ELSEVIER European Journal of Operational Research 113 (1999) 544–559

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

An exact algorithm for the capacitated facility location problems with single sourcing

Kaj Holmberg [a,*], Mikael Rönnqvist [b], Di Yuan [a]

[a] *Department of Mathematics, Division of Optimization, Linköping Institute of Technology, S-581 83 Linköping, Sweden*
[b] *Department of Engineering Science, University of Auckland, PB 92019 Auckland, New Zealand*

Received 2 April 1997; accepted 3 December 1997

Abstract

Facility location problems are often encountered in many areas such as distribution, transportation and telecommunication. We describe a new solution approach for the capacitated facility location problem in which each customer is served by a single facility. An important class of heuristic solution methods for these problems are Lagrangian heuristics which have been shown to produce high quality solutions and at the same time be quite robust. A primal heuristic, based on a repeated matching algorithm which essentially solves a series of matching problems until certain convergence criteria are satisfied, is incorporated into the Lagrangian heuristic. Finally, a branch-and-bound method, based on the Lagrangian heuristic is developed, and compared computationally to the commercial code CPLEX. The computational results indicate that the proposed method is very efficient. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Branch-and-bound; Integer programming; Facility location; Lagrangian heuristics

1. Introduction and problem formulation

Facility location models are used in many areas such as distribution, transportation and telecommunication. These models have received a great deal of attention in the research literature. Given a set of potential locations for facilities and a set of customers, the facility location problem is to locate facilities in such a way that the total cost for assigning facilities and satisfying the demand of customers is minimized. The facility location problem can be classified into different categories depending on the restrictions assumed. In the uncapacitated facility location problem each facility is assumed to have no limits on its capacity. In that case, each customer receives all required demand from one facility. When each facility has a limited capacity the problem is called the *capacitated* facility location problem. The *Single-Source Capacitated Facility Location* (SSCFL) problem is a special case of the capacitated facility location problem in which each customer can only be supplied from one facility.

* Corresponding author. Tel.: +46 13 282 867; fax: +46 13 100 746; e-mail: kahol@math.liu.se

To formulate the mathematical model for the SSCFL problem we introduce the following notation. Let m be the number of potential facilities, n the number of customers, a_j the demand of customer j, b_i the capacity of facility i, f_i the fixed cost for using/opening facility i, and c_{ij} the cost for assigning customer j to facility i. All coefficients are assumed to be nonnegative and integral. We define the following decision variables:

$$y_i = \begin{cases} 1 & \text{if facility } i \text{ is opened,} \\ 0 & \text{otherwise,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if facility } i \text{ serves customer } j, \\ 0 & \text{otherwise.} \end{cases}$$

The problem can then be stated by the following integer program:

$$\text{(P)} \qquad v^* = \min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i.$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_{ij} \leqslant b_i y_i \;\; \forall i, \tag{1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \;\; \forall j, \tag{2}$$

$$x_{ij} - y_i \leqslant 0 \;\; \forall i,j, \tag{3}$$

$$x_{ij} \in \{0,1\} \;\; \forall i,j, \tag{4}$$

$$y_i \in \{0,1\} \;\; \forall i. \tag{5}$$

The objective function is to minimize the cost of assigning customers to open facilities and the cost of establishing such facilities. Constraint set (1) can be referred to as the capacity constraints (or the facility constraints), and ensures that the customer demand served by a certain facility does not exceed its capacity. Constraint set (2) can be referred to as the demand constraints (or the customer constraints), and ensures that each customer is assigned to exactly one facility and constraint set (3) ensures that the assignments are made only to open facilities. Constraint set (3) is redundant in (P), but the LP-relaxation of (P) is strengthened considerably (i.e. gets a higher optimal objective function value) by its inclusion. In this model all variables are binary.

Within the area of telecommunications, location problems occur in many different circumstances. One important class of applications is when equipment of different kinds is to be installed, it could be concentrators (as in Pirkul, 1987) or other end equipment for optical fibers. The units to locate can be of very different sizes, from small electronic equipment to switching cabinets and exchange buildings. Often the situation is actually more complicated, but the location model captures the most essential parts of the problem, and is thus a sufficiently good approximation. The units (facilities) to be installed are almost always capacitated, and the single sourcing property is also quite usual, i.e. the demand unit (the "customer") is connected to only one facility.

One of the most successful approaches for solving the SSCFL problem is to use a so-called Lagrangian heuristic. These heuristics are based on a Lagrangian relaxation and solving the related Lagrangian dual problem, by solving a sequence of smaller and simpler subproblems. To ensure the construction of feasible solutions some heuristic procedure must be adopted, which, given a subproblem solution and/or the dual solution, attempts to generate a feasible solution. Klincewicz and Luss (1986) present an algorithm that is based on relaxing the constraints on the facility capacities. The corresponding Lagrangian subproblems then become uncapacitated facility location problems. Pirkul (1987), Barcelo and Casanovas (1984) and Sridharan (1993) develop algorithms based upon relaxing the customer assignment constraints. By this relaxation a number of knapsack problems are constructed in each iteration. Beasley (1993) presents a relaxation of both the capacity constraints and the customer assignment constraints. Beasley also gives a comprehensive comparison between various Lagrangian heuristics to facility location problems. For the problem considered in this paper, Beasley found that the approach developed by Pirkul (1987) provides the best feasible solutions, followed by Beasley (1993), and then Klincewicz and Luss (1986). This conclusion is corroborated in Åhlander (1994), where different Lagrangian heuristics for the SSCFL problem are compared.

Repeated matching has previously been used to obtain heuristic solutions to several routing and scheduling problems. The term *repeated matching* is used to describe the solving of a succession of related matching problems. The solution to one problem generates a new matching problem, and the sequence of problems typically terminates when the optimal matching leaves all elements unmatched. It has been used by Forbes (1992) in bus crew scheduling, and by Desrochers and Verhoog (1991) on the fleet size and mix vehicle routing problem. Altinkemer and Gavish (1991) and Wark and Holt (1994) use it as the basis for solving vehicle routing problems. It has also been used in air crew scheduling by Wark et al. (1997). Attempts to use it as a solution procedure for SSFLC is presented in Rönnqvist (1995).

In this paper, we propose a solution procedure based on a Lagrangian heuristic (using subgradient optimization), a strong primal heuristic (using repeated matching) and branch-and-bound. Specifically we combine a strong *dual* approach (the Lagrangian dual) with a strong *primal* (the repeated matching heuristic). The dual scheme generates lower bounds and the primal scheme generates feasible solutions and upper bounds. The solutions from the Lagrangian subproblems act as initial solutions for the primal heuristic. The branch-and-bound procedure utilizes information easily obtainable from the Lagrangian relaxation to speed up the bounding process.

Section 2 is devoted to the Lagrangian heuristic. The concept of repeated matching and how it can be adopted for the problem is then described in Section 3. In Section 4 the branch-and-bound framework is described. Solving the problem with the state-of-the-art commercial code CPLEX is discussed in Section 6. In Section 7 we present the computational results and in Section 8 we make some conclusions.

2. The Lagrangian heuristic

The Lagrangian heuristic consists of the following parts. A Lagrangian relaxation yields lower bounds on the optimal objective function value, but rarely feasible solutions. A subgradient procedure is used to solve the Lagrangian dual, yielding improvements of the lower bounds. A primal heuristic is used for finding primal feasible solutions, yielding upper bounds. Below we describe each of these components.

2.1. Lagrangian relaxation

Lagrangian relaxation can be applied to the SSCFL problem in various ways. In Åhlander (1994) a computational comparison of different relaxations is made. The following three approaches were investigated.

1. Relaxation of constraint set (1). The subproblem obtained is an *uncapacitated* facility location problem, which can be efficiently solved with the method of Erlenkotter (1978).
2. Relaxation of constraint set (2). The subproblem obtained separates into m knapsack problems, one for each facility, which can be solved by a knapsack code found in Martello and Toth (1990).
3. Relaxation of constraint set (3) and removal of the y-variables from constraint set (1). The subproblem obtained separates into one trivial problem in y and a generalized assignment problem on x. The latter is a quite difficult problem, but can be solved with a method found in Martello and Toth (1990).

Lagrangian heuristics with quite straightforward primal heuristics were implemented and tested on problems of the size $m = 50$ and $n = 500$. The results of the comparison in Åhlander (1994) are that the second approach seems to be most efficient, the third approach not quite as efficient, while the first approach is clearly inferior. Even though the tests in Åhlander (1994) may be considered as fairly preliminary, the same conclusion is obtained in Beasley (1993). Therefore we have chosen to use the second relaxation.

So in this paper we apply Lagrangian relaxation to constraint set (2), i.e. the demand/single-sourcing constraints. This was also done in Pirkul (1987) (the differences being the primal heuristic and the branch-and-bound framework). Denoting the multipliers associated with constraint set (2) with u_j yields the following Lagrangian relaxation:

$$\text{(DS)} \qquad g(u) = \min \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{i=1}^{m} f_i y_i + \sum_{j=1}^{n} u_j \left(1 - \sum_{i=1}^{m} x_{ij}\right)$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_{ij} \leqslant b_i y_i \ \forall i,$$
$$x_{ij} - y_i \leqslant 0 \ \forall i, j,$$
$$x_{ij} \in \{0,1\} \ \forall i, j,$$
$$y_i \in \{0,1\} \ \forall i.$$

This problem separates into one problem for each facility. For each i, we have

$$g_i(u) = \min \quad \sum_{j=1}^{n} (c_{ij} - u_j)x_{ij} + f_i y_i$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_{ij} \leqslant b_i y_i,$$
$$x_{ij} - y_i \leqslant 0 \ \forall j,$$
$$x_{ij} \in \{0,1\} \ \forall j,$$
$$y_i \in \{0,1\}.$$

Assuming $y_i = 1$ yields the following knapsack problem, where $\hat{c}_{ij} = c_{ij} - u_j$.

$$(\mathrm{K}_i) \qquad \kappa_i = \min \quad \sum_{j=1}^{n} \hat{c}_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_j x_{ij} \leqslant b_i,$$
$$x_{ij} \in \{0,1\} \ \forall j.$$

Clearly, if $\hat{c}_{ij} \geqslant 0$ then the optimal value of x_{ij} is zero, so x_{ij} can in such a case be removed from (K_i).

Let $\hat{g}_i = f_i + \kappa_i$, which can be considered as a reduced cost for y_i. If $\hat{g}_i < 0$ then $y_i = 1$ and the x-solution of (K_i) is the optimal solution. Otherwise it is not beneficial to open facility i, implying $y_i = 0$ (and $x_{ij} = 0 \ \forall j$). The optimal objective function value of (DS) becomes

$$g(u) = \sum_{i=1}^{m} \min\{0, \hat{g}_i\} + \sum_{j=1}^{n} u_j$$
$$= \sum_{i=1}^{m} \left(f_i \bar{y}_i + \sum_{j=1}^{n} (c_{ij} - u_j)\bar{x}_{ij} \right) + \sum_{j=1}^{n} u_j,$$

where $\bar{x}$ and $\bar{y}$ are the optimal values obtained as described above. The knapsack problems (K_i) are efficiently solved with the code MT1R in Martello and Toth (1990).

2.2. Subgradient optimization

The Lagrangian dual problem is

$$(\mathrm{LD}) \qquad v_{\mathrm{L}} = \max g(u)$$

and a subgradient, d, to the concave function $g(u)$ can be obtained as

$$d_j = 1 - \sum_{i=1}^{m} \bar{x}_{ij} \ \forall j,$$

where $\bar{x}$ is obtained from the optimal solution to (DS). The dual problem can be solved with subgradient optimization, as described below. K is the maximal number of iterations allowed. Let $\underline{v}$ and $\bar{v}$ denote the best known lower and upper bounds on v^*. Any feasible solution yields upper bounds. Initially one might use $\bar{v} = \sum_j (\max_i c_{ij}) + \sum_i f_i$.

1. Choose a starting point, $u^{(1)}$. Set $\underline{v} = -\infty$ and $k = 1$. Choose $\epsilon_1 > 0$ and $\epsilon_2 > 0$. Initialize $\bar{v}$.
2. Solve the subproblem (DS), yielding $g(u^{(k)})$ and $(\bar{x}, \bar{y})$. If $g(u^{(k)}) > \underline{v}$, let $\underline{v} = g(u^{(k)})$ and save $u^{(k)}$ as the best dual solution so far.
3. Make an attempt to modify $(\bar{x}, \bar{y})$ into a feasible solution (see Section 2.3), and possibly update $\bar{v}$.
4. Stop if $\bar{v} - g(u^{(k)}) < 1$, since then $\bar{v}$ is optimal.
5. Calculate the search direction $d^{(k)}$, and a step length, $t^{(k)}$, with the step length formula given below. Let $u^{(k+1)} = u^{(k)} + t^{(k)} d^{(k)}$.
6. Stop if any of the following criteria is satisfied: $\|d^{(k)}\| \leqslant \epsilon_1$, $\|u^{(k+1)} - u^{(k)}\| \leqslant \epsilon_2$ or $k \geqslant K$. Otherwise let $k = k + 1$ and go to step 2.

The step length is calculated by the following formula (Poljak, 1969):

$$t^{(k)} = \lambda \frac{\bar{v} - g(u^{(k)})}{\|d^{(k)}\|^2},$$

where $0 < \lambda < 2$. In the tests we have initially used $\lambda = 1$, and, if improvement of the lower bound $\underline{v}$ has not been achieved in five iterations, λ is halved. Step 4 is motivated by the fact that v^* is integral. If the objective of the procedure is not to identify the optimal upper bound, but to find the best lower bound, v_L, and the corresponding u, step 4 should be removed, since it is possible that $v_L > v^* - 1$.

2.3. A simple primal heuristic

There are several heuristics for finding primal feasible solutions proposed in the literature. We have chosen a simple way, based on the output from the Lagrangian relaxation subproblem. This solution is known to satisfy all the constraints, except the relaxed demand constraints. The following procedure is used regularly during the subgradient procedure (in step 3).

1. Unnecessary deliveries to customers receiving their demand from more than one facility are removed. Deliveries from facilities with the highest assignment costs are removed first.

2. Customers without deliveries are assigned to facilities. First the capacities of open facilities are compared to the demand, and for facilities capable of satisfying the demand the one with the least assignment cost is chosen. If no open facility can satisfy the demand, a new facility must be opened. The one with the least total cost, assignment cost and fixed cost, relative to the customer, is chosen.

Note that the heuristic might fail in step 2, since the remaining capacities of the facilities may make it impossible to satisfy all the demand. In that case no feasible solution is found at this subgradient iteration.

Let us now address the question of how often to use the primal heuristic described above during the subgradient procedure. There are three criteria for using it. In our tests, a maximum of 500 subgradient iterations are done, and the first criterion is to apply the primal heuristic every 50th iteration.

The primal heuristic uses the solution of the Lagrangian relaxation, and a dual solution with a better lower bound might yield an improved feasible primal solution. Therefore the second criterion is to use the primal heuristic every time the lower bound is improved. Furthermore, *if* the norm of the subgradient is very small, we may be close to the dual optimum. Due to the nondifferentiability of the dual function, such a subgradient might never be found. Still, the third criterion is to use the primal heuristic if the norm of the subgradient is "small" (empirically $n/5$ has been found to qualify as a small number in this context).

As mentioned above, the parameter λ is decreased if no improvements of the lower bound are obtained for a fixed number of subsequent iterations. However, if an improvement of the upper bound is obtained (with the help of the primal heuristic), λ is reset to its initial value. Reasons for this are that the lowered upper bound would otherwise yield an undesired shortening of the step length, and, furthermore, if the upper bound obtained is optimal and the active linear piece of the dual function is also active at the dual optimum, $\lambda = 1$ is the optimal choice.

2.4. Dual-based penalty tests

The solutions of the Lagrangian relaxation (DS) not only provide lower bounds and good starting points for the primal heuristic, but also enable penalty tests for identifying the correct value for some variables. These penalty tests can reduce the problem size and hence the computational effort, especially when the Lagrangian heuristic is embedded into a branch-and-bound framework. This section describes dual-based penalty tests for both y and x-variables, using a solution $(\bar{x}, \bar{y})$ of (DS) at $u = \bar{u}$ and a known upper bound $\bar{v}$ on v^*.

For a certain facility i, let us consider the effect on the objective function value $g(\bar{u})$ if y_i is forced to zero or one. Assume that $\bar{y}_i = 1$ in the current solution. According to Section 2.1, we have $\hat{g}_i = g_i(\bar{u}) < 0$. If variable y_i is forced to be zero, the corresponding $g_i(\bar{u})$ is obviously zero. Due to the separability of (DS), this has no effect on the subproblems related to other facilities. Therefore, the new objective function value given $\bar{u}$ and $y_i = 0$ would be $g(\bar{u}) + |\hat{g}_i|$, and we notice that $|\hat{g}_i|$ gives a lower bound of the increment of the lower bound $g(\bar{u})$. Therefore if $g(\bar{u}) + |\hat{g}_i|$ exceeds $\bar{v}$, it is clear that variable y_i must be one in an optimal solution. We can deduce a similar result for a facility i with $\hat{g}_i > 0$, in which case the optimal value of y_i must be zero. Hence we obtain the following penalty tests for the y-variables:

- Facility inclusion test: Fix y_i to one if $\bar{y}_i = 1$ and $g(\bar{u}) + |\hat{g}_i| \geqslant \bar{v}$.
- Facility exclusion test: Fix y_i to zero if $\bar{y}_i = 0$ and $g(\bar{u}) + \hat{g}_i \geqslant \bar{v}$.

Note that these tests are very simple to perform due to the fact that (DS) separates into one problem per facility.

Let us now investigate if similar tests can be used to determine the optimal value of an x-variable. Assume that $\bar{u}$ gives $\hat{c}_{ij} > 0$; x_{ij} is hence not considered while solving the knapsack problem (K_i) and will attain the value zero in the solution. Enforcing $x_{ij} = 1$ will obviously lead to an increase of the objective function value of knapsack (K_i). Furthermore, $x_{ij} = 1$ implies $y_i = 1$. This will give rise to an increase of $g_i(\bar{u})$ by *at least* $\hat{c}_{ij}$ irrespective of the value of y_i in the current solution. (If $y_i = 0$ in the solution, the increase will be larger.) Thus in this case $g(\bar{u})$ will be increased with at least $\hat{c}_{ij}$ if variable x_{ij} is forced to be one. According to the discussion above, $x_{ij} = 0$ must be optimal if $g(\bar{u}) + \hat{c}_{ij} \geqslant \bar{v}$.

On the other hand, if x_{ij} is zero in the solution but $\hat{c}_{ij} < 0$, we have to resolve the knapsack (K_i) in order to determine the effect of setting $x_{ij} = 1$. In this case, more computational effort is required, in contrast to the previous case.

Similarly, it is computational expensive to identify whether or not an x-variable must be one in an optimal solution. To do this, we need to select a variable x_{ij} which is one in the solution of a knapsack (K_i) and obtain a minimal increase of $g(\bar{u})$ if that x_{ij} is set to zero. This is only meaningful if $\bar{y}_i = 1$, otherwise $g(\bar{u})$ will not change at all. Since the solution of the knapsack (K_i) can be totally different with $x_{ij} = 0$, we need to resolve the knapsack problem to obtain a guaranteed increment. This means that the penalty tests for the x-variables are not symmetric in the sense that the inclusion and exclusion tests have the same complexity. It is easier to identify an x_{ij} that should be zero than conclude that it must be one. This is natural, since for each customer j, only one single x_{ij} will be one in a feasible solution.

In our algorithm, we only include the following dual-based penalty test on the x-variables:

- Customer exclusion test: Fix x_{ij} to zero if $\hat{c}_{ij} > 0$ and $g(\bar{u}) + \hat{c}_{ij} \geqslant \bar{v}$.

Since all three penalty tests described above require only one addition for each variable, they are performed each time the Lagrangian relaxation is solved, in step 2 of the subgradient procedure.

3. The repeated matching algorithm

Below follows a brief description of the repeated matching algorithm used as a strong primal heuristic in the Lagrangian heuristic framework. More details about the repeated matching algorithm can be found in Rönnqvist et al. (1995) and Rönnqvist (1995).

3.1. Matching problems

The basic matching problem may be described as follows. Given a set A of k elements $\{h_1, h_2, \ldots, h_k\}$, a *perfect matching* on A is a matching of elements in A such that each $h_i \in A$ is matched with one and only one $h_j \in A$. (Note that we allow the possibility of an element matching itself, effectively remaining unmatched.) Given costs d_{ij} of matching h_i with h_j, where $d_{ij} = d_{ji}$, the matching problem can be formulated as below. Note z_{ij} is 1 if h_i is matched with h_j and 0 otherwise.

$$\text{(M)} \qquad \min \sum_{i=1}^{k}\sum_{j=1}^{k} d_{ij} z_{ij}$$

$$\begin{aligned} \text{s.t.} \quad & \sum_{j=1}^{k} z_{ij} = 1, \quad i = 1, \ldots, k, \\ & \sum_{i=1}^{k} z_{ij} = 1, \quad j = 1, \ldots, k, \\ & z_{ij} = z_{ji} \quad \forall i, j, \\ & z_{ij} \in \{0, 1\}, \quad \forall i, j. \end{aligned}$$

3.2. Notation and terminology

In order to use the repeated matching algorithm for solving problem (P) we need to reformulate the problem. Let $I = \{1, 2, \ldots, m\}$ be the set of all facilities, $J = \{1, 2, \ldots, n\}$ the set of all customers, and $K = I \times G$, where G is the set of all nonempty subsets of J. We say $(i, C) \in K$ is *feasible* if $\sum_{j \in C} a_j \leqslant b_i$, and let F be the set of all feasible members of K. A *packing* is a subset P of F with the property that

$$(i_1, C_1), (i_2, C_2) \in P \Rightarrow C_1 \cap C_2 = \emptyset \quad \text{and} \quad i_1 \neq i_2.$$

Given a packing P, let

$$L_1 = I \setminus \bigcup_{(i,C) \in P} \{i\},$$

$$L_2 = J \setminus \bigcup_{(i,C) \in P} C, \quad L_3 = P.$$

The set L_1 consists of all facilities that are unused, L_2 consists of all customers which are not assigned to a facility, and L_3 consists of all used facilities with their assigned customers. The *cost* of the packing is

$$M|L_2| + \sum_{(i,C) \in P} f_i + \sum_{\{(i,j) | (i,C) \in P, j \in C\}} c_{ij},$$

where M is a large number.

By using the concept of unassigned customers we may have a packing which does not necessarily correspond to a feasible solution of problem P. If the set L_2 is nonempty then the packing violates the assignment constraints for customers in problem P. In the repeated matching approach we want to match elements of L_1, L_2 and L_3 with each other so as to generate new sets L_1', L_2' and L_3' so that the cost of the new packing is improved. If M is large, the matching will tend to decrease the number of elements in L_2 (as in a penalty function approach).

3.3. Matching costs

Given a packing we need to compute cost coefficients, d_{ij}, for formulation M. Different properties can be used for finding the matching costs when the different elements in L_1, L_2 and L_3 are matched. The cost matrix will consist of nine submatrices, named block [1], …, block [9]. However, since the cost matrix is symmetric we only need to investigate six of the blocks given in the following formula:

$$\mathrm{d} = \begin{bmatrix} [L_1 - L_1] & [-] & [-] \\ [L_2 - L_1] & [L_2 - L_2] & [-] \\ [L_3 - L_1] & [L_3 - L_2] & [L_3 - L_3] \end{bmatrix} = \begin{bmatrix} [1] & [-] & [-] \\ [2] & [3] & [-] \\ [4] & [5] & [6] \end{bmatrix}.$$

It is straightforward to compute the matching costs for blocks [1]–[3]. The main reason is that the feasibility checking is trivial. For the remaining blocks we have a more complicated situation, since we may swap customers between open facilities. It turns out that block [6] is the most general block and that blocks [4] and [5] are modified cases. For a more detailed description we refer to Rönnqvist et al. (1995).

To find the cost, in block [6], of a matching of two elements (i_1, C_1), $(i_2, C_2) \in P$ we need to find the best possible swapping of customers between the two facilities. This is an IP problem with a special property which makes it possible to reformulate it into an interval knapsack problem.

$$\begin{aligned} \text{(IK)} \quad & \min \sum_{j \in C_1} g_j w_j + \sum_{j \in C_2} h_j v_j \\ \text{s.t.} \quad & -\delta_v \leqslant \sum_{j \in C_1} - a_j w_j + \sum_{j \in C_2} a_j v_j \leqslant \delta_w, \\ & w_j, v_j \in \{0, 1\} \ \forall j. \end{aligned}$$

Here g_j and h_j are the costs if customers swap facility and w_j and v_j are the corresponding decision variables. The coefficients δ_w and δ_v are the surplus capacities at facilities i_1 and i_2, respectively. Problem (IK) can be solved by a number of different methods. We have used a dynamic programming procedure for its solution.

3.4. The repeated matching algorithm

We are interested in finding solutions to (P) by exploring different packings. The different packings are obtained using a heuristic method based on the ideas in Wark and Holt (1994). The heuristic uses the repeated matching approach to form new feasible packings from previously determined feasible packings. The process is started by taking any feasible packing and a sequence of feasible packings of decreasing cost is formed. When repeated matching fails to yield any further reduction in cost, the packing is split so that customers previously assigned to certain facilities become unassigned. After the splitting, a new sequence of packings is generated to facilitate further progress. This process is then repeated until no further progress can be made in a fixed number of splits. A scheme of the algorithm is as follows.

```
Initialize: Set Best_main = $\infty$, Best_local = $\infty$,
$k = 1$, $l = 1$, Max_split = $M_S$.
Find an initial solution (a feasible packing).
Main iteration $k$:
  1. Local iteration $l$:
     a. Compute matching costs, d.
     b. Solve matching problem (M).
     c. Update Best_local:
        (i) If no improvement, apply splitting
        and go to step 2.
        (ii) If improvement, update Best_local
        and set $l = l + 1$.
        Then return to step a.
  2. Update Best_main
     a. If no improvement in $M_S$ main iterations, stop.
     b. If improvement update Best_main, set
     $l = 1$, $k = k + 1$,
        Best_Local = $\infty$ and return to step 1.
End of algorithm
```

The splitting is needed when no improved solution can be found. Here, we take the current solution and remove some customers from one or more facilities. One alternative for splitting is the following. For each element $(i, C) \in L_3$, we generate a random number $\alpha \in [0, 1]$. Then for each customer $j \in C$ we generate another random number $\beta \in [0, 1]$. If $\beta \leqslant \alpha$ then we remove the customer from the set C and insert it in the set L_2.

This primal heuristic, based on repeated matching, denoted by RMH, is in Rönnqvist et al. (1995) and Rönnqvist (1995) shown to be a fairly efficient heuristic solution method for the SSCFL problem. An even stronger heuristic method is obtained by combining it with the Lagrangian heuristic, where the Lagrangian subproblem solutions can act as good starting solutions for RMH.

4. The branch-and-bound framework

The Lagrangian heuristic, denoted by LH, including the strong primal heuristic based on repeated matching, RMH, can be incorporated in a branch-and-bound framework, in order to produce the exact optimum of the problem and verify it. This method is denoted by LHBB. The Lagrangian heuristic is then used as a bounding procedure in each node of the branch-and-bound tree, yielding both lower and upper bounds. Branches are cut off when the lower bound exceeds the best upper bound known.

Concerning the branching strategy, we find the following. Branching only over the y-variables has been used, Fisk (1978), but is not enough to ensure integrality of the whole solution. Even if all y-variables are fixed to integer values, the x-solution might still be nonintegral. (What remains is a generalized assignment problem.) Hence further branching on x-variables is needed. Alternatively, we may only consider branching over the x-variables, as done in Neebe and Rao (1983). If x_{ij} is fixed to one, y_i must also be set equal to one. When all the x-variables are fixed, some of the y-variables must have been fixed to one, and the other ones can obviously be fixed to zero (since that minimizes the cost). Thus the whole solution is integral.

If only the x-variables are used for branching, we may use the ordinary LP-inspired branching, where a certain x-variable is set equal to one in one branch, and set equal to zero in the other. The two generated branches are not symmetric since while $x_{ij} = 1$ implies that $y_i = 1$ and $x_{ik} = 0$ for all $k \neq j$, $x_{ij} = 0$ usually has no significant effect on the subproblem. Another strategy, which is more symmetric, is to notice that the demand constraint simply says that a number of 0/1 variables should sum up to one. Hence if we divide x_{ij}, $i = 1, \ldots, m$, into two subsets, the variable equal to one will belong to exactly one of the sets. Branching can thus be made by dividing the set into two disjunctive subsets, and in each branch assume that the variable equal to one is in the corresponding subset. This is equal to fix some x-variables (the "other" set) to zero in each branch. However, computational tests show that this branching increases the lower bound quite slowly, especially in the initial nodes of the tree, and hence is not efficient.

By doing further comparisons, we found that a combined branching on both y and x-variables is clearly more efficient than only branching on x-variables, with respect to both the tree size and the execution time. Computational experiments show that the penalty tests described in Section 2.4 are very strong in identifying the optimal value of the y-variables and hence only a few branchings are needed before all y-variables are fixed. Hence the optimal y-values are determined very quickly, which is the major reason why it outperforms a purely x-variable based branching strategy. Therefore we always choose a y-variable to branch on, as long as at least one y-variable remains unfixed. Only when all y-variables are fixed, an x-variable is chosen, if further branching is needed. Since, in the worst case, the branch-and-bound method will enumerate all x-solutions, the optimum will be found within a finite number of steps by the method. Thus we can state that the LHBB method will find the *exact optimum* of (P) within a *finite* number of steps.

Let us recall that our method contains a very strong primal heuristic. This heuristic is *not* made to follow the fixations of the branch-and-bound tree. It starts from a solution obeying the fixations, but might proceed to better solutions, regardless of the fixations. (The exact convergence of the method is not lost, since if the heuristic does not find a better solution, it will stop at the solution obeying the fixations, which for example will happen if the fixations are complete and optimal.)

Therefore we are mostly interested in quickly increasing the lower bound. Tests made on small problems reveal that since the heuristic often finds the optimum quite early in the procedure, nonoptimal fixations might help the whole procedure to prove optimality quicker than optimal fixations. What happens is that the primal heuristic finds the optimum *in spite* of the fixations, while the nonoptimal branches are cut off very early due to the large increment of the lower bound. Thus the tree often becomes quite small.

The complete branching strategy is as follows. If there is any unfixed y-variable, we identify the variable y_i with the largest magnitude of the reduced cost, i.e. $|\hat{g}_i|$. If $\hat{g}_i > 0$, the algorithm starts with the 1-branch, otherwise it starts with the 0-branch. That is, the algorithm starts with the branch that seems to be nonoptimal. If all y-variables are fixed, the x-variable with the smallest reduced cost $\hat{c}_{ij}$ is chosen. This is usually the one with the most negative $\hat{c}_{ij}$, i.e. a variable that we certainly want to set equal to one. In a similar manner as for the y-variables, the algorithm starts with the branch that appears to be nonoptimal.

The algorithm searches the branch-and-bound tree by depth first. While getting deeper in the tree, the subgradient procedure does *not* start from scratch. The dual solution of the previous node gives a natural, often excellent starting point for the subgradient search. This also suggests that comparing to the number of subgradient iterations in the initial node, only a few iterations are needed deeper in the tree. We have used 100 as the maximum number of iterations in the first node and 10 deeper in the tree. After a backtracking in the tree, at most 15 iterations are made.

Comparing to branch-and-bound based on LP-relaxation, our solution method in the nodes of the tree is approximate, since branching is done when the subgradient procedure *stops*, which not necessarily means that the optimum at the node is found. Therefore, in many cases unnecessary branching is done, and for this reason we could expect the branch-and-bound tree to be larger than it would be for an LP-based branch-and-bound method. However, the subproblem used in the nodes is stronger than the LP-relaxation ((DS) does not have the integrality property, Geoffrion (1974)), so the best lower bound from the Lagrangian relaxation is in general better than that from the LP-relaxation. This has the effect of shrinking the tree. Obviously the efficiency of the algorithm depends on the net effect of these two factors.

5. Algorithm summary

In this section we summarize the proposed method LHBB by describing all its potential steps.

The following notation is used in the algorithmic description.

$\bar{v}$	the best known upper bound
$\hat{v}$	the upper bound obtained from the RMH algorithm
$\underline{v}$	the lower bound in each branch-and-bound node
k	iteration counter
$u^{(k)}$	the dual solution in iteration k
$d^{(k)}$	the subgradient in iteration k
$g(u^{(k)})$	the objective function value of (DS) in iteration k
$\lambda^{(k)}$	the coefficient value in the step length formula in iteration k
ϵ_1 and ϵ_2	small positive numbers
K	the maximum allowed number of iterations in each node
P	the number of subgradient iterations between usage of the RMH algorithm
L	the number of successive iterations without improvement of $\underline{v}$ before λ is halved.
$\mathscr{T}$	the uninvestigated node set of the branch-and-bound tree

1. Initialize branch-and-bound: Let $\mathscr{T}$ be the initial node in which no variable is fixed, and use the simple primal heuristic to get a feasible solution and an upper bound $\bar{v}$.
2. Initialize a tree node: Select a node from $\mathscr{T}$ according to the depth first strategy. Obtain a starting dual solution $u^{(1)}$, set $\underline{v} = -\infty$, $k = 1$ and $\lambda^{(1)} = 1$.
3. Lagrangian relaxation: Solve the Lagrangian relaxation (DS). If $g(u^{(k)}) > \underline{v}$, set $\underline{v} = g(u^{(k)})$. If $\bar{v} - \underline{v} < 1$, go to 9.
4. Penalty test: Perform the penalty tests for the y and x variables.
5. Feasible solution: If $(k \mathit{mod} P) = 0$ or if $\underline{v}$ was increased in step 3, apply the simple primal heuristic to the solution obtained in step 3, and use the resulting feasible solution as the starting point for algorithm RMH. If a better upper bound $\hat{v}$ is found, set $\bar{v} = \hat{v}$. If $\bar{v} - \underline{v} < 1$, go to 9.
6. Subgradient search: Calculate a subgradient, $d^{(k)}$, and step size, $t^{(k)}$. Compute the next dual solution $u^{(k+1)}$.
7. Stop check: If $||d^{(k)}|| < \epsilon_1$, $||u^{(k+1)} - u^{(k)}|| < \epsilon_2$ or $k \geqslant K$, go to 8. Otherwise, if $\underline{v}$ has not been improved in L iterations, let $\lambda^{(k+1)} = \lambda^{(k)}/2$. Set $k = k + 1$ and go to 3.
8. Branch: Select a variable to branch on, generate two new nodes and insert them into $\mathscr{T}$.
9. Cut: Remove the current node from $\mathscr{T}$. If $\mathscr{T} = \emptyset$, stop. Otherwise go to 2.

We notice that the Lagrangian heuristic LH is composed by the steps 2–7.

6. Direct solution with CPLEX

Another possibility for solving the SSCFL problem is of course to use a standard branch-and-bound code, since it is a linear integer programming problem with binary variables. The size of the problem is, however, a serious disadvantage, which actually makes solution of large problems this way impossible, due to the memory requirements.

The code CPLEX (version 3.0), for Integer Programming (CPLEX Optimization, Inc., 1994), is a state-of-the-art commercial general branch-and-bound IP-code (written in C), which however includes a network optimizer. The code can find and solve an underlying network optimization problem by ignoring some of the constraints. Then the dual simplex method is used for retrieving primal feasibility. This yields a rather efficient solution method for this kind of problem. In this case the underlying network is bipartite, i.e. a transportation problem is initially solved.

7. Computational results

7.1. Test problems

We have randomly generated four sets of test problems, each with different sizes and properties. Table 1 gives a summary of the sizes and the quotients between the total capacity, C_{TOT}, and the total demand, D_{TOT}. The coordinates of locations in set 1 were generated as U(10, 200). (We use the notation U(a, b) for a uniform distribution in the interval [a, b].) The Teitz and Bart (1968)

Table 1
The sizes of the test problems

Set	Problems	m	n	$C_{\mathrm{TOT}}/D_{\mathrm{TOT}}$
1	p1–p12	10	50	1.37–2.06
1	p13–p24	20	50	2.77–3.50
2	p25–p40	30	150	3.03–6.06
3	p41–p55	10–30	70–100	1.52–8.28
4	p56–p71	30	200	1.97–3.95

heuristic for the p-median problem is used to choose the locations of potential facilities from these coordinates. The costs are determined as $c_{ij} = \rho e_{ij}$, where e_{ij} is the Euclidean distance from facility i to location j and ρ is a positive scalar. (We use $\rho = 4$.)

In problems p1–p12 and p13–p24 we have 10 and 20 potential facilities respectively. The number of customers is 50 for all problems. The demand for p1–p12 and p13–p24 is generated from U(10, 50) and U(30, 80), respectively. We have tested different settings of the fixed costs and capacities for the potential facilities. The capacities range from 100 to 500 and the fixed costs range from 300 to 700.

In the second set of test problems, the locations were generated from U(10, 300). The assignment costs are based on a vehicle routing problem modified from Klincewicz et al. (1990). The cost, c_{ij}, is defined as the cost per unit distance ρ_1 times the incremental cost of inserting a location j into a route from a depot to a seed point i and back added to the cost of service at location j (service cost per demand ρ_2 times the amount of demand at location j).

$$c_{ij} = \rho_1(dw_j + e_{ij} + dw_i) + \rho_2 a_j.$$

Here, dw_j is the distance between the depot and the location of customer j. e_{ij} is the distance between the location of customer i and the location of customer j. We assume that the location of the depot is at coordinate (0, 0). We use $\rho_1 = 4$ and $\rho_2 = 2$. The Teitz and Bart heuristic is used to specify the seed points. The fixed set-up costs for facilities had different settings but were in the range 300–700. The demands were generated from U(10, 50) and the capacities from U(200, 600).

The third set of test problems is based on vehicle routing test problems used by Solomon (1987). To examine the behavior of the algorithm we vary the patterns of the customer locations and coefficient ranges. The test problems were generated as follows. The coordinates of locations and demands come from the random, random-clustered and clustered–clustered problems of Solomon. The fixed set-up costs for facilities come from U(300, 600) and the capacities of facilities come from U(100, 500). Two types of assignment cost c_{ij} are studied. For problems p41–p49, the assignment cost is based on the facility location problem, as for set 1. For problems p50–p55, the assignment cost is based on vehicle routing problem as for set 2.

The final fourth set is generated as set 1 but the number of potential facilities is 30 and the number of customers is 200. The capacities and fixed costs had different settings and were altered in the range 500–800 and 500–1500, respectively. More details about these test problems can be found in Rönnqvist et al. (1995).

7.2. The tests

In Tables 2–6 we give the results of the computational tests with fairly straightforward implementations in FORTRAN 77 of our branch-and-bound method based on the Lagrangian heuristic, LHBB, and the Lagrangian heuristic with the strong primal heuristic, LH.

For each of the 71 problems we give the results of direct solution with CPLEX and our methods LHBB and LH. We give the objective function value (the best upper bound), and indicate by an asterisk values that are not *proved* by the method to be optimal (though many of them are optimal), CPU-times in seconds on a Sun Sparc 20/HS151, and for CPLEX and LHBB, the number of nodes

Table 2
Computational results for problems p1–p12

Problem name	CPLEX			LHBB			LH		
	$\bar{v}$	Nodes	Time	$\bar{v}$	Nodes	Time	$\bar{v}$	Time	Err.
p1	8848	113	1.38	8848	0	0.13	8848	0.13	0.0000
p2	7913	156	1.75	7913	10	0.31	*7913	0.23	0.0010
p3	9314	117	1.42	9314	0	0.18	9314	0.18	0.0001
p4	10714	162	2.53	10714	14	0.59	*10714	0.47	0.0023
p5	8838	279	2.60	8838	34	0.62	*8838	0.43	0.0009
p6	7777	107	1.16	7777	2	0.17	7777	0.15	0.0001
p7	9488	210	2.51	9488	6	0.29	*9488	0.36	0.0009
p8	11088	253	3.29	11088	50	0.99	*11088	0.42	0.0053
p9	8462	63	0.80	8462	4	0.30	*8462	0.31	0.0010
p10	7617	9	0.42	7617	6	0.15	*7617	0.13	0.0009
p11	8932	98	1.15	8932	0	0.17	8932	0.17	0.0001
p12	10132	45	0.72	10132	10	0.71	*10132	0.54	0.0017

* Indicates an upper bound not proved to be optimal.

Table 3
Computational results for problems p13–p24

Problem name	CPLEX			LHBB			LH		
	$\bar{v}$	Nodes	Time	$\bar{v}$	Nodes	Time	$\bar{v}$	Time	Err.
p13	8252	48	2.39	8252	4	0.38	*8252	0.55	0.0002
p14	7137	7	1.26	7137	0	0.11	7137	0.11	0.0000
p15	8808	38	2.28	8808	0	0.27	8808	0.27	0.0001
p16	10408	30	2.43	10408	18	1.09	*10408	0.70	0.0027
p17	8227	26	1.82	8227	6	0.45	*8227	0.69	0.0001
p18	7125	11	1.41	7125	0	0.17	7125	0.17	0.0001
p19	8886	61	3.08	8886	7	0.88	*8887	0.84	0.0062
p20	10486	69	4.42	10486	12	1.43	*10487	1.26	0.0022
p21	8068	0	1.19	8068	0	0.13	8068	0.13	0.0001
p22	7092	0	0.24	7092	0	0.09	7092	0.09	0.0000
p23	8746	4	1.32	8746	5	0.61	*8746	0.67	0.0009
p24	10273	21	1.68	10273	18	1.27	*10273	0.67	0.0067

* Indicates an upper bound not proved to be optimal.

in the tree, while for LH we give the relative error between the upper and lower bounds obtained. (Failure to prove optimality occurs for CPLEX and LHBB when the method is terminated before the branch-and-bound tree is completely searched, and for LH when the bounds are not close enough at termination.)

CPLEX is used in an intelligent way, using the network optimizer in the initial node and dual simplex after branchings. (Also we note that the formulation used in Section 1 yields a stronger LP-relaxation than if y was not present in constraint set (1).) At most 20,000 nodes are allowed in the branch-and-bound tree, which prohibits the method to find the optimum for a few problems. We have limited the execution time to 1 hour for LHBB, and in LH at most 500 iterations are allowed.

The primal repeated matching heuristic, which is quite time-consuming, is used initially at every 5th iteration in LHBB, but at a decreasing rate if no improvement of the upper bound is found. (In most cases we have in this situation already found the optimum, but have not proved it.)

Studying the tables of computational results, we find the following (see Table 7). Of the 71 problems, eight are too difficult for CPLEX to solve exactly in 20,000 nodes (which gives solution times of up to 5 hours).

Table 4
Computational results for problems p25–p40

Problem name	CPLEX			LHBB			LH		
	$\bar{v}$	Nodes	Time	$\bar{v}$	Nodes	Time	$\bar{v}$	Time	Err.
p25	11630	757	102.68	11630	114	39.71	*11630	13.45	0.0053
p26	10771	3971	428.38	10771	146	35.76	*10771	7.10	0.0047
p27	12322	9513	1362.20	12322	106	91.65	*12322	13.66	0.0101
p28	13722	8029	1041.66	13722	112	102.68	*13722	15.77	0.0095
p29	*13039	20000	4479.92	12371	298	97.24	*12375	10.61	0.0042
p30	*11685	20000	2655.08	11331	522	254.17	*11346	8.61	0.0199
p31	*13415	20000	4309.74	13331	478	249.19	*13341	15.54	0.0174
p32	*15944	20000	5524.90	15331	752	367.09	*15361	15.32	0.0173
p33	11629	567	67.24	11629	86	14.22	*11629	7.21	0.0046
p34	10632	592	66.64	10632	8	6.62	*10632	7.55	0.0002
p35	12232	511	73.43	12232	84	21.01	*12232	10.55	0.0053
p36	13832	642	88.58	13832	118	35.64	*13832	11.21	0.0124
p37	11258	18	18.43	11258	0	3.28	11258	3.26	0.0001
p38	10551	6	21.52	10551	0	1.32	10551	1.31	0.0001
p39	11824	4	17.07	11824	0	3.59	11824	3.56	0.0001
p40	13024	4	20.89	13024	0	3.54	13024	3.49	0.0001

* Indicates an upper bound not proved to be optimal.

Table 5
Computational results for problems p41–p55

Problem name	CPLEX			LHBB			LH		
	$\bar{v}$	Nodes	Time	$\bar{v}$	Nodes	Time	$\bar{v}$	Time	Err.
p41	6589	193	6.24	6589	14	2.64	*6589	2.16	0.0017
p42	5663	84	7.66	5663	52	13.05	*5663	3.68	0.0064
p43	5214	0	1.71	5214	0	2.07	5214	2.07	0.0000
p44	7028	262	7.19	7028	4	0.80	*7028	1.19	0.0003
p45	6251	5	3.37	6251	0	1.06	6251	1.06	0.0002
p46	5651	44	6.89	5651	76	7.28	*5651	1.93	0.0023
p47	6228	486	9.12	6228	0	0.03	6228	0.03	0.0001
p48	5596	28	3.39	5596	8	0.41	*5596	0.41	0.0020
p49	5302	8	5.13	5302	0	0.18	5302	0.18	0.0002
p50	8741	512	11.17	8741	164	15.36	*8741	2.54	0.0089
p51	7414	10064	1081.39	7414	164	34.47	*7469	4.81	0.0260
p52	9178	3403	88.14	9178	26	3.97	*9178	3.77	0.0012
p53	8531	494	28.82	8531	0	0.29	8531	0.28	0.0000
p54	8777	6230	192.57	8777	0	0.14	8777	0.14	0.0001
p55	7654	1399	119.64	7654	34	3.28	*7672	1.48	0.0069

* Indicates an upper bound not proved to be optimal.

The Lagrangian heuristic by itself, LH, manages to verify optimality in 25 cases, and a comparison with the results of CPLEX shows that for an additional 30 problems the optimal solutions are actually found, but optimality is not verified. For eight of the 63 problems solved to optimality by CPLEX, LH yields worse upper bounds (for two of these problems the difference is only one unit). Finally, for the seven of the eight problems not solved to optimality by CPLEX, LH yields better solutions than CPLEX (in much shorter time).

As for the lower bounds, we find the following. In spite of the fact that LH does not always find the exact dual maximum v_L, it yields better lower bounds than the LP-relaxation for 64 of the 71 problems. The duality gap of the LP-relaxation is

Table 6
Computational results for problems p56–p71

Problem name	CPLEX			LHBB			LH		
	$\bar{v}$	Nodes	Time	$\bar{v}$	Nodes	Time	$\bar{v}$	Time	Err.
p56	*21112	20000	4680.99	21103	990	430.22	*21163	11.41	0.0092
p57	*27701	20000	7441.16	26039	2104	1532.67	*26167	24.76	0.0134
p58	*39414	20000	18248.50	*37260	3046	3600.25	*37792	26.84	0.0327
p59	*29557	20000	16137.80	27282	1088	1045.86	*27300	23.96	0.0122
p60	20534	14	29.63	20534	0	5.29	20534	5.23	0.0000
p61	24454	17	36.25	24454	0	11.24	24454	11.10	0.0000
p62	32643	569	704.84	32643	1052	2167.58	*32643	55.09	0.0111
p63	25105	244	65.70	25105	10	17.59	*25105	18.44	0.0008
p64	20530	0	2.47	20530	0	2.96	20530	2.94	0.0000
p65	24445	0	3.12	24445	0	6.71	24445	6.67	0.0000
p66	31415	1916	1261.62	31415	302	807.24	*31504	54.01	0.0082
p67	24848	2713	518.89	24848	38	58.96	*24876	19.23	0.0030
p68	20538	13	32.34	20538	0	4.67	20538	4.63	0.0000
p69	24532	6	37.17	24532	0	9.63	24532	9.56	0.0000
p70	32321	4306	1907.57	32321	282	338.31	*32393	39.42	0.0053
p71	25540	1004	280.17	25540	24	28.46	*25562	22.78	0.0011

* Indicates an upper bound not proved to be optimal.

Table 7
Average computational results for the problem groups

Problem names		CPLEX		LHBB		LH	
		Nodes	Time	Nodes	Time	Time	Err.
p1–p12	Max	279.00	3.29	50.00	0.99	0.54	0.0053
	Mean	134.33	1.64	11.33	0.38	0.29	0.0012
	Min	9.00	0.42	0.00	0.13	0.13	0.0000
p13–p24	Max	69.00	4.42	18.00	1.43	1.26	0.0067
	Mean	26.25	1.96	5.83	0.57	0.51	0.0016
	Min	0.00	0.24	0.00	0.09	0.09	0.0000
p25–p40	Max	20000.00	5524.90	752.00	367.09	15.77	0.0199
	Mean	6538.38	1267.40	176.50	82.92	9.26	0.0069
	Min	4.00	17.07	0.00	1.32	1.31	0.0001
p41–p55	Max	10064.00	1081.39	164.00	34.47	4.81	0.0260
	Mean	1547.47	104.83	36.13	5.67	1.72	0.0037
	Min	0.00	1.71	0.00	0.03	0.03	0.0000
p56–p71	Max	20000.00	18248.50	3046.00	3600.25	55.09	0.0327
	Mean	5675.12	3211.76	558.50	629.23	21.00	0.0061
	Min	0.00	2.47	0.00	2.96	2.94	0.0000
All	Max	20000.00	18248.50	3046.00	3600.25	55.09	0.0327
	Mean	3106.41	1032.14	176.17	161.84	7.32	0.0042
	Min	0.00	0.24	0.0	0.03	0.03	0.0000

in average 15 times larger, though in absolute numbers these differences are usually not very large.

The Lagrangian heuristic with branch-and-bound, LHBB, is able to verify optimality for 70 problems (all but one) within 1 hour. For the test problem p58 that was not solved by LHBB exactly, the best upper bound found by LHBB is about 5% better than the one found by CPLEX in 20,000 nodes and a CPU-time of about 5 hours. For the 70 problems solved by LHBB, only one problem (p62) requires a CPU-time more than half an hour, while CPLEX spends more than 1 hour on seven of these problems without finding the optimum.

Comparing LH with LHBB, we see that for small sized problems, LH has either proved optimality or provided good lower and upper bounds with very small gap. Hence only few branch-and-bound nodes are needed for LHBB to verify the optimality. The maximal number of iterations is 500 in LH and much less in each node in LHBB, which explains why for some problems (e.g. p7, p13, p17, p44) LHBB proves optimality by branch-and-bound in an execution time that is less than needed by LH, even though LH has not proved optimality for the problems. For large problems much more effort is required for LHBB, due to the enumeration nature of the branch-and-bound algorithm, even if the branching starts with a fairly small gap.

To further compare LHBB and CPLEX, we now consider the number of branch-and-bound tree nodes and the execution time of these two methods. Table 7 shows that in general the number of nodes in LHBB is quite low compared to CPLEX. The number of nodes in LHBB is less than in CPLEX in all problems except p23, p46 and p62. For some fairly difficult problems (p26–p32, p56–p57, p59), the tree size in LHBB is about 10% or less compared to CPLEX, disregarding the fact that for some of these problems, more nodes are needed if CPLEX is to complete the branch-and-bound procedure. For the 63 problems solved exactly by both methods, CPLEX requires in average more than 18 times as many nodes as LHBB. Again we can draw the conclusion that our Lagrangian subproblem is stronger than the LP-subproblem used by CPLEX.

LHBB also outperforms CPLEX when it comes to CPU-time. The average solution time for the 63 problems solved to optimality by CPLEX is 960 seconds, while for LHBB on the same problems, the average solution time is 51 seconds. Of the 70 problems solved by LHBB, it is faster than CPLEX in exactly 90% of these cases (63 problems). For these 63 problems, the average speed-up in LHBB is 36. If we disregard the two extreme cases p47 and p54, in which LHBB is 304 and 1375 times faster than CPLEX respectively, the speed-up factor for the remaining problems is 10. For the other seven problems, CPLEX is faster with a factor 1.68 in average. Note that since CPLEX did not complete its branch-and-bound for some of the problems and the tables show that a gap still is left between the upper bound found by CPLEX and the optimum, the true speed-up factor for LHBB is larger than the numbers mentioned above.

8. Conclusions

We have proposed a new approach to solve the SSCFL problem. It consists of a Lagrangian heuristic coupled to a strong primal heuristic, within a branch-and-bound framework. The primal heuristic is based on solving a sequence of related matching problems. In each of the primal heuristic iterations we solve a matching problem to optimality and to find the related matching costs we solve a number of knapsack problems. The Lagrangian relaxation is also solved by solving a number of knapsack problems, and the Lagrangian dual is solved by subgradient optimization.

The computational results lead to the following conclusions. The bounding procedures in the proposed method are quite efficient. More specifically, the Lagrangian relaxation together with subgradient optimization provide strong lower bounds to the problem. The primal heuristic, RMH, is very powerful in quickly finding optimal or near optimal solutions.

By combining the dual and the primal approach, the Lagrangian heuristic LH is an efficient heuristic approach for the SSCFL problem. The results show that LH terminates with either proved optimality or a fairly small gap.

The Lagrangian heuristic with branch-and-bound, LHBB, is a promising exact solution method for the SSCFL problem. All components in the method contribute to its performance. LHBB outperforms for example a state-of-the-art mixed integer code, with respect to both the tree size and solution time. These differences are more pronounced for the most difficult of our test problems, and make LHBB preferable to use for difficult problems.

Acknowledgements

This work has been partly financed by KFB (The Swedish Transport and Communication Research Board), NUTEK (The National Board for Industrial and Technical Development), Telia AB and Ericsson Telecom AB.

References

Åhlander, F., 1994. The capacitated facility location problem with single sourcing, Master Thesis LiTH-MAT-EX-1994-07, Linköping University, Sweden (in Swedish).

Altinkemer, K., Gavish, B., 1991. Parallel savings based heuristics for the delivery problem. Operations Research 39, 456–469.

Barcelo, J., Casanovas, J., 1984. A heuristic Lagrangian algorithm for the capacitated plant location problem. European Journal of Operational Research 15, 212–226.

Beasley, J.E., 1993. Lagrangian heuristics for location problems. European Journal of Operational Research 65, 383–399.

CPLEX Optimization, Inc. (1994), Using the CPLEX Callable Library, version 3.0.

Desrochers, M., Verhoog, T., 1991. A new heuristic for the fleet size and mix vehicle routing problem. Computers and Operations Research 18, 263–274.

Erlenkotter, D., 1978. A dual-based procedure for uncapacitated facility location. Operations Research 26, 992–1009.

Fisk, J., 1978. A solution procedure for a special type of capacitated warehouse location problem. Logistics Transportation Review 13, 305–320.

Forbes, M., 1992. Vehicle and Crew Scheduling in Public Transport. Ph.D. thesis, University of Queensland, Brisbane.

Geoffrion, A.M., 1974. Lagrangian relaxation for integer programming. Mathematical Programming Study 2, 82–114.

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.

Klincewicz, J., Luss, H., Pilcher, M., 1990. Fleet size planning when outside carrier service are available. Transportation Science 24, 169–182.

Martello, S., Toth, P., 1990. Knapsack Problems, Algorithms and Computer Implementations, Wiley, New York.

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

Pirkul, H., 1987. Efficient algorithm for the capacitated concentrator location problem. Computers and Operations Research 14, 197–208.

Poljak, B.T., 1969. Minimization of unsmooth functionals. USSR Computational Mathematics and Mathematical Physics 9, 14–29.

Rönnqvist, M., 1995. Solving single-source capacitated facility location problems using repeated matching. In: Proceedings of the 31st Annual Conference of the Operational Research Society of New Zealand, Wellington, New Zealand pp. 187–194.

Rönnqvist, M., Tragantalerngsak, S., Holt, J., 1995. A new approach to solve single-source capacitated facility location problems, European Journal of Operational Research, Forthcoming.

Solomon, M., 1987. Algorithms for the vehicle routing and scheduling problem with time window constraints. Operations Research 35, 254–265.

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

Teitz, M., Bart, P., 1968. Heuristic methods for estimating the general vertex median of a weighted graph. Operations Research 16, 955–961.

Wark, P., Holt, J., 1994. A repeated matching heuristic for the vehicle routeing problem. Journal of Operational Research Society 45, 1156–1167.

Wark, P., Holt, J., Rönnqvist, M., Ryan, D., 1995. Aircrew schedule generation using repeated matching, European Journal of Operational Research 102, 21–35.