-up techniques for the cut–and–solve algorithm that significantly improve the performance of this framework.

4. We carefully combine the three components above in a three–phased algorithm and compare it to state–of–the–art algorithms.

The paper is organized as follows: Section 2 gives a mathematical programming formulation of the SSCFLP. Section 3 describes the three phases of the algorithm, and Section 4 reports extensive computational results comparing the algorithm to commercial as well as specialized algorithms. Finally, Section 5 gives conclusions and directions for further research.

2 Problem formulation

Let $\mathcal{I}$, $|\mathcal{I}| = n$, be a set of potential facility sites and $\mathcal{J}$, $|\mathcal{J}| = m$, be a set of customers. Each facility, $i \in \mathcal{I}$, has a fixed capacity, $s_i > 0$, and each customer has a fixed and known demand, $d_j > 0$. Opening a facility i results in a fixed cost $f_i \geq 0$ and allocating a customer j to a facility i involves a cost of $c_{ij} \geq 0$. Let y_i be a binary variable equaling one only if a facility is opened at site i and similarly let x_{ij} be a binary variable that equals one only if customer j is allocated to facility i. The SSCFLP can then be stated as the following ILP

problem:

$$\min \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{i\in\mathcal{I}} f_i y_i \tag{O}$$

$$\text{st.:} \sum_{i\in\mathcal{I}} x_{ij} = 1 \quad \forall j \in \mathcal{J}, \tag{D}$$

$$\sum_{j\in\mathcal{J}} d_j x_{ij} \le s_i y_i \quad \forall i \in \mathcal{I}, \tag{C}$$

$$\sum_{i\in\mathcal{I}} s_i y_i \ge D, \tag{T}$$

$$0 \le x_{ij}, y_i \le 1 \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}, \tag{N}$$

$$y_i,\ x_{ij} \in \{0,1\} \quad \forall i \in \mathcal{I},\ j \in \mathcal{J}. \tag{I}$$

The objective function (O) minimizes the total cost. Constraints (D), referred to as the *demand constraints*, make sure that each customer is allocated to exactly one facility. The *capacity constraints* (C) ensure that the capacities of the facilities are respected. In the total demand constraint (T), D represents the total demand. This constraint is in fact redundant, but it will be used later to tighten the LP relaxation of the problem. Constraints (N) state that all variables should be non-negative and less or equal to one. Finally, constraints (I) state that all variables should be binary. It will be assumed throughout the paper that all parameters are integral.

3 Solution methodology

For solving the SSCFLP to optimality, we propose an algorithm running in three phases (see Algorithm 1). The first phase consists of a cutting plane algorithm that separates fractional solutions from the knapsack polytopes defined by the capacity constraints (C) and the total demand constraint (T). In the second phase we implement a local branching heuristic based on a two level local branching strategy that exploits the two levels of decisions (location and allocation) in order to produce an initial near optimal solution. And finally, a third phase based on the cut–and–solve framework is used to solve the problem to proven optimality.

Following the notation used in Cornuejols, Sridharan, and Thizy (1991) we let $P(R)$ denote the set of solutions satisfying a set of constraints R and $\text{conv}\big(R\big) \coloneqq \text{conv}\big(P(R)\big)$ denotes the convex hull of these solutions. For simplicity, $P(RS) \coloneqq P(R) \cap P(S)$ and $\text{conv}\big(RS\big) \coloneqq \text{conv}\big(P(RS)\big)$. Furthermore, by $\mathcal{P}(R)$ we define the optimization problem

$$\mathcal{P}(R): \quad \min\{cx + fy \,:\, (x,y) \in P(R)\}.$$

Using this notation, the SSCFLP can be written as $\mathcal{P}_{\text{IP}} \coloneqq \mathcal{P}(DCTI)$ and its (weak) linear relaxation $\mathcal{P}_{\text{LP}} \coloneqq \mathcal{P}(DCTN)$. Furthermore, the conventional notation $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ is used. Solutions corresponding to a lower bound on the optimal solution will be denoted $(\underline{x}, \underline{y})$ and solutions corresponding to upper bounds will be denoted $(\overline{x}, \overline{y})$.

Phase 1:

1.1: Apply the cutting plane algorithm described in Algorithm 2.

1.2: If the solution returned is integral, stop as it is an optimal solution. Else go to Phase 2.

Phase 2:

2.1: Apply the local branching heuristic described in Section 3.2.

2.2: If the solution value of the returned solution equals the lower bound from Phase 1, stop as the solution is optimal. Else go to Phase 3.

Phase 3:

3.1: Apply the cut–and–solve algorithm described in Algorithm 4 which returns an optimal solution or a proof of infeasibility.

Algorithm 1: Summary of the complete three–phase algorithm

3.1 Phase 1 – Cutting planes

It is well known that the lower bound obtained by solving $\mathcal{P}_{\text{LP}}$ is usually very weak. Therefore, *implied variable upper bounds*

$$x_{ij} - y_i \leq 0, \quad \forall i \in \mathcal{I},\ j \in \mathcal{J},$$

are often added to $\mathcal{P}_{\text{LP}}$. The resulting LP is, however, very large and consequently hard to solve, which is why we add these constraints on the fly as needed.

In order to further strengthen the LP bound, we ideally want to solve

$$\min\{cx + fy \ :\ (x, y) \in P(D) \cap \text{conv}\big(CTI\big)\}.$$

One particular way to do this is to relax the demand constraints (D) in a Lagrangean manner and to solve the resulting Lagrangean dual. From Geoffrion (1974) we know that solving the Lagrangean dual implicitly corresponds to building the convex hull of solutions to the kept constraints. In this paper, however, we want to approximate the convex hull of integer solutions to the capacity constraints (C) and the total demand constraint (T) by cutting planes. By doing so, we avoid separating from the entire polytope defined by $\text{conv}\big(CTI\big)$ and instead we need only consider one constraint at a time.

The following observations will be used to further reduce the effort in separating fractional points from $\text{conv}\big(CTI\big)$. Let

$$\mathcal{X} = \{x \in \{0,1\}^m \ :\ a^T x \leq a_0\},$$
$$\mathcal{X}_y = \{(x, y) \in \{0,1\}^{m+1} \ :\ a^T x \leq a_0 y\},$$

where $a_k > 0$ for $k = 0, 1, \ldots, m$. Moreover, assume that $a_j \leq a_0$ for all $j = 1, \ldots, m$ as if this was not the case, the corresponding variables could be removed. Now observe that if $\pi^T x \leq \pi_0$ is a valid inequality for $\mathcal{X}$, then $\pi^T x \leq \pi_0 y$ is valid for $\mathcal{X}_y$. Furthermore, it is easily seen that if $\pi^T x \leq \pi_0$ is facet defining for $\text{conv}(\mathcal{X})$, then $\pi^T x \leq \pi_0 y$ will be facet defining for $\text{conv}(\mathcal{X}_y)$, implying that strong cutting planes derived from $\text{conv}(\mathcal{X})$ will be strong for $\text{conv}(\mathcal{X}_y)$. In fact, Theorem 1 states the relation between facets of $\text{conv}(\mathcal{X})$ and $\text{conv}(\mathcal{X}_y)$.

Theorem 1. *All facet defining inequalities of* $\text{conv}(\mathcal{X}_y)$*, different from the trivial facets* $-x_j \leq 0$ *and* $y \leq 1$*, are of the form* $\pi^T x \leq \pi_y y$*, where* $\pi \geq 0$ *and* $\pi_y > 0$ *and* $\pi^T x \leq \pi_y$ *is facet defining for* $\text{conv}(\mathcal{X})$*.*

Proof. First of all, as $a_j > 0$ for $j \in \mathcal{J} \cup \{0\}$, $\mathcal{X}$ is an independence system. Hence, all valid inequalities of $\mathcal{X}$, except $x_j \geq 0$, are of the form $\pi^T x \leq \pi_0$ with $\pi_0 > 0$ and $\pi_j \geq 0$ for all $j \in \mathcal{J}$ (cf. (Nemhauser and Wolsey, 1988, p. 237)). Furthermore, substituting $y' = 1 - y$ results in $\mathcal{X}_{y'} = \{(x, y') \in \{0,1\}^{m+1} \ : \ a^T x + a_0 y' \leq a_0\}$ being an independence system as well. Therefore all valid inequalities of $\mathcal{X}_{y'}$ except $x_j \geq 0$ and $y' \geq 0$ are of the form $\alpha^T x + \alpha_y y' \leq \alpha'_0$ where $\alpha_j \geq 0$ for all $j = 1, \ldots, m$, $\alpha_y \geq 0$ and $\alpha'_0 > 0$. Hence, all valid inequalities for $\text{conv}(\mathcal{X}_y)$ different from $x_j \geq 0$ and $y \leq 1$ are of the form $\alpha^T x - \alpha_y y \leq \alpha'_0 - \alpha_y =: \alpha_0$, with $\alpha_j, \alpha_y \geq 0$ for all $j \in \mathcal{J}$. In addition $\alpha_0 \geq 0$ since $(x, y) = (0, 0)$ is feasible.

Secondly we have that $\pi^T x \leq \pi_0$ is facet–defining for $\text{conv}(\mathcal{X})$ if and only if $\pi^T x \leq \pi_0 y$ is facet-defining for $\text{conv}(\mathcal{X}_y)$. To see this, assume that $\pi^T x \leq \pi_0 y$ is facet–defining for $\text{conv}(\mathcal{X}_y)$ and let $(x, y)^l$, $l = 1, \ldots, m+1$ be $m+1$ affinely independent points from the facet

$$\{(x, y) \in \text{conv}(\mathcal{X}_y) \ : \ \pi x = \pi_0 y\}.$$

From Proposition 6.6 on page 108 of Nemhauser and Wolsey (1988) these affinely $m+1$ independent points can be assumed to be from $\mathcal{X}_y$. One of the $m+1$ independent points on the facet is the point $(x, y) = (\mathbf{0}, 0)$. Let this point be $(x, y)^0$. For all other points we have $y^l = 1$. Accordingly the points x^l $(l = 1, \ldots, m)$ give m affinely independent points on the face $\{x \in \text{conv}(\mathcal{X}) \ : \ \pi x = \pi_0\}$. Hence, $\pi x \leq \pi_0$ is facet–defining for $\text{conv}(\mathcal{X})$. Conversely, if $\pi x \leq \pi_0$ is facet–defining for $\text{conv}(\mathcal{X})$, there exists m affinely independent points $x^l \in \text{conv}(\mathcal{X})$, $l = 1, \ldots, m$ such that $\pi^T x^l = \pi$. The $m+1$ points defined by $(x, y)^0 = (\mathbf{0}, 0)$ and $(x, y)^l = (x^l, 1)$ are then affinely independent points all on the face $\{(x, y) \in \text{conv}(\mathcal{X}_y) \ : \ \pi^T x = \pi_0 y\}$. Hence $\pi^t x \leq \pi_0 y$ is facet–defining for $\text{conv}(\mathcal{X}_y)$. From this argument it also naturally follows that if $\pi^T x \leq \pi_0 y$ is facet–defining for $\text{conv}(\mathcal{X}_y)$ and different from $-x_j \leq 0$ and $y \leq 1$, then $\pi^T \geq 0$ and $\pi_0 > 0$.

The only thing that remains to be shown is that if $\alpha^T x - \alpha_y y \leq \alpha_0$ is facet–defining for $\text{conv}(\mathcal{X}_y)$ and different from $-x_j \leq 0$ and $y \leq 1$ then it is dominated by or equivalent to $\alpha^T x \leq (\alpha_y + \alpha_0) y$. Suppose $\alpha_0 = 0$, then the result follows immediately. If on the other hand $\alpha_0 > 0$ then we have that for $y = 1$ the two inequalities are identical, but for $y = 0$ the first inequality gives $\alpha^T x \leq \alpha_0$ which can be strengthened to $\alpha^T x \leq \alpha_0 y = 0$.

Step 0: Set $Z^0 = -\infty$ and iteration counter $k = 0$.

Step 1: Set $k = k + 1$. Solve the program $\mathcal{P}_{\text{LP}}$ and let $(\underline{x}, \underline{y})$ be an optimal solution and Z^k the solution value.

Step 2: If $(\underline{x}, \underline{y})$ is integral, return $(\underline{x}, \underline{y})$ as it is optimal.

Step 3: If $Z^k - Z^{k-1} < \varepsilon$, return $(\underline{x}, \underline{y})$.

Step 4: For each $i = 1$ to n do the following

Step 4.1 Add all violated implied bounds of the form $x_{ij} - y_i \leq 0$ to $\mathcal{P}_{\text{LP}}$.

Step 4.2 Separate $\underline{\mathbf{x}}_i$ from $\text{conv}\big(\{\mathbf{x}_i \in \{0,1\}^m : \sum_{j=1}^m d_j x_{ij} \leq s_i\}\big)$ using a General Lifted Cover Inequality or a Fenchel inequality (see Section 3.1.1). If the resulting inequality $\pi \mathbf{x}_i \leq \pi_0$ is violated by $\underline{\mathbf{x}}_i$, add the translated cutting plane $\pi \mathbf{x}_i \leq \pi_0 y_i$.

Step 5: Separate $\underline{y}$ from $\text{conv}\big(\{y \in \{0,1\}^{|\mathcal{I}|} : \sum_{i \in \mathcal{I}} s_i y_i \geq \sum_{j \in \mathcal{J}} d_j\}\big)$ using a General Lifted Cover Inequality or a Fenchel inequality (see Section 3.1.1). If the resulting inequality $\pi y \leq \pi_0$ is violated by $\underline{y}$, add the cutting plane $\pi y \leq \pi_0$.

Step 6: If $k < K$ go to Step 1, else return $(\underline{x}, \underline{y})$.

Algorithm 2: Summary of the cutting plane algorithm. The parameter $\varepsilon > 0$ is a suitable number used to check if the improvement in the lower bound is below a predefined limit. The parameter K defines the maximum number of iterations of the cutting plane algorithm.

Concluding, any facet–defining inequality of $\text{conv}\big(\mathcal{X}_y\big)$, different from the facets $x_j \geq 0$ and $y \leq 1$ are of the form $\pi^T x \leq \pi_y y$, where $\pi \geq 0$, $\pi_y > 0$, and $\pi^T x \leq \pi_y$ is facet–defining for $\text{conv}\big(\mathcal{X}\big)$. □

Theorem 1 states that any facet–defining inequality for $\text{conv}\big(\mathcal{X}_y\big)$ can be derived by generating facet–defining inequalities for $\text{conv}\big(\mathcal{X}\big)$ and then translating these inequalities. Therefore we will omit the location variable when generating cutting planes from the capacity constraints (C) and then translate the resulting inequality by multiplying the right–hand side by y. The cutting plane algorithm developed in this paper is described in Algorithm 2. Two types of cutting planes are used, namely lifted cover inequalities and Fenchel cutting planes. All cutting planes are described for the knapsack polytope defined by the capacity constraints, but it should be obvious that they apply to the total demand constraint (T) as well (simply complement location variables by $z_i = 1 - y_i$ and an ordinary knapsack constraint is obtained).

Regarding the generation of violated lifted cover inequalities, we use a slight modification of the procedure described by Gu, Nemhauser, and Savelsbergh (1998). The procedure

sorts the variables according to non–increasing value of the LP–value. An initial minimal cover is generated by picking the variables with largest LP–value until a cover is reached. The variables with LP–value one define an initial cover inequality, and the remaining variables are lifted to reach a valid, strong cutting plane.

As Fenchel cutting planes are less known than lifted cover inequalities, we go into more depth with these inequalities in the next section.

3.1.1 Fenchel cutting planes

Let $\mathcal{X}'_{iy} = \{(\mathbf{x}_i, y_i) \in \{0,1\}^{|\mathcal{J}|+1} \ : \ \sum_{j\in\mathcal{J}} d_j x_{ij} \leq s_i\}$ and $\text{proj}_x(\mathcal{X}'_{iy})$ be the projection of $\text{conv}\big(\mathcal{X}'_{iy}\big)$ on the space of the x variables. As the set of all integer solutions to $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$ is an independence system, all non-dominated valid inequalities for $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$ different from $-x_{ij} \leq 0$ are of the form $\lambda^T \mathbf{x}_i \leq 1$ (Nemhauser and Wolsey, 1988, p.237). A Fenchel cutting plane is in this case a cutting plane of the form $\lambda^T \mathbf{x}_i \leq 1$ that cuts as deep as possible into the relaxed polytope. Generating a Fenchel cutting plane for $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$ can be seen as a proof of the existence of a hyperplane, that separates a fractional solution $\underline{\mathbf{x}}_i$ from $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$. Boyd (1993) states the separation problem for the knapsack polytope $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$ as

$$\begin{aligned} \nu = \max\ & \underline{\mathbf{x}}_i^T \lambda \\ \text{st.:}\ & \mathbf{x}_i^T \lambda \leq 1, \quad \forall \mathbf{x}_i \in \text{proj}_x\big(\mathcal{X}'_{iy}\big), \\ & 0 \leq \lambda_j \leq 1, \quad \forall j \in \mathcal{J}. \end{aligned} \tag{1}$$

If $\nu > 1$, then there exists a hyperplane that separates $\underline{\mathbf{x}}_i$ from $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$, and this hyperplane is given by $\sum_{j\in\mathcal{J}} \lambda^*_j x_{ij} \leq 1$, where λ^* is an optimal solution to (1). In fact, Boyd (1993) proves a reduction result stating that there exists a hyperplane separating $\underline{\mathbf{x}}_i$ from $\text{proj}_x\big(\mathcal{X}'_{iy}\big)$ if and only if there exists a hyperplane separating $\underline{\mathbf{x}}_i$ from

$$\text{proj}_x\big(\mathcal{X}'_{iy}\big) \cap \big\{\mathbf{x}_i \in \mathbb{R}^{|\mathcal{J}|} \ : \ x_{ij} = 0 \ \forall j \in \mathcal{J}^0, \ x_{ij} = 1 \ \forall j \in \mathcal{J}^1\big\},$$

where $\mathcal{J}^0 = \{j \in \mathcal{J} : \underline{x}_{ij} = 0\}$ and $\mathcal{J}^1 = \{j \in \mathcal{J} : \underline{x}_{ij} = 1\}$. We can therefore exclude these variables from the separation problem and thereby reduce the effort quite substantially.

The separation problem (1) obviously has too many constraints to be taken directly into account. For that reason, Boccia, Sforza, Sterle, and Vasilyev (2008) propose a generic row-generation procedure. We do, however, propose an equivalent column generation procedure which is summarized in Algorithm 3 (Step 0–Step 8). We have chosen to implement the method as a column generation procedure instead of a row generation procedure as the basis matrix of the problem (2) only has the dimension of the number of variables included in the separation problem, where the basis matrix of the equivalent row generation procedure has the dimension of H (or the dimension of linear independent rows in H). As rows are generated, the dimension of the basis matrix increases, and the simplex iterations become computationally more demanding. This is avoided when solving problem (1) by column generation. When solving the binary knapsack problems in Step 5 of Algorithm 3 one

Step 0: Input a capacity constraint $\sum_{j\in\mathcal{J}} d_j x_{ij} \le s_i$ and a fractional solution $\underline{\mathbf{x}}_i$.

Step 1: Let $\mathcal{J}^1$, $\mathcal{J}^0$ and $\mathcal{J}^f$ be the set of indices where $\underline{x}_{ij} = 1$, $\underline{x}_{ij} = 0$, and $0 < \underline{x}_{ij} < 1$, respectively. Define the residual capacity, $\overline{s}$, as $\overline{s} = s_i - \sum_{j\in\mathcal{J}^1} d_j$. Set the iteration counter $l = 0$ and initialize the set of solutions to the knapsack problem $H = \{e_j\}_{j\in\mathcal{J}^f}$, where e_j is the j'th column vector of the $|\mathcal{J}^f| \times |\mathcal{J}^f|$ identity matrix.

Step 2: Set up the linear column generation master problem

$$\begin{aligned} \min \ & \sum_{h\in H} \gamma_h \\ \text{st.:} \ & \sum_{h\in H} x_i^h \gamma_h \ge \underline{x}_{ij}, \quad \forall j \in \mathcal{J}^f \\ & \gamma_h \ge 0 \end{aligned} \tag{2}$$

where H is a set of solutions satisfying the knapsack constraint $\sum_{j\in\mathcal{J}^f} d_j x_{ij} \le \overline{s}$ generated in step 5. If $l = 0$, then $H = \emptyset$.

Step 3: Solve the linear column generation master problem and denote an optimal primal–dual pair (γ^l, λ^l).

Step 4: If $\sum_{j\in\mathcal{J}^f} \underline{x}_{ij}\lambda_j^l \le 1$, no violated cutting plane exists. Therefore stop.

Step 5: Solve the binary knapsack problem

$$Z_{kp} = \max\{\sum_{j\in\mathcal{J}^f} \lambda_j^l x_j \ : \ \sum_{j\in\mathcal{J}^f} d_j x_j \le \overline{s}, \quad x_j \in \{0,1\}\}$$

and denote an optimal solution x^l.

Step 6: If $Z_{kp} \le 1$ go to step 7. Otherwise, the inequality $\sum_{j\in\mathcal{J}^f} \lambda_j^l x_{ij} \le 1$ is not valid for

$$\text{proj}_x\big(\mathcal{X}'_{iy}\big) \cap \{\mathbf{x}_i \in \mathbb{R}^{|J|} \ : \ x_{ij} = 1, \ \forall j \in J_1, \text{ and } x_{ij} = 0, \ \forall j \in \mathcal{J}^0\}$$

Therefore add the solution x^l to H, set $l \coloneqq l + 1$ and go to step 2.

Step 7: Down–lift all variables in $\mathcal{J}^1$ and up–lift all variables in $\mathcal{J}^0$.

Step 8: Output the violated cutting plane $\pi\mathbf{x}_i \le \pi_0 y_i$.

Algorithm 3: Column generation procedure for separating a Fenchel inequality for capacity constraint i in Step 4.2 of Algorithm 2.

should note, that the cost coefficients λ^k are fractional, whereby numerical problems can arise. In order to overcome this, we simply multiply the coefficients with a large scalar and then round the coefficients to the nearest integer, which allows to use the highly efficient COMBO algorithm (Martello, Pisinger, and Toth, 1999) for solving the binary knapsack problems. When it is no longer possible to find violated inequalities this way, we submit the generated cutting plane to the rounding procedure proposed by Kaparis and Letchford (2010) and check the result for validity. If this is the case, the cutting plane is returned, otherwise a weakened inequality is returned. The cutting plane separated by the column generation procedure described in Algorithm 3 is, after lifting the fixed variables, of the form $\pi^T x \leq \pi_0$. Again we translate the resulting cutting plane by multiplying the right–hand side by y_i and obtain $\pi^T x \leq \pi_0 y_i$ in Step 8.

3.2 Phase 2 – Local branching

Local branching (*LB*) is in principle a technique for finding exact solutions to *mixed integer programs* (MIP). It can, however, easily be converted into a heuristic, e.g. by setting a limit on the time used for solving a given instance or a limit on the number of improving solutions. The technique uses a general purpose MIP–solver to solve restricted subproblems of the original problem instance defined by linear *local branching constraints*, which in turn are derived from a feasible solution. For a detailed introduction to LB, the reader is referred to Fischetti and Lodi (2003) and Fischetti, Polo, and Scantamburlo (2004). The two level LB heuristic developed in this paper is summarized as follows; First, an initial feasible solution is found by exploiting the LP-solution, second, the locational decisions are refined by local branching, and finally the allocation of customers is determined, likewise by local branching.

3.2.1 Initial feasible solution

We utilize the information available from the cutting plane algorithm to find an initial solution. Let $(\underline{x}, \underline{y})$ be the fractional solution from the last iteration of the cutting plane algorithm (Algorithm 2). Assuming that it is more likely that a facility i is open in an optimal solution if $\underline{y}_i > 0$ than when $\underline{y}_i = 0$, we add a constraint of the form

$$\sum_{i:\underline{y}_i=0} y_i \leq h_1,$$

for a small integer value of h_1. This way at most h_1 of the facilities not used in the fractional solution can be used in a heuristic solution to the SSCFLP. The general purpose MIP–solver is then used to solve the resulting problem. The MIP-solver stops if a heuristic stop criterion is met (see Section 3.2.4). The solution returned is denoted (x^0, y^0).

3.2.2 Refining the locational decision

Next, based on the initial solution (x^0, y^0), we add a local branching constraint for the location variables of the form

$$\sum_{i \in \mathcal{I}} |y_i^0 - y_i| \leq h_2 \Leftrightarrow \sum_{i:y_i^0=0} y_i + \sum_{i:y_i^0=1} (1 - y_i) \leq h_2,$$

where h_2 is a small integer. The problem is now solved with the additional requirement that only improving solutions are accepted. Again, a heuristic stop criterion is used to end the optimization (see Section 3.2.4). The resulting solution is denoted (x^1, y^1). After this refinement of the locational decision, all location variables where $y_i^1 = 0$ are fixed to zero.

3.2.3 Refining the allocation decision

Having almost fixed the location variables (it is still possible to close open facilities), the allocation variables are considered next. Summing up the assignment constraints over $j \in \mathcal{J}$ any feasible solution satisfies

$$\sum_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} x_{ij} = |\mathcal{J}|.$$

We can therefore define a k-opt neighborhood around the allocation corresponding to x^1 by means of a local branching constraint:

$$N(x^1, k) = \{x \in \{0,1\}^{n \times m} \ : \sum_{(i,j):x_{ij}^1=1} x_{ij} \geq |\mathcal{J}| - k\}, \tag{3}$$

for a given positive integer k. Adding this constraint will significantly reduce the solution space, but the problem can still be too hard to solve to optimality. Therefore the search is stopped prematurely if a heuristic stop criterion is met. If an improving solution is found, say (x^2, y^2), the constraint (3) is removed and a reversed local branching constraint of the form

$$\sum_{(i,j):x_{ij}^1=1} x_{ij} \leq |\mathcal{J}| - k - 1,$$

is added to the problem. This constraint removes the neighbourhood $N(x^1, k)$ from further consideration. The procedure is repeated and the feasible set in iteration n is given by

$$P(DCTI, N(x^n, k)) \setminus P(N(x^1, k), \ldots, N(x^{n-1}, k))$$

If an iteration does not lead to any improvement, the best solution found so far is returned.

3.2.4 Heuristic stop criteria

Although the local branching constraints significantly reduce the size of the problem, solving every subproblem to optimality is usually too time consuming. Instead we set a limit on

the time and an upper bound for the number of improving solutions for the MIP-solver. Ideally, these limits should depend on the problem data, but fixed limits seem to work well in practice.

We do, however, distinguish between local branching on the location variables and on the allocation variables. As the neighborhoods searched when refining the location decisions are much larger than those examined for the allocation variables, we set a time limit of t_y seconds for those subproblems and a time limit of $0 < t_x < t_y$ for the allocation refinement.

When refining the locational decisions, we do not impose any upper bound on the number of improved solutions, because the MIP-solver usually finds a large number of improving solutions for the locational decisions in short computation times. For the allocation variables, we set a limit of $1 < k_x < \infty$ improving solutions, because in most cases only a few improving solutions exist in the much smaller neighborhoods of the current allocation.

3.3 Phase 3 – Cut–and–solve

The cut–and–solve framework is essentially a branch–and–bound algorithm which branches on a *set* of variables instead of branching on single fractional variables. At each level in the search tree (see Figure 1) there are only two nodes. The left node is associated with the linear constraint that the sum of the variables in a set Ω is less than or equal to an integer γ. This subproblem is called the *sparse problem* (*SP*). The right node is associated with the sum being larger than or equal to $\gamma + 1$ and this problem is called the *dense problem* (*DP*). The constraint associated with the DP is called a *piercing cut* and these cuts are accumulated. An obvious choice for binary programs is to set the parameter γ equal to zero, as this completely fixes the variables in the set Ω to zero in the SP. Doing this will often allow the use of a general purpose MIP–solver to solve the sparse problems. The optimal solution to the SP provides an upper bound on the optimal solution value to SSCFLP (as far as the SP shows a feasible solution). If this upper bound improves the incumbent, it is updated. Concurrently, a lower bound, $\underline{Z}$, for DP is obtained from a relaxation of the DP. If the lower bound for the DP is not smaller than the value of the incumbent, we know that no improving solutions can exist in the yet unexplored solution space defined by the DP, and the incumbent is proven optimal. This procedure is repeated until an optimal solution is found (using the convention that a lower bound on an infeasible DP is equal to infinity, this procedure will return an optimal solution). For a more detailed introduction to the cut–and–solve framework, the reader is referred to Climer and Zhang (2006).

3.3.1 Relaxation and piercing cuts

The way the DP is relaxed and piercing cuts are derived is of utmost importance for the efficiency of the cut–and–solve approach. First of all, we want to use a relaxation that gives strong lower bounds. The lower bound should also increase rapidly as piercing cuts

Step 0 Obtain a lower bound of the dense problem.

Step 1 Select a piercing cut.

Step 2 Find optimal solution in space removed by the piercing cut (sparse problem).
Update the incumbent if necessary.

Step 3 If lower bound $\geq$ incumbent, return the incumbent.

Step 4 Add piercing cut to dense problem and go to Step 0.

Algorithm 4: Generic cut–and–solve algorithm.

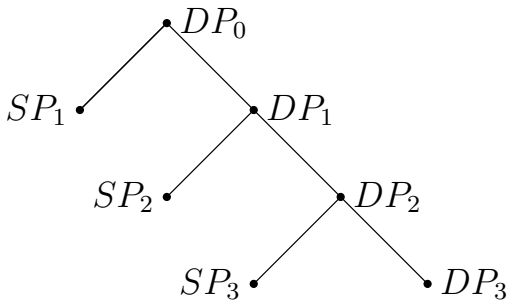

Figure 1: Cut–and–solve search tree.

are added. Climer and Zhang (2006) suggest to use the linear programming relaxation and to use reduced costs to define the piercing cuts. The problem is, however, that the linear programming relaxation of the SSCFLP is often highly degenerate and adding piercing cuts does not improve the lower bound fast enough to obtain an efficient algorithm. We therefore adopt the *partial integrality strategy* proposed by Yang et al. (2012). This strategy means that we preserve the integrality of the location variables and relax the allocation variables to be continuous variables, leading to a CFLP as a relaxation of the SSCFLP. That is, the relaxation keeps the 0–1 requirement on the y_i–variables but relaxes the x_{ij}–variables to be continuous, effectively allowing a customer's demand to be split between two or more open facilities. Defining the set Ω can therefore be done in a straightforward way, by letting

$$\Omega = \{i \in \mathcal{I} \,:\, \underline{y}_i = 0\}, \tag{4}$$

where $\underline{y}$ is an integer solution to the location variables in the DP and the piercing cut becomes $\sum_{i\in\Omega} y_i \geq 1$. This definition is mainly motivated by the assumption that an optimal solution to the CFLP should have a number of attributes in common with an optimal solution to the SSCFLP. This assumption is supported by the computational results reported in Section 4. Furthermore, choosing Ω as in (4), many variables can be fixed in the SP as $\sum_{i\in\Omega} y_i = 0$ implies that $x_{ij} = 0$ for all $i \in \Omega$ and $j \in J$.

3.3.2 Termination

When proving finiteness of the cut–and–solve algorithm, Climer and Zhang (2006) assume finiteness of the problem's feasible region and that all SPs have at least one feasible solution.

The latter is, however, not necessarily true with the choice of Ω given in (4) as it might be impossible to create a single–source solution using the set of facilities used in the CFLP–relaxation. Termination is, however, easily proven in Proposition 1.

Lemma 1. *Assume that the piercing cuts are defined by the set* Ω *stated in* (4)*. Then the algorithm terminates if the algorithms for solving the dense and the sparse problems are guaranteed to terminate.*

Proof. First of all, the set of feasible facility constellations for the CFLP version of the SSCFLP is finite. There are $|\{\iota \subseteq \mathcal{I} \,:\, \sum_{i\in\iota} s_i y_i \geq D\}|$ such solutions. The piercing cut

$$\sum_{i:\underline{y}_i=0} y_i \geq 1,$$

removes the current solution to the modified CFLP from further considerations (in fact it also removes all solutions which consist of a subset of the facilities used in the CFLP solution). As there is a finite number of solutions to the locational variables of the CFLP, the cut–and–solve algorithm terminates if the algorithms used to solve the dense and the sparse problems terminate. □

3.3.3 Variable fixing and pruning of nodes

As the open facilities in the solution to the DP often constitute a minimal cover of the total demand, it is usually possible to fix many of the location variables in the SP to a value of one in the following way. If

$$\sum_{i\in(I\setminus\Omega)\setminus\{\tilde{i}\}} s_i < D,$$

for some $\tilde{i} \in \mathcal{I} \setminus \Omega$, the location variable $y_{\tilde{i}}$ can be fixed to one as the total demand cannot be covered otherwise. If all variables in $\mathcal{I} \setminus \Omega$ can be fixed to one, the SP reduces to a *generalized assignment problem* (*GAP*). Though GAP is also $\mathcal{NP}$-hard, the only remaining decisions are to decide on the allocation of customers, whereby the effort for solving the SP is reduced.

Note that it is not necessary to find an optimal solution to the SP in each iteration. It suffices to prove that no improving solution exists in the space of solutions to the SP. The optimization of an SP can then be prematurely terminated if a lower bound of this SP exceeds or equals the value of the incumbent. For the same reason, the last DP can be stopped prematurely as soon as it can be proven that its optimal solution does not fall below that of the incumbent.

3.3.4 Cutting planes for the SP

In order to solve the DPs and SPs efficiently, the cutting planes generated in phase one of the algorithm are all appended to these programs. Although the cuts generated for the

capacity constraints are not valid for the CFLP version of the SSCFLP, they are valid for the SSCFLP. The modified CFLP is therefore still a relaxation of the SSCFLP. The variable fixing (4) defined by the solution to the DP may, of course, remove the part of the solution space for which these cuts were generated. If this is the case, the cuts are only of limited use in the SP. We thus include additional cutting planes at the root node of each SP so that these problems can be solved efficiently. The cutting planes are generated in the same way as in the cutting plane algorithm shown in Figure 2. The implied variable bounds are not used as they seem to contribute little to improvements in the lower bound at this stage. The reason for this lack of effect is simply that many location variables can be fixed to one a priori, making the implied bounds $x_{ij} \leq y_i$ redundant.

4 Computational experiments

In this section we test the efficiency of the three–phased algorithm. We do this by first comparing the efficiency of the improved cut–and–solve algorithm used in Phase 3 to CPLEX' branch–and–cut algorithm. Secondly, we compare the three–phased algorithm to a state–of–the–art algorithm developed by Yang et al. (2012), which to the best of our knowledge is one of the most efficient algorithms for solving the SSCFLP. Finally, our algorithm is tested on a set of new instances such that some insight can be obtained into which parameter settings constitute difficult instances.

4.1 Implementation detail

Our implementation uses the linear programming solver and the branch–and–cut framework provided by ILOG CPLEX Concert Technology 12.3 (*CPLEX*). All programs have been coded in C++ and C languages and compiled using g++ and gcc, respectively, with optimization option O2. All experiments were carried out on a Dell Vostro 3450 laptop with 4 GB RAM and a 2.5 GHz Intel® Core i5-2450M processor running a 32 bit version of Ubuntu 12.04.

In CPLEX, the ParallelMode switch is set to deterministic, such that the running times for different instances can be compared. Moreover, in Phase 1, the default settings are used to solve the linear programming problems in each iteration of the cutting plane algorithm. This means that CPLEX is allowed to preprocess the problem instance before the separation routines are used. The separation routines are called for each knapsack constraint (capacity or total demand) if the slack is below a certain threshold. A rather large slack is permitted, as good results were obtained by including separating hyperplanes even for capacity constraints showing large values of the slack variables at the current LP solution.

CPLEX is also used as MIP solver for solving the restricted problems in Phase 2. As mentioned in Section 3.2, the parameters h_1, h_2 and k that determine the neighborhoods' sizes are fixed independently of the problem data and attain values of 0, 1 and 5, respectively.

In order to solve the dense problem in Phase 3, CPLEX is applied with the MIPSearch

switch set to "traditional branch–and–cut". The relative and the absolute optimality gaps are both set to zero. All other settings are at their default values. All cuts generated in Phase 1 are appended to the dense problem. CPLEX is also used for solving the sparse problem. Settings are as for the dense problem, except that the absolute optimality gap is set to 0.99 due to integral data. Furthermore, we branch on the location variables before branching on the assignment variables. As the SPs often require memory exceeding the RAM capacity, the NodeFileInd switch is set to 3 to allow for the branching tree to be written to a file on the disk. All cuts generated in Phase 1 are added to this program as well, and furthermore, we use a cut callback to separate General Lifted Cover Inequalities and Fenchel cutting planes in order to strengthen the lower bound of the sparse problem. The cut generation is limited to the root node of the SP as experience has shown that cutting planes are more effective at nodes in the top of the branching tree. The codes are publicly available (see Gadegaard, Klose, and Nielsen (2016)).

4.2 Test instances

The three–phased algorithm has been evaluated on four different test beds, denoted TBi, $i = 1, \ldots, 4$. TB1 consists of 57 instances from Díaz and Fernández (2002) ranging from small instances with 10 potential facility sites and 20 customers to larger instances with 30 potential facility sites and 90 customers. These instances are publically available at `http://www-eio.upc.es/~elena/sscplp/index.html`. TB2 consists of 71 instances used by Holmberg et al. (1999) with problems consisting of 10 facility sites and 50 customers up to problems with 30 facility sites and 200 customers. These instances are available at `http://www.mai.liu.se/~kahol/problemdata/cloc/`. The third test bed, TB3, consists of relatively large problem instances reported in Yang et al. (2012). The sizes of these problems range from 30 facility sites and 200 customers to 80 facility sites and 400 customers.

TB4 is a new set of problem instances generated for this paper. It tries to mimic the situation where large fixed opening costs lead to small production costs (e.g. more efficient and therefore expensive machinery leads to lower production costs). The problem instances have been generated so as to be similar to instances in the literature on the CFLP and the SSCFLP (see for example Cornuejols et al. (1991), Klose and Görtz (2007) and Yang et al. (2012)). That is, demands and capacities are uniformly distributed in the intervals $[5, 35]$ and $[10, 160]$, respectively. Then the capacities are scaled so as to obtain a specific ratio between total capacity and total demand. Fixed opening costs are generated using the formula

$$f_i = U(0, 90) + \sqrt{s_i} U(100, 110)$$

where $U(a, b)$ denotes the uniform distribution of the set $\{a, a+1, \ldots, b-1, b\}$. Finally the assignment costs c_{ij} are usually determined as follows: generate facility sites and customers as uniformly distributed points in the unit square and set $c_{ij} = \lfloor\ 10\ \delta(i, j)\ \rfloor$, where $\delta(i, j)$ denotes the Euclidean distance between facility point i and customer point j and $\lfloor \delta(i, j) \rfloor$ denotes the largest integer no greater than $\delta(i, j)$. For the problems in TB4, however, we will let this number denote the transportation cost of delivering customer j's demand from

facility i. That is $t_{ij} = \lfloor 10\ \delta(i,j) \rfloor$. The production cost of producing d_j units of the desired product at facility i is then calculated as

$$p_{ij} = \left\lfloor \frac{2 \max_{i \in I} f_i}{f_i} \right\rceil d_j$$

where $\lfloor \cdot \rceil$ means rounded to nearest integer. This specification has been made in order to reflect the idea that items may be produced at a lower unit price in an expensive facility. Another implication is that low opening costs for a facility will lead to high supply costs, thus making it harder to determine the "right" set of facilities. The assignment cost is then set to be $c_{ij} = t_{ij} + p_{ij}$, that is transportation plus production costs. In order to make the percentage optimality gap, given by

$$gap = \frac{\text{best integer} - \text{best lower bound}}{\text{best lower bound}} 100$$

more "honest", the assignment costs are scaled as follows

$$c_{ij} = c_{ij} - \min_{i \in I} c_{ij}$$

which will decrease the magnitude of the solution values. The assignment cost c_{ij} can then be interpreted as the cost that should be paid if customer j is assigned to facility i instead of the cheapest alternative. Both TB3 and TB4 are publicly available at `https://github.com/SuneGadegaard/SSCFLPsolver`.

These four test beds range from small over medium–sized to large–sized problems. In the instances of TB2, the assignment costs dominate the fixed opening costs whereby an optimal solution tends to include more facilities. In the rest of the test beds, the fixed opening costs dominate the assignment costs, and the optimal solution will therefore often open as few facilities as possible. Furthermore, a new cost structure for the allocation costs is used in TB4. As a result, the instances in TB1-TB4 cover many different scenarios and it should be possible to evaluate the effectiveness of our three–phase algorithm.

Table 1 simplifies the reading of succeeding tables by summarizing the column headings.

4.3 The efficiency of the cut–and–solve algorithm used in Phase 3

In order to test the efficiency of the cut–and–solve algorithm in Phase 3, we have implemented a solution procedure that first uses our cutting plane algorithm (Phase 1), then the optimal solution value of the instance is used as an upper cut-off value in CPLEX in order to mimic a very strong and fast heuristic. Finally, CPLEX with all settings at their default values is used to solve the instances to proven optimality.

Table 2 presents the aggregated results obtained using the combination of our cutting plane algorithm and the branch–and–cut algorithm provided by CPLEX. The branch–and–cut algorithm does a good job on the small test instances of TB1 and TB2. The

Table 1: Abbreviations used in the column headings of the result tables

Description of abbreviations	
ID	The ID of the problem under consideration.
$\|I\| \times \|J\|$	Number of facilities and customers in the instances.
r	Ratio between total capacity and total demand.
%-time in phase	Displays the percentage of the total cpu time used in Phase 1, 2, and 3.
$\underline{Z}$, $\overline{Z}$, Z^*	Denotes the lower bound produced by the first dense problem, best upper bound found using the local branching heuristic and the optimal solution value, respectively.
LB_{gap}	$\frac{Z^* - \underline{Z}}{\underline{Z}} \cdot 100$. That is, a measure of the quality of the lower bound.
UB_{gap}	$\frac{\overline{Z} - \underline{Z}}{\underline{Z}} \cdot 100$. That is, a measure of the quality of the heuristic.
cpu	The average cpu time used by our algorithm to solve the instances in seconds.
CPU	The cpu time, in seconds, used to solve a single instance using our algorithm.
ρ	The ratio between the cpu time used by another algorithm and the cpu time used by the algorithm of this paper.

Table 2: Results obtained using branch–and–cut instead of cut–and–solve in Phase 3.

				ρ		
TB	$\|I\| \times \|J\|$	r	cpu	min	mean	max
1	$10 \times 20 - 30 \times 90$	1.27-5.17	10.04	0.29	5.28	133.20
2	$10 \times 50 - 30 \times 200$	1.37-8.28	1.92	0.23	0.78	2.44
3	$30 \times 200 - 80 \times 400$	1.73-7.30	401.62	1.38	136.47	1652.70
4	$50 \times 100 - 60 \times 300$	2.00-5.00	6,442.88	0.56	10.52	140.42

ρ-value shows that for $TB2$ the branch–and—cut algorithm generally outperforms the cut–and–solve algorithm over the entire test bed. This is due to the small size of these problems; it is less time consuming to solve one integer program than it is to alternate between solving a MIP corresponding to the DP and the restricted MIP defined by the SP. The reason why the branch–and–cut algorithm performs worse relative to our algorithm in TB1 than in TB2 is that the integrality gap between the value of the LP relaxation and the optimal solution value to SSCFLP is larger in TB1 than in TB2. As the cutting plane in conjunction with the partial integrality strategy provides better lower bounds than the LP relaxation, the gap is closed faster.

When we look at TB3 and TB4, it is apparent that the size of the problems in TB3 and the tight capacity to demand ratios of TB4 make the branch–and–cut algorithm much more time consuming than the cut–and–solve. On average, the branch–and–cut algorithm requires about 135 times more cpu time than the cut–and–solve to solve the instances in TB3. Even though the instances in TB4 exhibit a relatively small capacity to demand ratio, implying larger SPs, our algorithm runs 10 times faster than the branch–and–cut algorithm on average. Furthermore, the branch–and–cut algorithm was not able to solve a number of instances (instances n16, n19, n20, n31, n32, n34, and n35) within a time limit of 50,000 cpu seconds. Instances n16, n19, n20, and n35 were all solved by the improved cut–and–solve algorithm within half that time.

We therefore conclude that for small instances with a small integrality gap the cut–and–solve approach is comparable to CPLEX' branch–and–cut algorithm and for large instances the improved cut–and–solve algorithm clearly outperforms CPLEX. Furthermore, the authors would like to note that for some of the instances in TB4 the memory requirement for the branch–and–cut algorithm exceeded 80 GB before the time limit was exceeded; for the cut–and–solve algorithm it never exceeded 7 GB.

4.4 Comparing the three–phased algorithm to a state–of–the–art algorithm

This section compares our three–phased algorithm to the one proposed by Yang et al. (2012). To distinguish the two algorithms, we denote the three–phased algorithm developed in this paper `TP-ALG` and the cut–and–solve algorithm developed in Yang et al. (2012) is denoted `Y-CS`. As Mr. Zhen Yang kindly provided the implementation of the algorithm proposed in Yang et al. (2012), and as the two algorithms have been run on the same machine, the CPU-times are truly comparable. The results obtained for TB1 to TB4 are displayed in Table 3. It is apparent that the lower and upper bounds, $\underline{Z}$ and $\overline{Z}$, are very close to the optimal solution in almost all cases. Except for the first subsets of test instances in TB1, the average deviation of the lower bound $\underline{Z}$ from the optimal solution never exceeds 0.5 percent. The larger gaps in the two first subsets of TB1 mainly stem from two instances, namely d1 and d7, respectively. The absolute gaps in these two instances are relatively small, but as the objective function values are not that large for these instances either, the percentage gap becomes larger. In 4 and 22 instances of TB1 and TB2, respectively, the

Table 3: Aggregated results on the four test beds when `TP-ALG` is compared to `Y-CS`.

						%-time in phase				ρ		
TB	ID	$\|I\| \times \|J\|$	r	LB_{gap}	UB_{gap}	1	2	3	cpu	min	mean	max
1	d1-d6	10×20	$1,32 - 1,54$	1.15	1.53	19.0	35.6	45.4	0.90	1.72	4.68	8.53
	d7-d17	15×30	$1.33 - 3.15$	0.51	0.72	45.0	36.7	18.3	1.47	2.00	8.64	37.94
	d18-d25	20×40	$1.30 - 3.93$	0.13	0.46	46.6	27.3	26.1	3.31	1.65	12.27	38.25
	d26-d33	20×50	$1.27 - 4.06$	0.16	0.55	41.9	37.9	20.2	36.58	0.67	10.20	30.21
	d34-d41	30×60	$1.64 - 5.17$	0.16	0.28	43.2	26.8	30.0	5.42	4.00	12.46	35.17
	d42-d49	30×75	$1.43 - 3.01$	0.03	0.13	40.9	28.7	30.3	14.29	2.02	15.25	30.08
	d50-d57	30×90	$1.49 - 3.47$	0.08	0.20	52.8	25.7	21.5	8.88	3.26	25.94	74.42
			Avg.	0.30	0.53	40.6	28.5	30.9	9.99		12.84	
2	h1-h12	10×50	$1.37 - 2.06$	0.04	0.05	55.4	35.0	9.6	0.20	2.19	4.60	13.66
	h13-h24	20×50	$2.77 - 3.50$	0.01	0.08	69.7	16.6	13.7	0.34	0.65	2.10	8.06
	h25-h40	30×150	$3.03 - 6.06$	0.11	0.23	58.2	24.5	17.3	2.61	1.23	33.40	122.73
	h41-h55	30×100	$1.52 - 8.28$	0.10	0.20	48.1	31.5	20.4	0.67	0.46	24.40	67.20
	h56-h71	30×200	$1.97 - 3.95$	0.08	0.22	60.8	16.5	22.7	5.29	1.52	21.41	116.44
			Avg.	0.07	0.17	60.6	24.7	14.7	2.01		18.64	
3	y1-y5	30×200	$1.73 - 1.98$	0.10	0.78	20.1	40.8	39.1	51.00	4.81	28.11	48.60
	y6-y10	60×200	$2.88 - 3.49$	0.16	0.78	8.5	8.2	83.3	1261.82	0.87	29.54	52.00
	y11-y15	60×300	$3.42 - 5.78$	0.07	1.02	32.0	17.9	50.1	65.63	42.17	114.50	212.97
	y16-y20	80×400	$3.50 - 7.30$	0.05	0.68	27.4	15.0	57.6	228.01	27.49	169.62	370.80
			Avg.	0.10	0.82	22.0	20.5	57.5	401.62		85.44	
4	n1-n5	50×100	2	0.02	—	5,8	26,4	67,8	1078.94	0.84	6.12	18.72
	n6-n10		3	0.02	0.97	7,2	17,6	75,2	931.25	2.34	6.33	9.69
	n11-n15		5	0.02	0.76	45,2	15,2	39,6	16.83	11.77	59.93	127.41
	n16-n20	50×200	2	0.08	—	6,3	7,3	86,4	7871.77	9.72	14.03	18.33
	n21-n25		3	0.01	—	12,6	32,8	54,6	4221.90	5.48	12.17	23.74
	n26-n30		5	0.01	0.13	73,6	13,0	13,4	14.32	42.88	101.34	150.92
	n31-n35	60×300	2	—	—	0,1	0,5	99,4	42938,86	—	—	—
	n36-n40		3	0.01	—	12,4	34,7	52,9	863,94	0.58	20.28	44.14
	n41-n45		5	0.02	—	60,8	22,7	16,5	48.11	79.57	138.38	203.69
			Avg.	0.03	1.09	24,9	18,9	56,2	6442,88		43.11	

cutting plane algorithm was able to close the integrality gap and found an integer solution whereby Phases 2 and 3 were never entered. This never happened for any instances in TB3 and TB4. The main reason seems to be the large number of potential facilities in these test sets, effectively prohibiting the generation of all needed facets.

The simple local branching heuristic in Phase 2 also seems to work well; for each test bed it produces solutions deviating less than 1.09 percent from the lower bound on average. In fact, for test beds TB1, TB2, and TB3 the deviation is less than one percent in each test bed (see Table 3, the column denoted UB_{gap}). This indicates that even though the heuristic is relatively simple, it could be used as a stand–alone heuristic with a known quality of the computed solution. In fact, in 86 of the 193 instances tested here, the local branching heuristic found the optimal solution. It should be mentioned, however, that in six instances of TB4 the heuristic failed to find a feasible solution (see Section 4.5 for an elaboration of the results on TB4).

Regarding the percentage of the time spent in the three phases, one should note that for the smaller instances of TB1 and TB2 most of the time is spent in the first two phases, while in TB3 and TB4 the majority of the time is spent in Phase 3. The obvious reason is that both the cutting plane algorithm and the local branching heuristic performs better on the small problems of TB1 and TB2 leaving only a small gap to close in Phase 3. Also, the MIPs that need to be solved in Phase 3 become much larger and consequently harder to solve in TB3 and TB4 compared to TB1 and TB2.

Comparing the running times obtained using `TP-ALG` to the ones obtained using `Y-CS`, we observe that in all four test beds the average running time of `TP-ALG` is just a small fraction of that obtained by `Y-CS`. In TB1 and TB2, `Y-CS` runs faster than our algorithm in one out of 57 and in seven out of 71 instances, respectively. `Y-CS` usually only outperforms `TP-ALG` in cases where the cutting plane algorithm is capable of closing the integrality gap, thus eliminating the need for evoking phases 2 and 3. This is most likely due to the Fenchel cuts generated by Yang et al. (2012) being deeper than those generated here. Even though these deeper cuts come at a price in terms of longer computation times, this does not make that large an impact on these small instances, and the gap can be closed faster. Considering TB3 and TB4, the difference between the two algorithms becomes more apparent. On average, the `Y-CS` uses about 85 and almost 45 times more computation time compared to `TP-ALG` on the instances in TB3 and TB4, respectively. Thus, the improved cut-and-solve algorithm `TP-ALG` achieves a very significant decrease in the running times.

We ascribe the large differences in running times to five primary improvements: 1) the way in which cutting planes are generated and the relaxation is strengthened; 2) the place where the cutting planes are added in the cut-and-solve tree; 3) the definition of the piercing cuts; 4) the pruning of sparse problems before an optimal solution has been established; and 5) the strong upper bounds generated in Phase 2. In Phase 1 of the `TP-ALG` algorithm, we generate the Fenchel cuts in a different way than is done by Yang et al. (2012); First of all, we only consider the fractional support of the LP solution in the separation problem and we use Theorem 1 to exclude the y-variables from the separation problems. This significantly reduces the size of the linear master problem in Algorithm 3. In addition to this, we solve the separation problem for the Fenchel inequalities using column generation

instead of the traditional row generation procedure, which reduces the size of the basic matrix of the linear master problem. We also noticed that the main bottleneck of the cut–and–solve algorithm was the solution of the sparse problems and not the number of cut–and–solve nodes created. For that reason, we focused on reducing the effort for solving the sparse problems by means of additional cutting planes, instead of adding cuts to the dense problem in order to increase the lower bound. Furthermore, the way we defined the sparse problems generally results in smaller sparse problems compared to those generated by `Y-CS`. Yang et al. (2012) exploit the solution from the dense problem to guide a heuristic to generate a feasible solution, which then is used for defining the sparse problem. In this way, they can use the proof of termination given in Climer and Zhang (2006) as each sparse problem contains at least one feasible solution. We, however, use Proposition 1 and do this way not need each sparse problem to contain a feasible solution. Furthermore, we use the observation that if a lower bound of the sparse problem exceeds or equals the best known solution, no improving solution can be found in that cut–and–solve node, and we therefore stop the optimization of the sparse problem. This saves a significant amount of CPU time. Finally, as the sparse problems defined by `TP-ALG` are small, it is often hard to find a feasible solution to these problems. Without a good upper bound, solving the sparse problems would often be impossible as none of the branching nodes can be pruned before a feasible solution is known. The good upper bounds generated in Phase 2 are thus very helpful for solving the sparse problems. Actually, in the 86 cases where the local branching heuristic produced the optimal solution, no sparse problems needed to be solved to optimality due to the pruning rule.

4.5 Elaborated results on TB4

As mentioned previously, TB4 consists of problems where an extra "production cost" is added to the distance matrix. Three different problem sizes are considered, namely 50×100, 50×200 and 60×400. For each problem size, three sets of five instances showing a capacity to demand ratio of 2, 3, and 5 were generated. The results are presented in Table 4.

For these instances an extremely small gap between the lower bound and the optimal solution is observed; in none of the cases does the lower bound deviate more than one third of a percent. This is a very convincing result as the assignment costs have been scaled down in order to make the percentage-error measure more independent from the magnitude of the cost data. The two level local branching does, however, show some weaknesses. In five of the 45 instances, the heuristic is not able to generate a feasible solution, and for five other instances, a deviation of more than two percent from the lower bound is observed. Both the failure of finding a feasible solution and the large deviations are encountered in problems with a small capacity to demand ratio ($r = 2$ and 3). The main reason is that we use a general MIP-solver to find an initial solution, and when the capacity to demand ratio is small, a branching tree of a certain depth is often needed to find a feasible solution. On average, however, the local branching heuristic performs well showing an average deviation between lower bound and heuristic upper bound over the entire bed of about one percent only.

Table 4: Results on test bed TB4.

			LB		UB			%-time in phase				
ID	size	r	$\underline{Z}$	LB_{gap}	$\overline{Z}$	UB_{gap}	Z^*	1	2	3	CPU	ρ
n1	50 × 100	2	18,291.3	0.01	18,319	0.15	18,294	2.2	4.8	93.0	616.74	0.88
n2			19,684.3	0.06	20,590	4.40	19,688	0.8	4.5	94.7	3705.52	5.32
n3			19,073.2	0.01	—	—	19,075	7.2	43.8	49.0	192.32	4.34
n4			18,618.7	0.01	18,620	0.01	18,620	16.0	77.3	6.8	92.10	18.72
n5			18,498.0	0.02	18,871	1.98	18,502	2.5	1.7	95.8	788.01	1.40
n6		3	16,942.6	0.03	17,215	1.61	16,948	0.2	0.2	99.6	3887.94	2.87
n7			15,061.7	0.01	15,065	0.02	15,063	29.8	54.9	15.3	32.37	8.91
n8			15,103.5	0.02	15,372	1.78	15,107	2.8	16.9	80.3	170.56	7.87
n9			14,344.1	0.02	14,452	0.75	14,347	1.8	3.8	94.4	89.11	9.69
n10			14,808.8	0.03	14,912	0.70	14,813	1.1	12.3	86.6	476.28	2.34
n11		5	12,071.0	0.01	12,078	0.06	12,072	60.0	11.6	28.4	10.36	63.09
n12			11,893.9	0.03	12,014	1.01	11,898	7.1	5.6	87.3	46.46	11.77
n13			11,124.5	0.00	11,125	0.00	11,125	47.7	23.3	29.0	8.58	127.41
n14			11,815.8	0.01	12,000	1.56	11,817	69.6	15.8	14.6	10.05	36.67
n15			11,486.3	0.02	11,622	1.18	11,489	41.6	19.4	39.0	8.68	60.68
n16	50 × 200	2	25,989.1	0.01	27,272	4.94	25,992	2.6	8.9	88.5	2267.62	22.05
n17			25,866.5	0.01	26,132	1.03	25,868	22.5	17.8	59.7	122.87	9.72
n18			26,928.2	0.01	—	—	26,930	6.4	9.2	84.4	869.55	18.33
n19			25,950.4	0.11	26,120	0.65	25,980	0.1	0.2	99.7	14,790.36	3.38
n20			25,323.7	0.26	—	—	25,390	0.0	0.4	99.6	21,308.44	2.35
n21		3	20,697.0	0.02	20,787	0.43	20,701	1.3	0.8	97.9	1397.86	5.48
n22			22,018.8	0.01	—	—	22,021	0.1	0.4	99.5	19,588.60	2.55
n23			20,036.8	0.01	20,038	0.01	20,038	36.0	58.5	5.5	7.42	23.74
n24			20,594.3	0.00	20,941	1.68	20,595	15.6	14.3	70.1	28.08	8.44
n25			21,167.3	0.00	21,168	0.00	21,168	10.0	90.0	0.0	87.55	11.00
n26		5	16,658.7	0.00	16,659	0.00	16,659	69.8	30.2	0.0	11.54	42.88
n27			16,136.7	0.01	16,140	0.02	16,138	75.0	8.6	16.4	18.92	105.77
n28			17,754.6	0.00	17,804	0.28	17,755	70.1	7.9	22.0	15.37	124.45
n29			15,855.7	0.01	15,914	0.37	15,858	56.0	15.8	28.2	12.08	145.81
n30			16,883.7	0.00	16,884	0.00	16,884	97.3	2.7	0.0	13.68	82.68
n31	60 × 300	2	34,858.5	—	34,861	0.01	—	0.1	0.2	99.7	50,000	1
n32			36,543.5	—	36,742	0.54	—	0.1	0.3	99.6	50,000	1
n33			34,876.2	—	34,884	0,02	—	0.1	0.1	99.8	50,000	1
n34			34,817.6	—	36,057	3.27	—	0.1	0.3	99.6	50,000	1
n35			37,136.3	0.16	38,603	3.80	37,196	0.0	1.5	98.5	14,694.31	3.40
n36		3	27,901.2	0.01	—	—	27,903	0.6	2.4	97.0	3,492.47	0.58
n37			27,593.5	0.00	27,729	0.49	27,594	30.6	36.7	32.7	83,20	44.14
n38			29,229.5	0.01	31,101	6.02	29,231	5.6	52.3	42.1	393,35	5.02
n39			27,437.6	0.01	27,440	0.01	27,439	20.5	66.6	12.9	62.34	13.58
n40			28,030.3	0.01	28,190	0.57	28,033	4.8	15.4	79.8	288.35	38.10
n41		5	21,043.2	0.01	21,045	0.01	21,045	71.5	22.5	6.0	30.57	174.09
n42			22,587.6	0.01	22,924	1.47	22,589	46.0	7.3	46.7	29.51	203.69
n43			21,447.5	0.01	21,822	1.72	21,449	47.0	49.0	4.0	87.32	102.83
n44			21,464.6	0.01	21,466	0.01	21,466	72.8	21.8	5.4	34.17	79.57
n45			21,859.0	0.00	21,879	0.09	21,860	66.9	12.8	20.3	58.97	131.73
			Avg.	0.02		1.17		24.9	18.9	56.2	644.9	39.36

In four of the 45 instances (instances n31-n34), the algorithm failed to find the optimal solution within the time limit. The tendency is that the smaller the r-value, the more computation time is required for solving the instance. If the capacity to demand ratio gets smaller, more facilities are needed to cover the demand, implying that the size of the sparse problems increases. It seems we have reached the limit of the problem size for the case $r = 2$ solvable by our algorithm. Note that the larger instances solved to optimality in TB3 have a significantly larger ratio r than the unsolved instances in TB4. It is worth noting that our way of generating the allocation costs c_{ij} differs from the way traditionally used in existing literature, but this seems to have almost no impact on the performance and thus our algorithm proves robust in this respect.

Comparing `TP-ALG` to `Y-CS`, we see that in only two out of 45 cases (instances n1 and n36) our algorithm does not perform as well as that of Yang et al. (2012). In the four unsolved instances, the time limit of 50,000 seconds was reached for both `TP-ALG` and `Y-CS`. One should note that as the ratio r increases, so does the value of ρ, meaning that `TP-ALG` becomes more efficient relative to `Y-CS` as the cut–and–solve algorithm in Phase 3 becomes more relevant. This underlines the results obtained in Section 4.3, namely that the improved cut–and–solve algorithm is indeed very fast. Over the entire test bed, `TP-ALG` runs about 40 times faster than `Y-CS` on average, suggesting an efficient solution procedure.

5 Conclusions

We have proposed an improved cut–and–solve algorithm for solving the single–source capacitated facility location problem to optimality. The proposed algorithm works in three phases: The first phase consists of a cutting plane algorithm based on knapsack separation used to strengthen the SSCFLP. Secondly, strong upper bounds are generated in Phase 2 by a local branching heuristic that starts by heuristically fixing location variables and then improves the allocation of customers. Finally, in Phase 3, an accelerated cut–and–solve algorithm is proposed to search for an optimal solution if one has not been found in the first two phases.

The computational results show that the lower bounds produced by the cutting plane algorithm in Phase 1 combined with the partial integrality strategy produces very strong lower bounds in relatively short computation times. Furthermore, the characterization of the facets of the integer hull of the capacity constraints facilitates a more efficient generation of the strong Fenchel cutting planes.

When we combine the strong lower bound with the simple LB heuristic, we get a provably very good solution in most cases. Although the heuristic is very simple by nature, combined with Phase 1 it could in fact work as a stand alone heuristic.

The accelerated cut–and–solve algorithm in Phase 3 also proved to be efficient in searching for an optimal solution. The lower bound of the dense problem increases fast and only a few time consuming sparse problems need to be solved. Adding cuts in the root node of every sparse problem did also contribute very positively to the performance of our algorithm.

Compared to the algorithm presented in Yang et al. (2012) our algorithm solves the problems significantly faster in almost all instances. Our new algorithm far outperforms their algorithm with running times 10 to 80 times faster, on average. As the algorithm by Yang et al. (2012) is considered state–of–the–art, this suggests a very efficient algorithm.

Directions for further research include the examination of the effect of generating cutting planes, not only for the knapsack-like structures, but also for substructures including the demand constraints. Furthermore, the proposed framework seems appropriate for other types of location problems as well, for example hierarchical location models with capacity constraints. Finally, the model for the capacitated facility location problems with modular distribution costs proposed in Correia, Gouveia, and Saldanha-da Gama (2010) exhibits many of the same features as the SSCFLP, and therefore an adaptation of the algorithm presented here might serve as a good solution method.

Acknowledgment

The authors are grateful to Mr. Zhen Yang for providing the code, enabling us to compare the two algorithms, and to Professor Kim Allan Andersen for insightful comments and suggestions.

References

Ahuja, R., Orlin, J., Pallottino, S., Scaparra, M., Scutellà, M., 2004. A multi-exchange heuristic for the single-source capacitated facility location problem. Management Science 50 (6), 749–760.

Avella, P., Boccia, M., Salerno, S., 2011. A computational study of dicut reformulation for the single source capacitated facility location problem. Studia Informatica Universalis 9 (3), 21–42.

Balas, E., Zemel, E., 1978. Facets of the Knapsack polytope from minimal covers. SIAM Journal on Applied Mathematics 34 (1), 119–148.

Barceló, J., Casanovas, J., 1984. A heuristic Lagrangean algorithm for the capacitated plant location problem. European Journal of Operational Research 15 (2), 212–226.

Barceló, J., Fernández, E., Jörnsten, K., 1991. Computational results from a new lagrangean relaxation algorithm for the capacitated plant location problem. European Journal of Operational Research 53 (1), 38–45.

Barceló, J., Hallefjord, Å., Fernández, E., Jörnsten, K., 1990. Lagrangean relaxation and constraint generation procedures for capacitated plant location problems with single sourcing. Operations-Research-Spektrum 12 (2), 79–88.

Bitran, G., Chandru, V., Sempolinski, D., Shapiro, J., 1981. Inverse optimization: An application to the capacitated plant location problem. Management Science 27 (10), 1120 – 1141.

Boccia, M., Sforza, A., Sterle, C., Vasilyev, I., Mar. 2008. A cut and branch approach for the capacitated p-median problem based on Fenchel cutting planes. Journal of Mathematical Modelling and Algorithms 7 (1), 43–58.

Boyd, E. A., 1993. Generating Fenchel cutting planes for Knapsack polyhedra. Siam Journal on Optimization 3 (4), 734–750.

Ceselli, A., Righini, G., 2005. A branch-and-price algorithm for the capacitated p-median problem. Networks 45 (3), 125–142.

Chen, C., Ting, C., 2008. Combining Lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. Transportation Research Part E: Logistics and Transportation Review 44 (6), 1099 – 1122.

Climer, S., Zhang, W., 2006. Cut-and-solve: An iterative search strategy for combinatorial optimization problems. Artificial Intelligence 170 (8), 714–738.

Contreras, I., Díaz, J., 2008. Scatter search for the single source capacitated facility location problem. Annals of Operations Research 157 (1), 73–89.

Cornuejols, G., Sridharan, R., Thizy, J., 1991. A comparison of heuristics and relaxations for the capacitated plant location problem. European Journal of Operational Research 50 (3), 280–297.

Correia, I., Gouveia, L., Saldanha-da Gama, L., 2010. Discretized formulations for capacitated location problems with modular distribution costs. European Journal of Operational Research 204 (2), 237 – 244.

Díaz, J., Fernández, E., 2002. A branch-and-price algorithm for the single source capacitated plant location problem. Journal of the Operational Research Society 53, 728–740.

Fischetti, M., Lodi, A., 2003. Local branching. Mathematical Programming 98 (1-3), 23–47.

Fischetti, M., Polo, C., Scantamburlo, M., 2004. A local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. Networks 44 (2), 61–72.

Gadegaard, S., Klose, A., Nielsen, L., 2016. A "cut–and–solve" solver for the single source capacitated facility location problem. GitHub, source code (v1.2.0).
URL `https://github.com/SuneGadegaard/SSCFLPsolver`

Geoffrion, A., 1974. Lagrangean relaxation for integer programming. In: Approaches to Integer Programming. Vol. 2 of Mathematical Programming Studies. Springer Berlin Heidelberg, pp. 82–114.

Gu, Z., Nemhauser, G., Savelsbergh, M., 1998. Lifted Cover Inequalities for 0-1 Integer Programs: Computation. INFORMS Journal on Computing 10 (4), 427–437.

Holmberg, K., Rönnqvist, M., Yuan, D., 1999. An exact algorithm for the capacitated facility location problems with single sourcing. European Journal of Operational Research 113 (3), 544–559.

Kaparis, K., Letchford, A., 2010. Separation algorithms for 0-1 knapsack polytopes. Mathematical Programming 124 (1-2), 69–91.

Karp, R., 1972. Reducibility among combinatorial problems. In: Miller, R., Thatcher, J., Bohlinger, J. (Eds.), Complexity of Computer Computations. The IBM Research Symposia Series. Springer US, pp. 85–103.

Klincewicz, J., Luss, H., 1986. A Lagrangian relaxation heuristic for capacitated facility location with single-source constraints. Journal of the Operational Research Society 37, 495–500.

Klose, A., Görtz, S., 2007. A branch-and-price algorithm for the capacitated facility location problem. European Journal of Operational Research 179 (3), 1109–1125.

Martello, S., Pisinger, D., Toth, P., 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. Management Science 45 (3), 414–424.

Neebe, A., Rao, M., 1983. An algorithm for the fixed-charge assigning users to sources problem. Journal of the Operational Research Society 34, 1107–1113.

Nemhauser, G., Wolsey, L., 1988. Integer and Combinatorial Optimization. Wiley New York.

Pirkul, H., 1987. Efficient algorithms for the capacitated concentrator location problem. Computers & Operations Research 14 (3), 197 – 208.

Rönnqvist, M., Tragantalerngsak, S., Holt, J., 1999. A repeated matching heuristic for the single-source capacitated facility location problem. European Journal of Operational Research 116 (1), 51–68.

Sridharan, R., 1993. A Lagrangian heuristic for the capacitated plant location problem with single source constraints. European Journal of Operational Research 66 (3), 305 – 312.

Weismantel, R., 1997. On the 0/1 knapsack polytope. Mathematical Programming 77 (3), 49–68.

Yang, Z., Chu, F., Chen, H., 2012. A cut-and-solve based algorithm for the single-source capacitated facility location problem. European Journal of Operational Research 221 (3), 521–532.