DEPARTMENT OF OPERATIONS RESEARCH
UNIVERSITY OF AARHUS

Working Paper no. 2009 / 1

2009 / 03 / 25

A subgradient-based branch-and-bound algorithm for the capacitated facility location problem

Simon Görtz and Andreas Klose

ISSN 1600-8987

Department of Mathematical Sciences
Telephone: +45 8942 1111
E-mail: institut@imf.au.dk

Building 1530, Ny Munkegade 118
DK-8000 Aarhus C, Denmark
URL: www.imf.au.dk

A subgradient-based branch-and-bound algorithm for the capacitated facility location problem

Simon Görtz [a]

[a] *Faculty of Economics – Schumpeter School of Business and Economics, University of Wuppertal, 42119 Wuppertal, Germany*

Andreas Klose [b]

[b] *Department of Mathematical Sciences, University of Aarhus, Ny Munkegade, Building 1530, 8000 Aarhus C, Denmark*

Abstract

This paper presents a simple branch-and-bound method based on Lagrangean relaxation and subgradient optimization for solving large instances of the capacitated facility location problem (CFLP) to optimality. In order to guess a primal solution to the Lagrangean dual, we average solutions to the Lagrangean subproblem. Branching decisions are then based on this estimated (fractional) primal solution. Extensive numerical results reveal that the method is much more faster and robust than other state-of-the-art methods for solving the CFLP exactly.

Key words: Mixed-integer programming; Lagrangean relaxation; Capacitated Facility Location; Subgradient optimization; Volume algorithm; Branch-and-bound

1 Introduction

The *capacitated facility location problem* (CFLP) is a well-known combinatorial optimization problem with a number of applications in the area of distribution and production planning. It consists in deciding which facilities to open from a given set J of potential facility locations and how to assign customers $i \in I$ to those facilities. The objective is to minimize total fixed and shipping costs. Constraints are that each customer's demand $d_i \geq 0$ must be satisfied and that each plant cannot supply more than its capacity $s_j > 0$ if it is open. Denoting the cost of supplying all of

Email addresses: simon.goertz@wiwi.uni-wuppertal.de (Simon Görtz), aklose@imf.au.dk (Andreas Klose).

customer i's demand from facility j by c_{ij} and the fixed cost of operating facility j by f_j, the CFLP is usually written as the mixed-integer program

$$Z = \min \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{j \in J} f_j y_j \tag{1}$$

$$\text{s.t.} \sum_{j \in J} x_{ij} = 1\,, \quad i \in I\,, \tag{2}$$

$$\sum_{i \in I} d_i x_{ij} \leq s_j y_j\,, \quad j \in J\,, \tag{3}$$

$$\sum_{j \in J} s_j y_j \geq d(I) := \sum_{i \in I} d_i\,, \tag{4}$$

$$x_{ij} - y_j \leq 0\,, \quad i \in I\,,\ j \in J\,, \tag{5}$$

$$0 \leq x_{ij} \leq 1\,,\ 0 \leq y_j \leq 1\,, \quad i \in I\,,\ j \in J\,, \tag{6}$$

$$y_j \in \{0, 1\}\,, \quad j \in J\,. \tag{7}$$

A usual way of obtaining lower bounds on Z is to relax the demand constraints (2) in a Lagrangean manner and to apply subgradient optimization for approximately computing the resulting Lagrangean dual bound. Several Lagrangean heuristics and branch-and-bound algorithms for the CFLP follow this approach (Shetty, 1990; Cornuejols *et al.*, 1991; Ryu and Guignard, 1992; Sridharan, 1993). Admittedly, subgradient optimization shows the drawback of not providing a (fractional) primal solution to the Lagrangean dual. Deciding on which variable to branch gets therefore cumbersome and ad hoc heuristic rules are mostly used to this end. If the branch-and-bound method is then additionally based on a depth-first search, the resulting implementation usually fails in solving larger problem instances.

An optimal primal solution to the Lagrangean dual is a convex combination of optimal solutions to the Lagrangean subproblem at given optimal Lagrangean multiplier values. Simply counting how many times a binary variable y_j equals one in the solutions obtained for the Lagrangean subproblem should thus probably give sufficient information on optimal solutions to the linear primal master problem, that is the Lagrangean dual's dual program. In a similar way the Volume Algorithm (Barahona and Anbil, 2000) approximates primal solutions within the framework of subgradient optimization. Barahona and Chudak (2005) use this method in conjunction with randomized rounding for computing heuristic solutions to the CFLP and UFLP. Section 2 discusses our method for computing lower bounds and estimating a corresponding (fractional) primal solution. Section 3 then summarizes how this bounding scheme is embedded in a branch-and-bound procedure for computing optimal solutions to the CFLP, and Sect. 4 presents extensive numerical results as well as a comparison with other state-of-the-art methods for exactly solving the CFLP. Finally, some conclusions are drawn in Sect. 5.

2 Computation of a lower bound

Dualizing the demand constraints (2) with multipliers λ_i, $i \in I$, yields the Lagrangean subproblem

$$
\begin{aligned}
Z_D(\lambda) = \sum_{i\in I} \lambda_i + \min_{x,y} \sum_{i\in I} \sum_{j\in J} (c_{ij} - \lambda_i) x_{ij} + \sum_{j\in J} f_j y_j \\
\text{s.t.: } (3),\ (4),\ (5),\ (6),\ (7).
\end{aligned}
\tag{8}
$$

Let $j_i \in \arg\min\{c_{ij} : j \in J\}$. It is straightforward to show that optimal Lagrangean multipliers can be found in the interval $[\lambda^{\min}, \lambda^{\max}]$, where

$$
\lambda_i^{\min} = \min\Big\{c_{ij} : j \in J \setminus \{j_i\}\Big\} \text{ and } \lambda_i^{\max} = \max\Big\{c_{ij} : j \in J\Big\}\,. \tag{9}
$$

Moreover, it is well-known that (8) reduces to the binary knapsack problem

$$
\lambda_0 = \min_y \Big\{ \sum_{j\in J} (f_j - v_j) y_j : \sum_{j\in J} s_j y_j \geq d(I)\,,\ y_j \in \{0,1\}\ \forall\, j \in J \Big\}\,, \tag{10}
$$

where

$$
v_j = \max_x \Big\{ \sum_{i\in I} (\lambda_i - c_{ij}) x_{ij} : \sum_{i\in I} d_i x_{ij} \leq s_j\,,\ 0 \leq x_{ij} \leq 1\ \forall\, i \in I \Big\}\,, j \in J\,.
$$

The Lagrangean function $Z_D(\lambda) = \lambda_0 + \sum_{i\in I} \lambda_i$ and thus also the Lagrangean dual bound

$$
Z_D = \max\Big\{ Z_D(\lambda) : \lambda \in [\lambda^{\min}, \lambda^{\max}] \Big\} \tag{11}
$$

can therefore be computed in pseudo-polynomial time (Cornuejols *et al.*, 1991).

A broad range of different methods is available for exactly solving the Lagrangean dual (11) and obtaining a corresponding primal solution. A stabilized column generation method for solving the corresponding primal linear master problem is applied in (Klose and Drexl, 2005) and extended to a branch-and-price algorithm for the CFLP in (Klose and Görtz, 2007). Alternatively, regularized decomposition or bundle methods (Lemaréchal, 1989; Ruszczyński, 1995; Ruszczyński and Świętanowski, 1997) may be used for solving (11). The main principle of a bundle method is to keep an inner polyhedral approximation to the ϵ-subdifferential $\partial_\epsilon Z_D(\lambda)$ of the piecewise linear and concave function $Z_D(\lambda)$. A trial step is then taken into the best direction found in this set (which requires to solve a quadratic master program). If this gives a sufficient increase in $Z_D(\lambda)$, a "serious step" is taken to the next iterate; otherwise a "null step" is performed and the current approximation of $\partial_\epsilon Z_D(\lambda)$ improved by adding further subgradients. The Volume Algorithm of Barahona and Anbil (2000) can be seen as a heuristic version of a bundle method. The procedure keeps an estimate of a primal solution to the Lagrangean dual. This estimated solution is an exponentially weighted average of the solutions obtained to the Lagrangean subproblem in the course of the algorithm. Furthermore, this estimated primal solution is also used for determining subgradient-like search directions. If a move into such a direction gives an ascent, a serious step is taken and the new dual iterate accepted;

otherwise a "null step" is performed. With respect to the Lagrangean dual (11), the Volume Algorithm consists in the application of the following steps.

Step 1: Initialize the Lagrangean multipliers λ with initial values $\overline{\lambda}$. Solve the Lagrangean subproblem (8) with $\lambda = \overline{\lambda}$ and let $(\overline{x}, \overline{y})$ denote the solution. Set $t = 1$ and $\overline{Z}_D = Z_D(\overline{\lambda})$.

Step 2: Set $g_i^t = 1 - \sum_{j \in J} \overline{x}_{ij}$ for all $i \in I$ and $\lambda^t = \overline{\lambda} + \theta_t g^t$, where $\theta_t > 0$ is the step length. Solve (8) with $\lambda = \lambda^t$. Let (x^t, y^t) denote the solution. Set $(\overline{x}, \overline{y}) := \mu(x^t, y^t) + (1 - \mu)(\overline{x}, \overline{y})$, where $0 < \mu < 1$.

Step 3: If $Z_D(\lambda^t) > \overline{Z}_D$, update $\overline{\lambda} = \lambda^t$ and set $\overline{Z}_D = Z_D(\lambda^t)$. Let $t := t + 1$ and return to Step 2 if a termination criteria is not met.

In Step 2, the usual step size formula

$$\theta_t = \alpha_t \frac{UB - Z_D(\lambda^t)}{\|g^t\|^2} \tag{12}$$

is applied, where UB is an upper bound on Z_D and $0 < \alpha_t \leq 2$. Barahona and Anbil (2000) propose a special updating scheme for the step size parameter α_t. There is no guarantee that the method converges, neither to an optimal primal nor to an optimal dual solution. Bahiense *et al.* (2002) show how to modify the Volume Algorithm such that convergence is guaranteed. This revised version is however very close to bundle methods.

In order to restore at least dual convergence, it seems appropriate to resort to standard subgradient optimization. Instead of the exponentially weighted average $(\overline{x}, \overline{y})$ used in the Volume Algorithm above, we might then also just simply use the arithmetic average

$$(\tilde{x}, \tilde{y}) = \frac{1}{t} \sum_{\tau=1}^{t} (x^\tau, \, y^\tau) \tag{13}$$

for the purpose of estimating primal solutions. When the dual multipliers λ^t converge to the optimal ones, say λ^*, the solutions (x^t, y^t) of the Lagrangean subproblem tend to be very close to those that are optimal at optimal multipliers λ^*. Hence, despite its simplicity, the arithmetic average should be relatively close to a suitable convex combination of optimal solutions to the Lagrangean relaxation for optimal multipliers λ^*, and thus quite well approximate an optimal solution to the linear primal master problem. Moreover, in case of the CFLP, the bound Z_D on Z is generally quite strong. It is thus usually not required and also does not pay off to include additional (polyhedral) cutting planes. A fractional primal solution is then only used for the purpose of selecting a suitable branching variable – optionally, the solution may also be used for obtaining further heuristic solutions. To this end, however, exact knowledge of an optimal primal solution should not be required and rough information sufficient for finding a suitable branching variable, e.g., by simply branching on a variable y_j showing a value of $\tilde{y}_j$ closest to 0.5. Sherali and Choi (1996) also give some theoretical support for the simple average weighting rule (13). They propose to recover primal solutions by taking convex combinations of the solutions (x^t, y^t) generated in the course of the subgradient optimization. They moreover provide sufficient conditions on the dual step sizes and primal weights for ensuring that the primal and dual iterates converge to optimal solutions. It is in particular shown that (13) converges to an optimal primal solution if $\theta_t \geq \theta_{t+1} > 0$ $\forall\, t$ and $\lim_{t\to\infty} t\theta_t = \infty$. Although the step size strategy (12) will probably not meet these conditions, it can be expected that this way a sufficiently precise estimate is generated, in particular if the number of binary variables is small compared to the total number of variables as in the case of the CFLP.

Sherali and Choi (1996) also consider the case of deflected subgradient optimization. We have experimented with a deflected subgradient strategy and also with exponentially smoothed subgradients as suggested in Baker and Sheasby (1999); but our computational experiments indicated that in case of the CFLP these enhanced strategies do not show any advantage over the simple standard subgradient strategy, in particular not for larger problem instances.

3 Branch-and-bound procedure

The lower bounding procedure described in the preceding section is used within a branch-and-bound algorithm for computing optimal solutions to the CFLP. The method uses a best lower bound search strategy and is based on the following components.

- The lower bound Z_D is computed approximately by means of subgradient optimization. At the root node, the Lagrangean multipliers are initialized by setting $\lambda = \lambda^{\min}$, where $\lambda^{\min}$ is defined in (9). At all other nodes of the enumeration tree, the multipliers are initialized with the values obtained at the father node. The number of subgradient steps is limited to 350 at the root node and to 30 at the subsequent nodes. At each node, the step size α_t in the step size formula (12) is first set to 2 and halved if there is no improve in the (local) lower bound after 5 subsequent subgradient steps.
- Upper bounds UB and feasible solutions are computed by solving the transportation problem that results by opening facilities $j \in J$ with $y_j^t = 1$. At the root node, a feasible solution is obtained after each subgradient step; at each other node, the transportation problem is solved only one time after completion of the subgradient procedure using the Lagrangean solution y^t obtained at the best multipliers found in the course of the subgradient method. In order to avoid that the same transportation problem is repeatedly solved, the different sets of open facilities generated so far are stored in a hash table of at most 1009 hash values based on a solution's total fixed cost.
- The variable y_j showing a "relative frequency" $\tilde{y}_j$ closest to 0.5 is selected as the branching variable. Ties are broken arbitrarily. The current node is then replaced by two sons where y_j is forced to be 0 and 1, respectively.
- A best lower bound search is used, that is, the unprocessed node of the enumeration tree showing the smallest lower bound is processed next. Ties are broken arbitrarily.
- In order to further reduce the problem, Lagrangean probing is applied whenever the best feasible solution found during the search improves. Let y^t be the current Lagrangean solution. The binary variable y_j is tentatively set to $1 - y_j^t$ and the Lagrangean bound recomputed. If the resulting bound is not less than the best

global upper bound computed so far, the variable y_j can be pegged to the value y_j^t. At the root node the Lagrangean probing is performed after each subgradient step leading to an improved feasible solution. At each other node of the enumeration tree, the probing is only executed one time after completion of the subgradient procedure.

4 Computational results

The proposed branch-and-bound algorithm (BB-SG) was coded in C and compiled using the Gnu C compiler version 3.3.5 on an IBM PC with 3 GB RAM, an Intel PD 930 processor of 3 GHz and a Linux operating system. The COMBO algorithm of Martello *et al.* (1999) and David Pisinger's C code (`http://www.diku.dk/~pisinger`) was used to solve the binary knapsack problems (10). The transportation problems that need to be solved to obtain the upper bounds were solved by calling the network simplex algorithm of CPLEX's callable library, version 8.0 (ILOG, 2002).

The code was used to solve four different sets of test problems. The first set of test problems comprises the 75 problem instances from Klose and Görtz (2007). These test problems range in size from instances with 100 customers and facilities up to instances involving 500 customers and 200 potential facility sites. The instances were generated using the following procedure proposed by Cornuejols *et al.* (1991):

(1) Customer and facility sites are generated as uniformly randomly distributed points in a unit square. The unit transportation cost c_{ij}/d_i are then obtained as the Euclidian distance multiplied by 10.

(2) Demands d_i and capacities s_j are respectively generated from $U[5, 35]$ and $U[10, 160]$, where $U[a.b]$ denotes the uniformly distributed random number from $[a, b)$. A facility's fixed cost reflects economies of scale and is obtained from $f_j = U[0, 90] + U[100, 110]\sqrt{s_j}$.

(3) The capacities s_j are then rescaled such that a prefixed capacity ratio $r = \sum_j s_j / \sum_i d_i$ results. In Klose and Görtz (2007) and also here, a ratio of $r = 3$, 5 and 10 is used.

The second set of 100 test instances are the ones used by Avella and Boccia (2007) and available at the web page `http://www.ing.unisannio.it/boccia/CFLP.htm`. The sizes of these problem instances range from 300 up to 1000 customer and facility sites. According to Avella and Boccia (2007), these instances are also generated by the procedure described above. A closer look at the data reveals, however, that for all these problem instances the range of the fixed facility cost is much larger than usual for problem instances generated with the procedure of Cornuejols et al. In case of the problem instances with $|I| > |J|$, the unit transportation costs are, moreover, ten times higher than it should be according to the Cornuejols et al. procedure. We thus used the same problem sizes and capacity ratios r as Avella and Boccia (2007) to generate a third test bed of problem instances based on the procedure described above. The appendix lists the C-code used for creating these test problem instances. Finally, the fourth set of test instances is taken from the OR-library (Beasley, 1990).

We compared the results obtained with the subgradient-based branch-and-bound method (BB-SG) with the following other exact solution procedures:

(1) the branch-and-price algorithm of Klose and Görtz (2007);
(2) Ryu's and Guignard's (1992) CAPLOC algorithm;
(3) CPLEX's MIP solver, version 8.0;
(4) the branch-and-cut procedure of Avella and Boccia (2007);
(5) the same branch-and-bound procedure as BB-SG, where however the volume algorithm is applied instead of subgradient optimization (BB-VA).

4.1 Comparison with the branch-and-price algorithm of Klose and Görtz (2007)

Table 1 compares the subgradient-based method with the branch-and-price algorithm (B&P) on the test problem instances used in Klose and Görtz (2007). Each row of the table shows the average results over 5 single problem instances of the given problem size $|I| \times |J|$ and capacity ratio $r = \sum_j s_j / \sum_i d_i$. For both methods the table indicates the number of enumerated nodes (nodes), the maximum depth of the enumeration tree reached during the search (depth) and the computation time in CPU seconds (time). For reasons of this comparison, the branch-and-price algorithm was recoded in C and compiled and run on the same machine as BB-SG.

Both algorithms are based on the same Lagrangean relaxation. Whilst the branch-and-price methods computes the lower bound and in particular a corresponding (fractional) primal solution exactly by means of stabilized column generation, the subgradient-based method just gets a lower approximate to this lower bound and an estimate of a corresponding (fractional) primal solution. Accordingly, the branch-and-price method enumerates much less nodes than BB-SG. Although significant, the increase in the number of nodes enumerated by BB-SG is relatively moderate, when compared to the branch-and-price method. This indicates that estimating a primal solution by averaging the Lagrangean solutions works pretty well– at least in case of the CFLP and the relaxation under consideration. BB-SG hardly ever needed to go deeper in the enumeration tree than the branch-and-price method did. BB-SG requires however much less effort for enumerating a single node than the branch-and-price method does. The net effect is that BB-SG highly outperforms the branch-and-price procedure. On average, BB-SG was about 20 times faster than B&P; in case of larger instances and a capacity ratio of $r = 10$, the subgradient-based method in particular showed to be much more efficient. Table 2 summarizes the comparison by additionally averaging results over different values of r.

4.2 Comparison with CAPLOC

Ryu's and Guignard's (1992) CAPLOC algorithm is based on the same Lagrangean relaxation as BB-SG and also applies subgradient optimization for computing the Lagrangean dual bound. CAPLOC is, however, a depth-first search and uses quite different branching rules. Before branching at the top node, CAPLOC tries to fix as many y_j variables as possible by means of an extensive Lagrangean probing. Let (x^B, y^B) denote the best feasible solution CAPLOC found so far and let z_B denote its objective value. Variables y_j are temporarily fixed to $1 - y_j^B$ and a limited number of subgradient steps is applied. If the resulting lower bound is no smaller than z_B, the binary variable y_j is pegged to y_j^B. If further branching is required, the procedure always branches on a variable y_j with smallest value of

Table 1
Comparison of B&P and BB-SG

	B&P			BB-SG		
$\|I\| \times \|J\|$	nodes	depth	time	nodes	depth	time
			$r = 3$			
100×100	9	3	0.50	9	3	0.43
200×100	53	9	6.86	58	7	1.29
200×200	29	6	4.68	39	6	2.18
500×100	537	15	722.30	794	14	37.94
500×200	693	17	745.05	678	16	54.80
max	1347	24	1785.33	1733	19	91.79
mean	264	10	295.88	316	9	19.33
			$r = 5$			
100×100	14	4	0.98	15	4	0.44
200×100	89	10	22.80	133	10	3.07
200×200	53	6	12.47	49	6	2.63
500×100	256	11	915.66	437	13	21.73
500×200	3070	21	6376.10	5061	17	434.98
max	8605	25	17860.59	12419	21	1319.37
mean	696	10	1465.60	1139	10	92.57
			$r = 10$			
100×100	6	2	0.52	13	4	0.28
200×100	26	5	13.53	37	5	1.36
200×200	21	6	5.47	54	8	2.16
500×100	37	6	242.93	83	7	6.45
500×200	610	12	3458.33	1312	13	85.32
max	2099	18	10872.92	4495	18	230.96
mean	140	6	744.16	300	7	19.11

Table 2
Summarized comparison of B&P and BB-SG

	B&P			BB-SG		
$\|I\| \times \|J\|$	nodes	depth	time	nodes	depth	time
100×100	10	3	0.67	12	4	0.38
200×100	56	8	14.4	76	7	1.91
200×200	34	6	7.54	47	7	2.32
500×100	277	11	626.96	438	11	22.04
500×200	1458	17	3526.49	2350	15	191.70
mean	367	9	835.21	585	9	43.67

$(\nu_j + f_j)/s_j$, where

$$\nu_j = \min_x \Big\{ \sum_{i \in I} c_{ij} x_{ij} : \sum_{i \in I} d_i x_{ij} = s_j \, , \; 0 \leq x_{ij} \leq 1 \; \forall \, i \in I \Big\} \, .$$

The branch that results from requiring the selected plant to be open is then always investigated first.

We used a FORTRAN code of Ryu and Guignard, but replaced the out-of-kilter method used in this code for solving the transportation problems by calls to the network simplex algorithm of CPLEX's callable library. The code was then compiled and run on the same machine as BB-SG. Table 3 compares CAPLOC and BB-SG on the test problem instances from Klose and Görtz (2007) and Table 4 summarizes these results by again averaging over the capacity ratio r. As can be seen from Table 3 and 4, BB-SG is far superior to CAPLOC. (The number of enumerated nodes reported for CAPLOC in this table does not include the branches investigated in the Lagrangean probing.) On the test problems of Table 3, BB-SG was on average about 55 times faster than CAPLOC. Basically, both algorithms are based on the same methodology; they apply the same Lagrangean relaxation and use subgradient optimization for lower bounding. BB-SG however greatly benefits from, firstly, the best lower bound search and, secondly and mostly, from better branching decisions based on a reasonable estimate of a primal solution.

Since also the test problem instances from the OR-library are of a size and difficulty that can be managed by CAPLOC, we also compared both methods on these test problems. Each of these problem instances is of size $|I| \times |J| = 1000 \times 100$. Table 5 shows the results, which again indicate the superiority of BB-SG. The ORLIB instances are easier to solve than the instances of Table 3, but still BB-SG is about 17 times faster than CAPLOC on these instances.

4.3 Comparison with CPLEX and the branch-and-cut method of Avella and Boccia (2007)

We used CPLEX's MIP solver in the following way for solving the CFLP (1)–(7). We started with the weaker aggregate formulation that is obtained by taking for each $j \in J$ the sum of constraints (5) over all $i \in I$. Additional variable upper bounds (5) were then included "on the fly" whenever violated by the current LP solution. The model formulation found this way was then passed to the MIP solver, after removing beforehand those added constraints (5) that showed a positive slack at the final LP solution. We also passed a feasible solution and upper bound to CPLEX's MIP solver. The solution was obtained using a simple rounding procedure, which was applied each time the current LP relaxation was solved. To this end, the facilities $j \in J$ are sorted according to non-decreasing LP value of the associated variable y_j and then opened as long as $y_j \geq 0.9$ or the capacity of the open facilities is smaller than the total demand. CPLEX's MIP solver was called with default option values except that (i) CPLEX's internal primal heuristic was switched off; (ii) the relative optimality tolerance was set to 10^{-6}; (iii) the parameters CutsFactor, CutPass and AggCutLim respectively controlling the number of cuts CPLEX may add, the number of cutting rounds CPLEX may perform, and the number of constraints that can

Table 3
Comparison of CAPLOC and BB-SG

	CAPLOC		BB-SG		
$\vert I\vert \times \vert J\vert$	nodes	time	nodes	depth	time
			$r = 3$		
100×100	11	0.36	9	3	0.43
200×100	293	5.36	58	7	1.29
200×200	586	12.12	39	6	2.18
500×100	8424	402.48	794	14	37.94
500×200	22470	978.93	678	16	54.80
max	54679	2773.29	1733	19	91.79
mean	6357	279.85	316	9	19.33
			$r = 5$		
100×100	67	0.84	15	4	0.44
200×100	810	13.61	133	10	3.07
200×200	365	10.33	49	6	2.63
500×100	13427	591.76	437	13	21.73
500×200	250202	29314.56	5061	17	434.98
max	850016	119911.52	12419	21	1319.37
mean	52974	5986.22	1139	10	92.57
			$r = 10$		
100×100	13	0.34	13	4	0.28
200×100	504	12.31	37	5	1.36
200×200	76	3.63	54	8	2.16
500×100	5193	298.16	83	7	6.45
500×200	65110	4973.16	1312	13	85.32
max	265228	21419.38	4495	18	230.96
mean	14179	1057.52	300	7	19.11

Table 4
Summarized comparison of CAPLOC and BB-SG

	CAPLOC		BB-SG		
$\vert I\vert \times \vert J\vert$	nodes	time	nodes	depth	time
100×100	30	0.51	12	4	0.38
200×100	536	10.43	76	7	1.91
200×200	342	8.69	47	7	2.32
500×100	9015	430.80	438	11	22.04
500×200	112594	11755.55	2350	15	191.70
mean	24503	2441.20	585	9	43.67

Table 5
Comparison of CAPLOC and BB-SG on ORLIB instances

	CAPLOC		BB-SG		
problem	nodes	time	nodes	depth	time
capa1	30	26.55	9	3	2.91
capa2	8	23.84	7	2	2.89
capa3	7	26.86	9	3	2.18
capa4	1	21.22	1	0	0.84
capb1	1	10.12	1	0	1.66
capb2	1510	361.92	27	5	11.06
capb3	280	243.44	29	6	11.49
capb4	4	25.12	17	7	4.32
capc1	95	61.25	9	3	3.40
capc2	569	164.91	59	8	12.63
capc3	22	52.19	11	4	6.01
capc4	18	51.87	5	2	2.40
mean	212	89.11	15	4	5.15

be aggregated for deriving flow cover inequalities and mixed-integer rounding cuts were considerably increased over the default values.

Table 6 compares this way of using CPLEX with BB-SG on the instances of Klose and Görtz (2007) and Table 7 summarizes again these results. As these tables show, BB-SG also outperformed CPLEX on these instances. On average, BB-SG was about 16 times faster than CPLEX in solving these test problem instances, and in no single case, CPLEX showed to be faster.

Table 8 compares the application of CPLEX and BB-SG on the instances from the OR library. Avella and Boccia (2007) also report on the application of their branch-and-cut algorithm (in the sequel denoted by B&C) as well as CPLEX's MIP solver to these instances. They used their branch-and-cut method and CPLEX 8.1 on a Pentium IV with 1.7 GHz and 512 MB RAM. In Table 8, we repeat the computation times they reported for CPLEX 8.1 and their own branch-and-cut method; we however divided these times by 2, since the computer they used might be (at most) up to two times slower than the one we used. On average, BB-SG showed to be about 75 times faster than the way we used CPLEX 8.0 and 113 times faster than the computation times reported by Avella and Boccia (2007) for CPLEX 8.1. BB-SG also outperformed Avella's and Boccia's branch-and-cut method and showed, on average, to be about 40 times faster in solving the ORLIB instances.

We then also applied BB-SG and CPLEX to the test problem instances of Avella and Boccia (2007) and compared the computation times to those Avella and Boccia report for their B&C method. Avella and Boccia generated five instances for each problem size and capacity ratio r. In Table 9 averages over these five instances are taken, and Table 10 additionally averages over r in order to further summarize the results. It has to be noted that the method of Avella and Boccia failed to solve two instances of size 1000×1000 and ratio $r = 15$ to optimality within the time limit of

Table 6
Comparison of CPLEX and BB-SG on instances of Klose & Görtz (2007)

	CPLEX		BB-SG		
$\lvert I\rvert \times \lvert J\rvert$	nodes	time	nodes	depth	time
		$r = 3$			
100×100	231	1.76	9	3	0.43
200×100	814	10.40	58	7	1.29
200×200	2333	46.82	39	6	2.18
500×100	4764	290.96	794	14	37.94
500×200	8103	567.57	678	16	54.80
max	15982	1209.54	1733	19	91.79
mean	3249	183.5	316	9	19.33
		$r = 5$			
100×100	522	4.04	15	4	0.44
200×100	1114	27.62	133	10	3.07
200×200	3998	71.44	49	6	2.63
500×100	2952	291.90	437	13	21.73
500×200	42969	4529.83	5061	17	434.98
max	101633	11449.88	12419	21	1319.37
mean	10311	984.97	1139	10	92.57
		$r = 10$			
100×100	111	2.58	13	4	0.28
200×100	423	26.58	37	5	1.36
200×200	2952	62.24	54	8	2.16
500×100	188	108.23	83	7	6.45
500×200	13456	4747.06	1312	13	85.32
max	32445	11402.14	4495	18	230.96
mean	3426	989.34	300	7	19.11

Table 7
Summarized comparison CPLEX and BB-SG on instances of Klose & Görtz (2007)

	CPLEX		BB-SG		
$\lvert I\rvert \times \lvert J\rvert$	nodes	time	nodes	depth	time
100×100	288	2.79	12	4	0.38
200×100	784	21.53	76	7	1.91
200×200	3094	60.17	47	7	2.32
500×100	2635	230.36	438	11	22.04
500×200	21509	3281.49	2350	15	191.70
mean	5662	719.27	585	9	43.67

Table 8
Comparison of CPLEX, Avella's & Boccia's B&C and BB-SG on ORLIB instances

	CPLEX[a]		CPLEX 8.1[b]	B&C[b]		BB-SG		
problem	nodes	time	time	nodes	time	nodes	depth	time
capa1	8	364.58	151.11	608	159.63	9	3	2.91
capa2	0	391.56	92.26	452	82.02	7	2	2.89
capa3	28	741.18	66.74	129	95.29	9	3	2.18
capa4	0	251.97	18.40	2	47.91	1	0	0.84
capb1	0	106.12	147.86	622	128.42	1	0	1.66
capb2	82	598.70	597.59	853	293.78	27	5	11.06
capb3	134	498.35	2845.63	5121	743.83	29	6	11.49
capb4	40	573.28	857.31	5684	288.65	17	7	4.32
capc1	7	288.43	114.08	368	84.88	9	3	3.40
capc2	282	493.59	1987.18	1048	449.21	59	8	12.63
capc3	15	214.75	116.52	345	84.73	11	4	6.01
capc4	0	160.28	27.49	48	36.41	5	2	2.40
mean	50	390.23	585.18	1273	207.90	15	4	5.15

[a] Zero number of nodes means that the rounding heuristic solution's value equalled the LP lower bound computed before actually calling the MIP solver.
[b] Computation times as reported by Avella and Boccia (2007) divided by 2.

100,000 CPU seconds they used in their experiments (which approximately corresponds to at most 50,000 CPU seconds on our machine). Our application of CPLEX solved these two instances within 14,670 and 26,290 CPU seconds, respectively. The BB-SG code even just required 5052 and 6779 CPU seconds for solving these two instances. On the other hand, BB-SG failed to solve one instance of size 1000×1000 and capacity ratio $r = 5$. The computations stopped after 33,617 CPU seconds due to insufficient memory with a remaining gap of 0.04% between the global lower and upper bound. Avella and Boccia solved this instance within 38,264 seconds, which is about 19,132 seconds on our machine; and our application of CPLEX even required only 2904 seconds. In Table 9, we take in case of the Avella and Boccia method (column B&C) the average only over the three remaining instances of size 1000×1000 and ratio $r = 15$ that were solved to optimality by Avella and Boccia. Accordingly, in column BB-SG, we only average the results obtained on the four instances of size 1000×1000 and ratio $r = 5$ successfully solved by this method. Table 10 shows that BB-SG is, on average, about 3 times faster than B&C in solving the instances of Avella and Boccia; in particular in case of $r > 5$ BB-SG proved to be much faster. In only a few single cases B&C required less computation time than BB-SG. BB-SG was also, on average, significantly faster than CPLEX; except in the case of $r = 5$, where CPLEX was fastest. CPLEX did also better than the method of Avella and Boccia, however, on almost all of the 100 instances, except in case of five instances with capacity ratio $r = 20$.

In case of the instances of Avella and Boccia, the fixed facility cost are in a range of 50 to 1450 with a mean of about 580. Instances generated with Cornuejols

Table 9
Comparison of CPLEX, B&C and BB-SG on Avella's & Boccia's instances

	CPLEX		B&C[a,b]		BB-SG[c]		
$\|I\| \times \|J\|$	nodes	time	nodes	time	nodes	depth	time
				$r = 5$			
300×300	599	31.09	436	294.24	657	16	19.00
500×500	1746	232.34	1258	1549.32	2867	22	209.33
700×700	2544	509.82	1696	4578.12	28709	32	3717.48
1000×1000	8909	3820.74	4123	18722.79	46893	37	15635.69
1500×300	205	69.76	32	836.33	547	13	235.71
max	18958	8681.39	6314	32099.24	104561	52	35682.33
mean	2801	932.75	1509	5196.16	15935	24	3963.44
				$r = 10$			
300×300	760	46.21	131	201.31	535	14	12.53
500×500	1830	303.00	476	912.32	2431	19	120.56
700×700	4673	1166.34	808	5532.38	8729	27	899.63
1000×1000	19614	9677.84	3370	30432.92	26267	32	5096.53
1500×300	22	42.61	9	426.49	19	4	27.45
max	38706	17405.73	5862	44959.06	63333	36	12526.19
mean	5380	2247.20	959	7501.08	7596	19	1231.34
				$r = 15$			
300×300	251	24.91	52	89.24	239	11	4.82
500×500	588	141.05	65	334.72	358	17	18.69
700×700	3847	1571.06	356	2495.46	8423	25	674.53
1000×1000	27033	17944.19	2070	32987.13	47956	34	7323.29
1500×300	8	35.93	2	204.33	9	2	23.25
max	43295	26289.96	2334	40759.87	96421	37	15802.47
mean	6345	3943.43	509	7222.18	11397	18	1608.92
				$r = 20$			
300×300	139	21.06	18	71.21	60	7	1.75
500×500	614	145.47	47	240.77	264	12	13.13
700×700	1075	557.57	40	533.13	221	12	22.83
1000×1000	3800	4034.44	354	4757.19	2362	21	328.71
1500×300	4	33.70	2	146.99	3	1	14.82
max	10267	10882.98	986	14646.77	4783	25	707.57
mean	1126	958.45	92	1149.86	582	11	76.25

[a] Averaged computation times reported by Avella and Boccia (2007) divided by 2.
[b] Two unsolved instances of size 1000×1000 and $r = 15$ not included in the average.
[c] One unsolved instance of size 1000×1000 and $r = 5$ not included in the average.

Table 10
Summarized comparison of CPLEX, B&C, BB-SG on Avella's & Boccia's instances

	CPLEX		B&C[a,b]		BB-SG[c]		
$\|I\| \times \|J\|$	nodes	time	nodes	time	nodes	depth	time
300×300	437	30.82	159	164.00	373	12	9.53
500×500	1195	205.47	462	759.28	1480	18	90.43
700×700	3035	951.20	725	3284.78	11521	24	1328.62
1000×1000	14839	8869.30	2479	21725.01	30870	31	7096.06
1500×300	60	45.50	11	403.54	145	5	75.31
mean	3913	2020.46	767	5267.32	8878	18	1719.99

[a] Averaged computation times reported by Avella and Boccia (2007) divided by 2.
[b] Two unsolved instances of size 1000×1000 and $r = 15$ not included in the average.
[c] One unsolved instance of size 1000×1000 and $r = 5$ not included in the average.

et al.'s procedure should however usually show fixed facility cost of approximately 350 to 1400 with a mean of about 1000. Due to the smaller average fixed cost, more facilities should be open in optimal solutions to these instances, which in tendency should contribute to less restrictive capacity constraints. In particular the instances with $r = 20$ were easy to solve. The instances of size $|I| \times |J| = 1500 \times 300$ show, moreover, unit transportation cost of about 50, which is ten times larger than usual for problem instances generated with Cornuejols et al.'s procedure. This explains why the instances of this size were quite easy to solve. We therefore again applied the Cornuejols et al. procedure to generate new test problem instances of the same size and capacity ratio r as the ones used by Avella and Boccia. Additionally, we respectively generated for each ratio $r \in \{5, 10, 15, 20\}$ five problem instances of size $|I| \times |J| = 1500 \times 600$. This time, we were not able to solve all of these 120 problem instances to optimality by means of CPLEX and BB-SG. Some of the computations terminated due to insufficient memory; other computations were stopped after reaching a time limit of 100,000 CPU seconds. Table 11 compares the performance of CPLEX and BB-SG in solving these problem instances. Since not all instances could be solved to optimality, the table shows averages taken only over those instances that could be solved. The column headed "unsolved" shows how many of the five instances per given size and ratio r were not solved to optimality. Detailed results for every single problem instance are listed in the appendix. We also did not apply CPLEX to the instances of size 1500×300 and 1500×600; the results on the other instances already show the strong superiority of BB-SG over CPLEX. Our application of CPLEX did not succeed in solving any of the instances of size 1000×1000 within the available time and memory limitations. Only four out of the twenty instances of size 700×700 could be solved to optimality. Also in case of eight out of twenty instances of size 500×500, the computations needed to be stopped, since the size of the enumeration tree took all of the available main memory; one single instance even stopped due to a segmentation fault after about 31,000 seconds of computations. BB-SG solved all instances of size 500×500, failed on only two instances of size 700×700, and succeeded in solving nine of the twenty instances of size 1000×1000. All instances of size 1500×300 and also fourteen of

Table 11
Comparison of CPLEX and BB-SG on new the problem instances

size	BB-SG				CPLEX		
$\lvert I\rvert \times \lvert J\rvert$	nodes	depth	time	unsolved	nodes	time	unsolved
				$r = 5$			
300×300	2906	16	141.01	0	17709	1223.64	0
500×500	19975	24	2126.33	0	76111	13909.53	1
700×700	108496	29	21630.22	0	442184	82444.34	4
1000×1000	98489	25	36900.07	3	–	–	5
1500×300	21564	27	5826.42	0		not tried	
1500×600	69406	38	32865.19	3		not tried	
				$r = 10$			
300×300	711	16	46.64	0	11943	1407.59	0
500×500	10147	23	1367.05	0	26966	11732.39	2
700×700	70396	28	13871.00	2	–	–	5
1000×1000	58135	30	22323.44	3	–	–	5
1500×300	9511	24	2653.70	0		not tried	
1500×600	94607	31	40941.48	3		not tried	
				$r = 15$			
300×300	607	11	37.10	0	6399	1730.46	0
500×500	5235	18	613.16	0	12538	11556.94	3
700×700	22841	23	3583.20	0	43385	61078.82	4
1000×1000	127615	31	49895.25	3	–	–	5
1500×300	3286	18	844.57	0		not tried	
1500×600	26393	26	9191.11	0		not tried	
				$r = 20$			
300×300	217	11	15.58	0	5664	1577.20	0
500×500	4803	19	469.61	0	28751	25509.65	2
700×700	7917	20	1316.71	0	33863	57957.03	3
1000×1000	21066	25	12018.90	2	–	–	5
1500×300	8387	23	3280.89	0		not tried	
1500×600	11949	22	9412.24	0		not tried	

the twenty instances of size 1500×600 were solved by BB-SG within the available memory and time limitations. No single instance that could not be solved by BB-SG was successfully solved by CPLEX, and on no successfully solved single problem instance CPLEX was faster. On those instances tested and successfully solved by CPLEX, CPLEX required, on average, 17 times more computation time than BB-SG did. For a single instance of size 500×500 and ratio $r = 5$, BB-SG was only 3 times faster; for one instance of size 500×500 and ratio $r = 10$, BB-SG did even 360 times faster than CPLEX.

4.4 *Using the volume algorithm instead of subgradient optimization*

We finally also compared BB-SG with the same branch-and-bound method, where however instead of subgradient optimization the volume algorithm is used for computing lower bounds (BB-VA). In contrast to BB-SG, the volume algorithm estimates primal solutions by taking an exponentially smoothed average of the generated Lagrangean solutions. The guessed primal solution is also used to determine a direction into which the method searches for improved dual solutions, whilst BB-SG simply makes a step in direction of a subgradient. In our experiments, we used a smoothing parameter $\mu = 0.1$. We also experimented with taking simple arithmetic averages, but this showed to be far less effective. Barahona and Anbil (2000) and Barahona and Chudak (2005) suggested to choose $\mu = \max\Big\{\mu_{\max}/10, \min\{\mu^*, \mu_{\max}\}\Big\}$, where μ^* minimizes $\|\mu s^t + (1-\mu)g^t\|$, s^t and g^t respectively denotes the current subgradient and direction, and $\mu_{\max}$ is initially set to 0.1 and halved if $Z_D(\lambda)$ is not increased by 1 % in 100 consecutive iterations. Table 12 compares the performance of the two methods on the test problem instances from Klose and Görtz (2007). Table 13 again summarizes these results by additionally taking averages over the different ratios r. BB-SG significantly outperformed the method based on the volume algorithm. On average, BB-SG was 4 to 5 times faster than BB-VA. The clear superiority of BB-SG suggests that the method will still do better than BB-VA even if the smoothing parameter μ is determined in a more sophisticated way. We therefore also refrained from extending the comparison to the other types of test problems. In our computational experiments, we in particular observed that the method based on the volume algorithm showed at times week dual convergence behavior and difficulties to get close enough to the optimal Lagrangean dual solution. In such cases, the branch-and-bound method went deep down the enumeration tree and enumerated too much nodes. In contrast to BB-SG, the method based on the volume algorithm also needs to average the solutions x^t and thus usually required more computational effort per node than BB-SG.

5 Conclusions

In this paper we proposed a simple subgradient-based branch-and-bound algorithm for solving the CFLP exactly. The method's main idea is to average solutions obtained for the Lagrangean subproblem in order to estimate fractional primal solutions to the Lagrangean dual and to base branching decisions on the estimated primal solution. If combined with a best-lower bound search strategy, the method consistently showed to significantly outperform other existing state-of-the-art methods for solving the CFLP exactly. The method is capable to solve difficult instances of the CFLP with up to 1000 customer and 1000 potential facility sites as well as 1500 customer and 600 potential facility sites. The method also showed to be rather robust, in the sense that it worked fine on sets of different test problem instances showing different problem characteristics. In addition, the method has the advantage of being relatively simple and relatively easy to implement. We think that the efficiency of the method can primarily attributed to the following points: (i) Subgradient optimization works very well for getting very close to the Lagrangean

Table 12
Comparison of BB-SG and BB-VA on the instances from Klose and Görtz (2007)

	BB-SG			BB-VA		
$\lvert I\rvert \times \lvert J\rvert$	nodes	depth	time	nodes	depth	time
			$r = 3$			
100×100	9	3	0.43	91	42	0.62
200×100	58	7	1.29	523	60	25.51
200×200	39	6	2.18	31	8	3.96
500×100	794	14	37.94	1321	58	259.42
500×200	678	16	54.80	1282	108	417.11
max	1733	19	91.79	3485	200	1035.95
mean	316	9	19.33	650	55	141.32
			$r = 5$			
100×100	15	4	0.44	136	32	1.78
200×100	133	10	3.07	227	35	10.69
200×200	49	6	2.63	232	39	10.40
500×100	437	13	21.73	497	44	95.84
500×200	5061	17	434.98	4490	92	1598.30
max	12419	21	1319.37	9459	128	4055.70
mean	1139	10	92.57	1116	48	343.40
			$r = 10$			
100×100	13	4	0.28	84	33	0.49
200×100	37	5	1.36	73	18	3.06
200×200	54	8	2.16	727	134	13.43
500×100	83	7	6.45	73	10	21.23
500×200	1312	13	85.32	1594	95	520.52
max	4495	18	230.96	5181	200	1675.48
mean	300	7	19.11	510	58	111.75

Table 13
Summarized comparison of BB-SG and BB-VA

	BB-SG			BB-VA		
$\lvert I\rvert \times \lvert J\rvert$	nodes	depth	time	nodes	depth	time
100×100	12	4	0.38	104	36	0.96
200×100	76	7	1.91	274	38	13.09
200×200	47	7	2.32	330	60	9.26
500×100	438	11	22.04	630	37	125.50
500×200	2350	15	191.70	2455	98	845.31
mean	585	9	43.67	759	54	198.82

dual bound resulting from relaxing demand constraints in the CFLP. (ii) This Lagrangean dual bound is generally a sharp lower bound on the optimal objective function value of the CFLP. (iii) The number of binary variables is only a small percentage of the total number of variables. The good results obtained for the CFLP with this subgradient-based approach and the simple averaging of Lagrangean solutions for primal solution recovery suggests that similar results may possibly also be obtainable for mixed-integer programming problems, which only show a relatively small number of integer variables and where subgradient optimization works well in providing sharp lower bounds.

References

Avella, P. and Boccia, M. (2007). A cutting plane algorithm for the capacitated facility location problem. *Computational Optimization and Applications*. Published online 6. December 2007. Doi: 0.1007/s10589-007-9125-x.

Bahiense, L., Maculan, N., and Sagastizabal, C. (2002). The volume algorithm revisited: relation with bundle methods. *Mathematical Programming*, **94**, 41–69.

Baker, B. M. and Sheasby, J. (1999). Accelerating the convergence of subgradient optimization. *European Journal of Operational Research*, **117**, 136–144.

Barahona, F. and Anbil, R. (2000). The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, **87**, 385–399.

Barahona, F. and Chudak, F. (2005). Near-optimal solutions to large-scale facility location problems. *Discrete Optimization*, **2**, 35–50.

Beasley, J. E. (1990). OR-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, **41**(11), 1069–1072.

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

ILOG (2002). CPLEX 8.0. User's manual.

Klose, A. and Drexl, A. (2005). Lower bounds for the capacitated facility location problem based on column generation. *Management Science*, **51**(11), 1689–1705.

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

Lemaréchal, C. (1989). Nondifferentiable optimization. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*, pages 529–572, Amsterdam. North-Holland.

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

Ruszczyński, A. (1995). On the regularized decomposition method for stochastic programming problems. In K. Marti and P. Kall, editors, *Stochastic Programming: Numerical Techniques and Engineering Applications*, volume 423 of *Lecture Notes in Control and Information Sciences*, pages 93–108. Springer-Verlag, Berlin, Heidelberg, New York.

Ruszczyński, A. and Świętanowski, A. (1997). Accelerating the regularized decomposition method for two stage stochastic linear problems. *European Journal of Operational Research*, **101**(2), 328–342.

Ryu, C. and Guignard, M. (1992). An efficient algorithm for the capacitated plant location problem. Working Paper 92-11-02, Decision Sciences Department, University of Pennsylvania, The Wharton School.

Sherali, H. D. and Choi, G. (1996). Recovery of primal solutions when using subgradient optimization methods to solve Lagrangian duals of linear programs. *Operations Research Letters*, **19**(3), 105–113.

Shetty, B. (1990). Approximate solutions to large scale capacitated facility location problems. *Applied Mathematics and Computation*, **39**, 159–175.

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

A Detailed results obtained with BB-SG on instances from Klose and Görtz (2007)

Tables A.1–A.3 show the detailed results obtained on the test problem instances from Klose and Görtz (2007). In these tables, Z denotes the optimal objective value.

Table A.1
Detailed results on instances with ratio $r = 3$

no.	Z	BB-SG			CPLEX	
		nodes	depth	time	nodes	time
			$\lvert I\rvert \times \lvert J\rvert = 100 \times 100$			
1	28345.99	7	3	0.56	9	0.54
2	29580.17	7	3	0.34	679	3.65
3	27062.23	7	3	0.21	200	2.02
4	28988.34	17	4	0.54	109	0.97
5	25279.40	9	3	0.52	160	1.61
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 100$			
1	29740.15	41	7	1.33	1293	12.00
2	31509.51	13	3	0.64	56	2.49
3	29135.00	47	8	1.28	282	6.39
4	29910.45	139	9	2.28	2108	25.64
5	29923.01	49	8	0.93	330	5.49
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 200$			
1	52824.22	23	5	1.10	223	6.46
2	52148.07	15	7	1.60	2092	34.32
3	52810.44	89	9	4.41	7148	134.09
4	50434.02	11	3	1.24	1232	37.31
5	52643.74	55	6	2.54	968	21.93
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 100$			
1	36629.27	207	13	11.11	2116	146.27
2	36145.85	1733	19	68.13	8891	531.23
3	36070.42	1565	15	85.42	10794	544.32
4	37976.41	399	11	19.65	1162	71.36
5	36445.01	67	10	5.37	859	161.64
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 200$			
1	58992.74	149	14	9.27	1651	109.15
2	59295.73	1079	16	83.03	11726	1209.54
3	63551.98	859	17	91.79	9653	446.80
4	56691.81	213	15	11.50	1505	91.00
5	60315.26	1091	17	78.41	15982	981.37

Table A.2
Detailed results on instances with ratio $r = 5$

no.	Z	BB-SG			CPLEX	
		nodes	depth	time	nodes	time
			$\lvert I\rvert \times \lvert J\rvert = 100 \times 100$			
1	17489.90	17	5	0.46	272	2.38
2	18329.44	3	1	0.21	89	1.49
3	17118.53	15	5	0.47	354	3.47
4	18082.94	29	5	0.69	1551	10.04
5	17949.61	13	5	0.35	344	2.83
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 100$			
1	19677.03	33	6	1.43	766	23.14
2	21288.57	101	9	2.13	1830	28.16
3	19621.73	159	13	4.23	1113	35.97
4	20856.96	85	14	1.92	449	13.16
5	20789.09	285	9	5.63	1412	37.69
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 200$			
1	32586.04	27	4	1.81	1189	24.12
2	32714.25	75	9	2.88	4347	69.87
3	32741.33	21	4	1.20	1046	21.29
4	32542.34	113	9	5.96	13345	235.66
5	33078.76	9	4	1.31	62	6.28
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 100$			
1	27591.52	299	11	16.61	1930	221.17
2	28647.41	871	16	43.52	5269	589.31
3	27587.79	747	15	32.18	5600	368.76
4	27501.23	115	10	9.52	1282	174.42
5	26701.75	153	12	6.81	678	105.83
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 200$			
1	39240.06	977	16	82.84	34464	3936.16
2	40406.43	12419	20	1319.37	101633	11449.88
3	39352.31	563	12	39.55	7776	1013.79
4	38179.74	3279	18	183.21	38746	3569.95
5	40050.19	8067	21	549.94	32227	2679.35

Table A.3
Detailed results on instances with ratio $r = 10$

no.	Z	BB-SG			CPLEX	
		nodes	depth	time	nodes	time
			$\lvert I\rvert \times \lvert J\rvert = 100 \times 100$			
1	9041.94	5	2	0.22	15	1.68
2	9100.71	9	4	0.33	102	1.83
3	10271.16	17	4	0.31	203	4.86
4	9546.92	17	4	0.28	197	2.80
5	9493.98	17	5	0.27	37	1.73
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 100$			
1	13997.38	59	8	2.12	635	34.37
2	14231.66	23	5	0.85	122	21.12
3	13902.67	1	0	0.15	0	5.94
4	14091.49	49	6	1.35	803	42.37
5	14044.54	51	7	2.35	554	29.10
			$\lvert I\rvert \times \lvert J\rvert = 200 \times 200$			
1	18887.23	115	11	3.16	2261	49.08
2	17170.88	23	8	1.78	726	40.29
3	17105.23	17	5	1.11	1179	26.83
4	18410.24	105	11	3.00	3332	72.06
5	17716.32	9	4	1.74	7263	122.93
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 100$			
1	23457.95	173	9	13.17	290	133.72
2	23254.49	13	3	2.32	0	30.75
3	23544.74	125	10	9.27	325	139.38
4	22883.94	61	8	3.41	229	89.38
5	22489.53	43	7	4.07	97	147.94
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 200$			
1	26633.61	213	10	28.77	1441	655.63
2	27356.97	271	11	40.45	795	417.88
3	26762.47	411	13	34.43	1677	740.06
4	26967.45	1169	15	92.01	32445	10519.60
5	27158.26	4495	18	230.96	30922	11402.14

B Detailed results obtained with BB-SG on the instances from Avella and Boccia (2007)

Table B.1
Detailed results on instances with ratio $r = 5$

no.	Z	BB-SG			B&C[a]	CPLEX	
		nodes	depth	time	time	nodes	time
			$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$				
1	16350.66	145	11	4.81	199.28	265	35.21
2	15948.45	751	14	22.59	407.56	1010	43.98
3	15474.85	471	13	13.34	132.23	161	9.23
4	17989.98	735	23	20.52	357.42	797	36.88
5	18037.62	1181	19	33.73	374.73	761	30.15
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$				
1	26412.42	3655	31	269.30	640.59	1477	182.72
2	28130.74	4371	24	299.07	2033.39	3209	343.43
3	27904.52	1191	15	92.51	1280.69	624	89.09
4	28159.03	3511	20	284.63	1342.48	1273	175.52
5	24702.77	1609	19	101.12	2449.44	2149	370.96
			$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$				
1	36905.93	83325	52	9773.50	4923.98	973	225.24
2	34311.72	11687	28	1831.99	6798.11	4336	786.61
3	34294.64	4179	21	625.62	994.28	1732	404.81
4	38090.91	7597	28	1087.33	3097.86	2768	569.14
5	37802.11	36759	31	5268.96	7076.38	2909	563.30
			$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$				
1	49509.82	42719	47	14563.75	26025.86	10135	4112.86
2	50688.10	104561	44	35682.33	32099.24	18958	8681.39
3	47202.64	16709	30	5150.33	13930.51	2573	1016.97
4	48868.55	59559		33616.95[b]	19132.00	6954	2904.15
5	50743.54	23581	28	7146.35	2426.33	5927	2388.31
			$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$				
1	154750.44	175	10	100.10	1167.32	89	53.98
2	159256.55	1243	14	468.87	745.08	139	61.31
3	157011.46	297	16	129.87	1043.22[c]	71	58.37
4	157406.34	857	16	372.22	814.04	611	116.43
5	160946.21	163	11	107.48	412.01	113	58.72

[a] Computation times reported in Avella and Boccia (2007) divided by 2.
[b] Computations stopped due to insufficient memory. Remaining gap = 0.04 %.
[c] Non-optimal solution value reported in Avella and Boccia (2007).

Table B.2
Detailed results on instances with ratio $r = 10$

no.	Z	BB-SG			B&C[1]	CPLEX	
		nodes	depth	time	time	nodes	time
			$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$				
6	11251.20	371	18	10.40	226.29	1252	77.61
7	11392.53	1179	19	25.85	298.13	1215	54.65
8	11377.34	45	8	1.27	58.10	177	15.75
9	10878.05	1003	16	22.60	364.20	1097	74.95
10	11232.78	77	10	2.55	59.83	57	8.08
			$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$				
6	15756.82	1083	21	55.21	370.92	291	62.97
7	16109.29	743	21	38.27	1076.99	908	173.64
8	16041.73	4415	22	225.50	1582.58	1209	211.04
9	16327.72	5693	20	270.16	984.27	6294	968.71
10	15815.13	223	13	13.66	546.85	446	98.66
			$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$				
6	19910.68	15929	27	1624.39	4453.21	8341	2379.27
7	21297.30	7807	18	793.86	4126.32	4886	1163.08
8	20659.96	2651	36	271.84	1664.20	1020	289.16
9	20979.89	8591	29	829.95	8284.52	4170	923.62
10	22055.42	8667	26	978.09	9133.66	4949	1076.57
			$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$				
6	27823.85	12169	32	2464.26	27110.03	12075	6854.47
7	27252.33	36659	36	6927.12	29982.27	23986	12971.66
8	27375.38	11739	30	2124.05	44959.06	14898	7408.11
9	26857.09	63333	36	12526.19	31306.72	38706	17405.73
10	27187.00	7435	25	1441.03	18806.51	8405	3749.22
			$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$				
6	156621.18	65	8	57.85	472.36	56	48.42
7	156950.23	3	1	17.59	345.60	4	40.34
8	157687.61	7	3	18.85	353.78	18	40.54
9	156893.90	13	5	16.33	514.28	12	37.42
10	157678.50	7	3	26.63	446.40	22	46.34

[a] Computation times reported in Avella and Boccia (2007) divided by 2.

Table B.3
Detailed results on instances with ratio $r = 15$

no.	Z	BB-SG			B&C[a]	CPLEX	
		nodes	depth	time	time	nodes	time
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$			
11	10023.94	73	9	2.24	43.41	226	18.33
12	9336.64	177	10	2.72	72.72	151	13.11
13	10058.49	821	16	15.16	79.34	686	62.43
14	9699.36	51	9	2.03	179.88	91	15.76
15	9842.17	75	11	1.94	70.85	103	14.94
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$			
11	13437.72	181	16	9.89	122.33	232	103.62
12	14675.03	775	20	39.72	584.99	518	126.23
13	13666.25	177	13	12.15	177.01	553	130.34
14	13580.02	323	19	13.82	616.69	1071	195.80
15	13896.76	333	18	17.87	172.55	568	149.26
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$			
11	17120.16	36179	26	2873.80	5194.97	9225	3850.82
12	18130.43	717	33	68.03	1038.63	1557	665.32
13	17239.97	967	19	86.66	1071.26	1103	485.28
14	17337.63	3189	28	244.02	3491.86	3849	1340.12
15	18145.50	1063	20	100.13	1680.59	3502	1513.74
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$			
11	22180.34	37103	31	5052.26	50222.91[c]	25823	14670.24
12	22160.40	46435	35	6779.15	50202.63[c]	33947	26289.96
13	22648.25	96421	32	15802.47	40759.87[b]	43295	25419.57
14	22313.02	11231	33	1701.02	20911.33	13314	8980.62
15	22627.63	48591	37	7281.53	37290.20[b]	18786	14360.56
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$			
11	149995.75	3	1	16.61	211.40[b]	2	29.65
12	154883.50	35	7	34.49	289.55	29	43.38
13	151593.03	1	0	16.28	74.06	1	39.47
14	151788.86	1	0	13.40	135.27	1	30.33
15	156417.45	7	3	35.45	311.39	9	36.84

[a] Computation times reported in Avella and Boccia (2007) divided by 2.
[b] Non-optimal solution value reported in Avella and Boccia (2007).
[c] Time limit of 100,000 sec. (50,000 sec. on our machine) reached.

Table B.4
Detailed results on instances with ratio $r = 20$

no.	Z	BB-SG			B&C[a]	CPLEX	
		nodes	depth	time	time	nodes	time
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$			
16	9158.76	65	9	1.89	59.41	76	18.94
17	9171.67	103	9	2.13	144.35	211	24.18
18	9553.60	33	7	2.09	50.83	77	24.18
19	9053.71	93	10	1.95	94.05	326	28.64
20	9046.33	5	2	0.69	7.43	6	9.37
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$			
16	12584.49	229	15	10.50	436.20	818	165.68
17	13347.40	569	14	27.67	299.86	636	169.79
18	12831.13	263	10	11.46	193.37	465	119.17
19	13489.62	143	10	7.36	161.94	760	148.87
20	12342.26	117	12	8.67	112.50	389	123.86
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$			
16	16000.04	153	13	14.56	313.81	852	387.52
17	16171.67	193	10	24.56	603.38	615	387.81
18	16414.81	205	12	20.42	667.77	684	550.16
19	16366.79	359	10	37.19	577.59	1021	571.46
20	15434.22	195	13	17.44	503.11	2201	890.88
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$			
16	21331.82	4783	21	707.57	14646.77	4087	4449.72
17	21188.89	305	15	48.82	1747.62	400	604.61
18	20713.43	657	25	111.41	1264.76	780	1098.16
19	20537.45	1317	19	201.27	2427.50	3464	3136.74
20	21560.86	4749	25	574.46	3699.31	10267	10882.98
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$			
16	155489.36	1	0	10.46	146.52	2	31.50
17	156033.33	11	4	24.78	143.59	5	36.87
18	156777.58	1	0	10.71	127.95	4	34.35
19	155946.26	1	0	12.04	158.96	2	31.18
20	156409.40[b]	1	0	16.13	157.92	6	34.60

[a] Computation times reported in Avella and Boccia (2007) divided by 2.
[b] Avella and Boccia (2007) report on this instance a solution value of $156,407.23$. We could however not achieve this value.

C Detailed results obtained with BB-SG on the new instances

Table C.1–C.4 show the results obtained on the newly generated test instances. In these tables, the column headed *UB* is the final upper bound computed by means of the respective method. In case that optimality is not proven and *UB* may thus differ from the optimal solution value, we additionally indicate the percentage gap between the upper bound (*UB*) and the global lower bound (*LB*) computed by the respective method. The percentage gap is defined as

$$\text{gap} = 100\frac{UB - LB}{LB - \sum_{i\in I}\min_{j\in J} c_{ij}}\,.$$

We substract $\sum_{i\in I}\min_{j\in J} c_{ij}$ in the denominator, since otherwise the gap can be made arbitrarily small by adding for each customer $i \in I$ a sufficiently large constant to the transportation costs c_{ij} for each $j \in J$.

Table C.1
Detailed results on instances with ratio $r = 5$

no.	BB-SG					CPLEX			
	UB	nodes	depth	time	gap	*UB*	nodes	time	gap
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$					
1	50638.73	2357	17	91.24		50638.73	27373	931.93	
2	49261.65	617	12	28.35		49261.65	15668	559.36	
3	49879.18	441	12	18.75		49879.18	1597	73.63	
4	50449.52	485	13	23.25		50449.52	22438	1096.29	
5	49721.60	10631	24	543.44		49721.60	21470	3457.00	
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$					
1	78662.68	2211	24	257.05		78662.68	88662	30704.46	
2	82057.96	26691	22	3383.02		82057.96	75000	35475.21	0.03[a]
3	80763.30	15755	27	1440.32		80763.30	59042	6288.02	
4	80988.59	52475	27	5299.12		80988.59	126818	15146.69	
5	78429.26	2741	19	252.13		78429.26	29921	3498.95	
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$					
1	111890.01	39537	34	9172.19		111923.52	70900	50528.57	0.07[a]
2	112581.89	133053	27	21525.98		112581.89	442184	82444.34	
3	112628.49	166285	33	34942.47		112665.00	144500	38714.36	0.07[a]
4	111333.92	141045	30	28952.73		111357.60	155400	35325.08	0.06[a]
5	112124.09	62559	22	13557.74		112399.49	116900	35226.71	0.30[a]
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$					
1	155907.13	59252		76047.67	0.04[a]	155933.90	85500	41650.08	0.06[a]
2	160380.33	58684		52244.50	0.02[a]	160401.46	96500	44810.08	0.04[a]
3	159971.71	147221	24	56346.88		159991.70	98800	45194.62	0.03[a]
4	158189.69	49757	25	17453.25		158194.50	48000	39434.93	0.02[a]
5	159228.79	59160		47446.83	0.02[a]	159270.35	80500	37940.90	0.06[a]
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$					
1	65630.64	43375	35	11487.98					
2	65831.00	15379	25	3798.34					
3	67537.85	29891	25	8136.86					
4	67670.70	5069	28	1662.28					
5	67578.34	14107	24	4046.64					
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 600$					
1	105381.44	45804		57589.38	0.13[a]				
2	106736.07	45530		54212.58	0.05[a]				
3	107296.34	44206		44071.32	0.03[a]				
4	104051.34	87677	37	41896.71					
5	104171.76	51135	39	23833.66					

[a] Terminated due to insufficient memory.

Table C.2
Detailed results on instances with ratio $r = 10$

no.	BB-SG					CPLEX			
	UB	nodes	depth	time	gap	*UB*	nodes	time	gap
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$					
1	28474.02	293	13	16.18		28474.02	16422	2129.03	
2	27008.17	527	14	53.10		27008.17	4302	1889.55	
3	28826.24	1179	16	76.62		28826.24	23969	1465.70	
4	27662.75	641	19	35.41		27662.75	9628	679.58	
5	28682.42	917	18	51.89		28682.42	5395	874.07	
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$					
1	45081.83	3921	22	455.12		45081.83	45037	20855.99	
2	43420.27	273	17	33.40		43420.27	24829	12012.48	
3	44899.66	23257	28	3458.19		44968.09	73300	44505.23	0.40[a]
4	43396.79	22153	23	2794.76		43438.93	52200	48614.37	0.27[a]
5	43511.99	1129	23	93.76		43511.99	11031	2328.70	
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$					
1	59572.55	71615	30	14022.36		59572.55	95800	40789.26	0.08[a]
2	60832.31	44023	23	7468.60		60846.78	60000	56448.26	0.13[a]
3	62120.52	237975	30	100000.42	0.03[b]	62249.18	83400	37008.56	0.37[a]
4	60055.82	95551	31	20122.04		60101.53	64700	72927.47	0.19[a]
5	60920.86	167490		84648.77	0.04[a]	61139.75	62400	30543.04	0.58[a]
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$					
1	85384.46	47515	22	100000.02	0.10[b]	85549.72	61000	64249.75	0.40[a]
2	83054.52	64427	23	100002.41	0.11[b]	83394.50	54600	41384.11	0.61[a]
3	82864.38	59797	27	25859.98		82933.14	57500	47629.46	0.16[a]
4	84331.57	80936		73527.65	0.06[a]	84506.92	48600	32129.46	0.36[a]
5	83394.66	56473	33	18786.89		83394.66	42800	66488.31	0.07[a]
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$					
1	49730.88	20261	30	4433.79					
2	50255.27	7581	19	3458.74					
3	49974.14	8605	22	1970.75					
4	49869.27	5233	24	1466.44					
5	50925.93	5877	23	1938.76					
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 600$					
1	65859.94	152305	34	68850.16					
2	63493.10	36909	27	13032.79					
3	67117.28	61290		57054.54	0.07[a]				
4	65277.42	60512		73016.79	0.14[a]				
5	65151.72	61718		51813.71	0.07[a]				

[a] Terminated due to insufficient memory.
[b] Time limit of 100,000 seconds reached.

Table C.3
Detailed results on instances with ratio $r = 15$

no.	BB-SG					CPLEX			
	UB	nodes	depth	time	gap	*UB*	nodes	time	gap
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$					
1	21253.35	13	5	1.63		21253.35	53	79.32	
2	21814.36	73	8	4.74		21814.36	820	1 45.76	
3	22529.39	1941	17	130.95		22529.39	16858	5382.56	
4	22168.57	91	9	3.91		22168.57	744	154.62	
5	22523.41	917	17	44.26		22523.41	13522	2890.04	
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$					
1	33435.69	1019	21	123.56		33435.69	18186	19391.52	
2	33507.69	329	13	44.14		33507.69	6890	3722.35	
3	33959.00	2277	15	497.95		33959.00	20300	30996.04	0.16[c]
4	33709.19	15603	20	1576.54		33785.23	68400	32433.31	0.42[a]
5	33833.95	6947	23	823.59		34025.29	69100	29448.15	0.81[a]
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$					
1	45104.89	5183	24	1147.89		45104.89	43385	61078.82	
2	45100.44	59121	30	8749.40		45101.74	56500	73651.92	0.17[a]
3	45032.72	34687	26	5503.07		45100.81	51000	60907.68	0.28[a]
4	44992.00	4219	16	852.30		44992.00	75694	100425.30	0.04[b]
5	46197.17	22581	21	6428.29		46451.16	44900	39507.25	0.74[a]
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$					
1	62522.86	71933	30	100000.17	0.14[b]	62605.11	33300	77854.68	0.43[a]
2	62492.39	75877	23	100001.02	0.08[b]	62796.25	31700	44235.52	0.75[a]
3	62025.59	149679	31	56430.57		62186.92	39700	61674.28	0.46[a]
4	62404.53	105551	31	43359.93		62471.88	32900	72169.08	0.23[a]
5	62401.91	218013		100000.04	0.02[b]	62768.69	28400	49560.19	0.81[a]
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$					
1	46379.99	1007	18	391.25					
2	45562.90	13541	22	2995.91					
3	45872.96	1181	17	497.83					
4	46456.43	441	14	199.65					
5	46516.67	259	17	138.23					
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 600$					
1	54285.13	82527	41	26848.46					
2	54459.33	2995	16	1341.99					
3	54273.96	14707	23	4902.58					
4	54688.17	18575	23	6923.03					
5	54626.06	13159	25	5939.51					

[a] Terminated due to insufficient memory.
[b] Time limit of 100,000 seconds reached.
[c] CPLEX terminated with a segmentation fault.

Table C.4
Detailed results on instances with ratio $r = 20$

no.	BB-SG					CPLEX			
	UB	nodes	depth	time	gap	*UB*	nodes	time	gap
				$\lvert I\rvert \times \lvert J\rvert = 300 \times 300$					
1	19519.92	547	14	27.52		19519.92	17175	4700.24	
2	19090.29	261	11	15.32		19090.29	5411	1397.63	
3	19618.22	49	8	8.50		19618.22	1857	465.16	
4	19323.48	115	11	9.28		19323.48	2344	785.32	
5	19287.12	111	9	17.28		19287.12	1531	537.66	
				$\lvert I\rvert \times \lvert J\rvert = 500 \times 500$					
1	29294.42	12015	20	978.32		29417.09	44900	39519.52	0.72[a]
2	28485.57	5711	25	520.09		28485.57	34884	30797.72	
3	29183.51	3687	17	506.92		29400.98	50900	31358.79	1.19[a]
4	29245.49	815	17	119.12		29245.49	8422	9121.40	
5	29128.85	1787	15	223.61		29128.85	42948	36609.82	
				$\lvert I\rvert \times \lvert J\rvert = 700 \times 700$					
1	38426.66	1875	19	297.97		38426.66	24988	55198.34	
2	37863.00	1037	19	200.63		37863.00	54100	93763.46	0.13[a]
3	37794.64	16285	20	2763.86		37805.71	36261	100924.82	0.20[b]
4	38083.00	2485	21	353.92		38083.00	42737	60715.71	
5	37542.15	17903	22	2967.16		37580.17	35072	100895.36	0.29[b]
				$\lvert I\rvert \times \lvert J\rvert = 1000 \times 1000$					
1	51408.02	24523	26	11397.95		51516.16	19098	102639.95	0.44[b]
2	51721.28	120158		67046.95	0.06[a]	52080.13	19700	48874.67	1.04[a]
3	51363.95	92283	31	100000.65	0.17[b]	51593.60	17681	102355.67	0.94[b]
4	52309.10	19775	23	7938.05		52319.24	25842	102373.61	0.17[b]
5	50859.51	18899	25	16720.70		51066.12	19700	74882.71	0.64[a]
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 300$					
1	44723.91	3529	30	863.72					
2	44441.89	12131	20	5689.69					
3	44096.78	1197	16	573.94					
4	44705.09	6429	23	1687.60					
5	43000.61	18649	25	7589.52					
				$\lvert I\rvert \times \lvert J\rvert = 1500 \times 600$					
1	49838.08	35243	24	30271.56					
2	49648.02	7231	24	5684.19					
3	49880.14	5223	23	4891.04					
4	50094.26	7815	20	3060.95					
5	49905.11	4231	21	3153.47					

[a] Terminated due to insufficient memory.
[a] Time limit of 100,000 seconds reached.

D Test problem generator

In the following, we reproduce the C-code used for generating the test problem instances.

```
/*---------------------------------------------------------------*/
/* FILE      : gencflp.c                                         */
/* VERSION   : 1.0                                               */
/* DATE      : July 13, 2000                                     */
/* AUTHOR    : Andreas Klose                                     */
/* SUBJECT   : program to generate CFLP test problems            */
/*---------------------------------------------------------------*/
/* USAGE     : gencflp <inputfile> <path>                        */
/*                                                               */
/* where "inputfile" is an ascii file providing the following    */
/* information on how to generate the test instances:            */
/* (1) Seed, that is an integer number specifying the seed to be */
*/     used for the random number generator (if Seed=0 then a seed*/
/*     is generated automatically)                               */
/* (2) for every problem class:                                  */
/*       #customers  #depot sites  ratio  problem name           */
/*     where ratio is the desired ratio of total capacity to total*/
/*                    total demand.                              */
/* The second argument "path`` is an optional and should specify */
/* the output path                                               */
/*---------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
long unifRand (long m);
/* returns uniform integer random number in [0,m). See file rnd.c */
double URand ();
/* returns uniform real random number in [0,1). See file rnd.c    */
/*---------------------------------------------------------------*/
int    m,n,num;     /* number of customers, depots, instances     */
int    *f, *d, *s;  /* fixed depot costs, demands and capacities  */
int    *x, *y;      /* coordinates                                */
int    totd,totc;   /* total demand and total capacity            */
char   name[512];   /* problem name                               */
char   fname[512];  /* name of file where to store problem data   */
double r;           /* r = totc/totd                              */
time_t tt;
/*---------------------------------------------------------------*/
void gencusts( void ) {
/* Generate customers with demand 5 + u[0,30] */
  int i;
  totd = 0;
  for (i=0;i<m;i++){
    d[i] = unifRand(31)+5;
    x[i] = unifRand(1000);
    y[i] = unifRand(1000);
    totd += d[i];
  }
}
```

```
/*-----------------------------------------------------------------------*/
void gendepots( void ) {
/* generate depots with capacity s[j] = 10 + u[0,150] and fixed cost
   f[j] = (u[0,10] + 100)*sqrt(s[j])+u[0,90] */
  int    j;
  double ff;
  double corr;
  totc = 0;
  for (j=0;j<n;j++){
    s[j]   = unifRand(151)+10;
    ff     = (unifRand(10) + 100.0)*sqrt((double)s[j])
             + unifRand(90) + 0.5;
    f[j]   = (int) ff;
    x[j+m] = unifRand(1000);
    y[j+m] = unifRand(1000);
    totc  += s[j];
  }
  corr = ( (double)totd/(double)totc)*r;
  totc = 0;
  for (j=0;j<n;j++){
    ff = s[j]*corr+0.5;
    s[j] = (int)ff;
    totc += s[j];
  }
}
/*-----------------------------------------------------------------------*/
void writeprob( void ) {
/* write problem to file */
  FILE   *out;
  double cost,dx,dy;
  int    i, j, *pdx, *pdy, *pcx, *pcy;
  time(&tt);
  out = fopen(fname,"wt");
  pcx = x,   pcy = y;
  pdx = x+m, pdy = y+m;
  if ( out ){
    fprintf(out,"[CFLP-PROBLEMFILE]\n");
    fprintf(out,"%s %s","generated at: ",ctime(&tt));
    fprintf(out,"%s %-d %s %-d %s %-.2f\n","#customers:",m,
            "; #depot sites:",n,"; ratio:",r);
    fprintf(out,"\n[DEPOTS]\n");
    fprintf(out,"capacity fixcost varcost xcoord ycoord name\n");
    for (j=0;j<n;j++){
      fprintf(out,"%-d %-d %-s %-d %-d %-s%-d\n",
              s[j],f[j],"0",pdx[j],pdy[j],"Depot",j);
    }
    fprintf(out,"\n[CUSTOMERS]\n");
    fprintf(out,"demand xcoord ycoord name\n");
    for (i=0;i<m;i++){
      fprintf(out,"%-d %-d %-d %-s%-d\n",d[i],pcx[i],pcy[i],
              "Customer",i);
    }
    fprintf(out,"\n[COSTMATRIX]\n");
    fprintf(out,"c= d_eucli(a,b) * 0.01\n");
    fprintf(out,"ROWS = DEPOTS / COLS=CUSTOMERS\n");
    fprintf(out,"[MATRIX]\n");
```

```
    fprintf(out,"%-s %-d\n","Dim",n,m);
    for (j=0;j<n;j++){
      for (i=0;i<m;i++){
        dx = abs(x[i]-x[m+j]);
        dy = abs(y[i]-y[m+j]);
        cost = sqrt(dx*dx + dy*dy)*0.01*d[i];
        fprintf(out,"%-.4f ",cost);
      }
      fprintf(out,"\n");
    }
  }
  fclose(out);
}
/*---------------------------------------------------------*/
int main(int argc, char **argv) {
  char *Ext = ".cfl", *nc, path[256], *slash = "/";
  int  count, goon, len;
  long seed;
  float ratio;
  FILE *file;
  printf("\n%s\n","============================================");
  printf("%s\n","        CFLP TEST PROBLEM GENERATOR          ");
  printf("%s\n","============================================");
  printf("%s\n"," USAGE: input_file [path]");
  printf("%s\n","        input_file = input file");
  printf("%s\n","        path       = output directory (optional)");
  printf("%s\n","============================================");
  printf("%s\n"," Structure of input file:");
  printf("%s\n"," (1) seed = seed to initialize rndnum generator");
  printf("%s\n","           (selected automatically if seed<=0)");
  printf("%s\n"," (2) for every problem class:");
  printf("%s\n","     #cust  #depots  ratio  problem name");
  printf("%s\n","============================================");

  strcpy(path,"");
  nc = (char *) calloc(10, sizeof(char));
  file = fopen(argv[1],"r");
  goon = (int)(file != NULL);
  if (goon){
    if (argc > 2){
      strcpy(path,argv[2]);
      len = strlen(path)-1;
      if ( path[len] != *slash ) strcat(path,slash);
    }
    goon = fscanf(file,"%d\n",&seed);
    if ( seed <= 0 ) {
      time(&tt);
      seed = (long)tt;
    }
    if (goon) initRand(seed);
  }
  while (goon > 0){
    goon = fscanf(file,"%d%d%f%d%s\n",&m,&n,&ratio,&num,name);
    r = (double)ratio;
    if (goon > 0){
      f = (int *) calloc(n, sizeof(int) );
```

```
      d = (int *) calloc(m, sizeof(int) );
      s = (int *) calloc(n, sizeof(int) );
      x = (int *) calloc(n+m, sizeof(int) );
      y = (int *) calloc(n+m, sizeof(int) );
      for (count=1;count<=num;count++){
        strcpy(fname,path);
        strcat(fname,name);
        gcvt((double)count,1,nc);
        strcat(fname,nc);
        strcat(fname,Ext);
        gencusts();
        gendepots();
        writeprob();
        printf("%s%d %s %s\n","Problem instance no.",count,
               "written to",fname);
      }
      free(f);
      free(d);
      free(s);
      free(x);
      free(y);
    }
  }
  if ( file != NULL ) fclose(file);
  printf("%s%d\n","Terminated. Used seed number = ",seed);
  return( 0 );
}
```

Random number generator

The above test instance generator uses the following C-code for generating random numbers. (The code is from Lionnel Maugis and taken from `http://www.cenaath.cena.dgac.fr/~maugis`)

```
/*------------------------------------------------------------------*/
FILE      Rnd.c
VERSION : 1.0
DATE    : 21 September 1998
LANGUAGE: C
AUTHOR  : Lionnel Maugis * Sofreavia / ATM
          maugis@cenaath.cena.dgac.fr
          http://www.cenaath.cena.dgac.fr/~maugis
          1, rue de Champagne - 91200 ATHIS-MONS
          Postal Address : Orly Sud 205 - 94542 ORLY AEROGARE CEDEX
SUBJECT : Portable Uniform Integer Random Number in [0-2^31] range
          Performs better than ansi-C rand()
          D.E Knuth, 1994 - The Stanford GraphBase
/*------------------------------------------------------------------*/
#define RANDOM() (*rand_fptr >= 0 ? *rand_fptr-- : flipCycle ())
#define two_to_the_31 ((unsigned long)0x80000000)
#define RREAL ((double)RANDOM()/(double)two_to_the_31)
#define mod_diff(x,y) (((x)-(y))&0x7fffffff)
static long A[56]= {-1};
long   *rand_fptr = A;
```

```
/* ----------------------------------------------------------------*/
long flipCycle() {
  register long *ii,*jj;
  for (ii = &A[1], jj = &A[32]; jj <= &A[55]; ii++, jj++)
    *ii= mod_diff (*ii, *jj);
  for (jj = &A[1]; ii <= &A[55]; ii++, jj++)
    *ii= mod_diff (*ii, *jj);
  rand_fptr = &A[54];
  return A[55];
}
/* ----------------------------------------------------------------*/
void initRand (long seed) {
  register long i;
  register long prev = seed, next = 1;
  seed = prev = mod_diff (prev,0);
  A[55] = prev;
  for (i = 21; i; i = (i+21)%55) {
    A[i] = next;
    next = mod_diff (prev, next);
    if (seed&1) seed = 0x40000000 + (seed >> 1);
    else seed >>= 1;
    next = mod_diff (next,seed);
    prev = A[i];
  }
  for (i = 0; i < 7; i++) flipCycle();
}
/* ----------------------------------------------------------------*/
long unifRand (long m) {
  register unsigned long t = two_to_the_31 - (two_to_the_31%m);
  register long r;
  do {
    r = RANDOM();
  } while (t <= (unsigned long)r);
  return r%m;
}
/* ----------------------------------------------------------------*/
double URand () {
  double x;
  x = RREAL;
  return ( x );
}
```

Compiling the program

The “makefile” listed below can be used for compiling the program

```
#-----------------------------------------------------------------
# FILE: makefile for program gencflp
# Date   : August 15, 2001
#-----------------------------------------------------------------
# Compilers:
CC     = gcc
CFLAGS = -O4 -Wall -I./
obj    = gencflp.o rnd.o
main   = gencflp
```

```
libs    = -lm

$(main):        $(obj)
        $(LINK.c) -o $@ $(obj) $(libs)

%.o:    %.c
        $(COMPILE.c) $<
```

After compiling the code, the test instances can be generated by entering the command “gencfl gencflp.input”, where the file gencflp.input is as follows.

```
963490972
300 300 5.0 5 T300x300_5_
500 500 5.0 5 T500x500_5_
700 700 5.0 5 T700x700_5_
1000 1000 5.0 5 T1000x1000_5_
1500 300 5.0 5 T1500x300_5_
1500 600 5.0 5 T1500x600_5_
300 300 10.0 5 T300x300_10_
500 500 10.0 5 T500x500_10_
700 700 10.0 5 T700x700_10_
1000 1000 10.0 5 T1000x1000_10_
1500 300 10.0 5 T1500x300_10_
1500 600 10.0 5 T1500x600_10_
300 300 15.0 5 T300x300_15_
500 500 15.0 5 T500x500_15_
700 700 15.0 5 T700x700_15_
1000 1000 15.0 5 T1000x1000_15_
1500 300 15.0 5 T1500x300_15_
1500 600 15.0 5 T1500x600_15_
300 300 20.0 5 T300x300_20_
500 500 20.0 5 T500x500_20_
700 700 20.0 5 T700x700_20_
1000 1000 20.0 5 T1000x1000_20_
1500 300 20.0 5 T1500x300_20_
1500 600 20.0 5 T1500x600_20_
```