European Journal of Operational Research 238 (2014) 438–450

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A heuristic for BILP problems: The Single Source Capacitated Facility Location Problem

G. Guastaroba *, M.G. Speranza

Department of Economics and Management, University of Brescia, C.da S. Chiara 50, Brescia, Italy

ARTICLE INFO

Article history:
Received 22 May 2013
Accepted 4 April 2014
Available online 16 April 2014

Keywords:
Single Source Capacitated Facility Location Problems
Binary Integer Linear Programming
Heuristic algorithms
Kernel Search framework

ABSTRACT

In the Single Source Capacitated Facility Location Problem (SSCFLP) each customer has to be assigned to one facility that supplies its whole demand. The total demand of customers assigned to each facility cannot exceed its capacity. An opening cost is associated with each facility, and is paid if at least one customer is assigned to it. The objective is to minimize the total cost of opening the facilities and supply all the customers. In this paper we extend the Kernel Search heuristic framework to general Binary Integer Linear Programming (BILP) problems, and apply it to the SSCFLP. The heuristic is based on the solution to optimality of a sequence of subproblems, where each subproblem is restricted to a subset of the decision variables. The subsets of decision variables are constructed starting from the optimal values of the linear relaxation. Variants based on variable fixing are proposed to improve the efficiency of the Kernel Search framework. The algorithms are tested on benchmark instances and new very large-scale test problems. Computational results demonstrate the effectiveness of the approach. The Kernel Search algorithm outperforms the best heuristics for the SSCFLP available in the literature. It found the optimal solution for 165 out of the 170 instances with a proven optimum. The error achieved in the remaining instances is negligible. Moreover, it achieved, on 100 new very large-scale instances, an average gap equal to 0.64% computed with respect to a lower bound or the optimum, when available. The variants based on variable fixing improved the efficiency of the algorithm with minor deteriorations of the solution quality.

1. Introduction

A Binary Integer Linear Programming (BILP) problem is a linear programming problem where all variables are constrained to take a binary (either 0 or 1) value. In the Single Source Capacitated Facility Location Problem (SSCFLP), a specific BILP problem, we are given a set I of customers. Each customer $i \in I$ has a demand d_i to be served. We are also given a set J of potential locations where one facility could be opened. For the sake of brevity, hereafter we refer to each potential location $j \in J$ as facility j. A fixed opening cost f_j and a capacity s_j are associated with each facility $j \in J$. Assigning customer i to facility j, i.e. supplying its whole demand from j, costs c_{ij}. The SSCFLP consists in selecting which facilities to open from set J and how to assign customers in set I to the selected facilities while minimizing the sum of opening and assignment costs. Each customer demand must be fully satisfied and each facility, if opened, cannot supply more than its capacity. The additional requirement that differentiates the SSCFLP from the multi source Capacitated Facility Location Problem (CFLP) is that each customer has to be supplied by exactly one facility, whereas in the CFLP any customer can be supplied by more than one facility. The single source constraint makes the problem much harder to solve (see Klose & Drexl (2005)). Given a set of open facilities, the problem of supplying customers from those facilities in the CFLP is a linear program (specifically, a transportation problem). On the contrary, for a given set of open facilities, the associated assignment problem in the SSCFLP is a particular case of the Generalized Assignment Problem (GAP) which is $\mathcal{NP}$-hard itself (e.g., see Fisher, Jaikumar, & Van Wassenhove (1986)).

The single source assumption is a critical issue in several real-life applications. We mention, among others, the problem of finding the location of drilling platforms and the consequent allocation of oil wells to platforms studied in Devine and Lesso (1972), the capacitated concentrator location problem described in Pirkul (1987) and the distribution systems mentioned in Díaz and Fernández (2002).

We introduce the following binary variables. Let

* Corresponding author. Tel.: +39 30 2988588.
E-mail addresses: guastaro@eco.unibs.it (G. Guastaroba), speranza@eco.unibs.it (M.G. Speranza).

http://dx.doi.org/10.1016/j.ejor.2014.04.007
0377-2217/

$$x_{ij} = \begin{cases} 1 & \text{if customer } i \text{ is assigned to facility } j, \\ 0 & \text{otherwise;} \end{cases}$$

and

$$y_j = \begin{cases} 1 & \text{if facility } j \text{ is opened}, \\ 0 & \text{otherwise}. \end{cases}$$

Then, the SSCFLP can be stated as the following BILP problem

SSCFLP Model

$$\text{minimize} \quad \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{j\in J} f_j y_j \tag{1}$$

$$\text{subject to} \quad \sum_{i\in I} d_i x_{ij} \leqslant s_j y_j \quad j \in J \tag{2}$$

$$\sum_{j\in J} x_{ij} = 1 \quad i \in I \tag{3}$$

$$x_{ij} \leqslant y_j \quad i \in I, \quad j \in J \tag{4}$$

$$x_{ij} \in \{0,1\} \quad i \in I, \quad j \in J \tag{5}$$

$$y_j \in \{0,1\} \quad j \in J. \tag{6}$$

Objective function (1) minimizes the total cost given by two components. The first term computes the total cost of assigning the customers to the facilities. The second term is the total cost of opening the facilities. Inequalities (2) establish that the total demand of customers assigned to each facility must not exceed its capacity. Assignment constraints (3), along with (5), ensure that the demand of each customer is supplied by exactly one facility. Constraints (4), that are redundant in (1)–(6), yield a much tighter linear relaxation than the equivalent formulation without constraints (4) (e.g., see Yang, Chu, & Chen (2012)). Constraints (5) and (6) define the decision variables. Finally, without loss of generality, it is assumed that $c_{ij} \geqslant 0,\ i \in I,\ j \in J;\ f_j \geqslant 0,\ j \in J;\ s_j > 0,\ j \in J;\ d_i \geqslant 0,\ i \in I,$ and $\sum_{j\in J} s_j \geqslant \sum_{i\in I} d_i$.

Facility location problems have received a great deal of attention in the literature as they appear in many different areas. A non-comprehensive list of applications includes distribution, transportation and telecommunication problems. Interested readers are referred to the survey by Klose and Drexl (2005), the book edited by Mirchandani and Francis (1990) and the more recent book edited by Eiselt and Marianov (2011) for an overview on this class of problems.

As the SSCFLP belongs to the class of $\mathcal{NP}$-hard problems (e.g., see Yang et al., 2012), most of the solution approaches appeared in the literature are heuristics. Among them, solution methods based on a Lagrangean relaxation of the SSCFLP play a dominant role. These methods mainly differ from each other in the set of constraints that are relaxed (i.e., the capacity constraints (2), or the assignment constraints (3), or both) and in the way feasible solutions are generated from the solutions of the relaxed problem. For instance, Hindi and Pienkosz (1999) compute a lower bound dualizing constraints (3) in a Lagrangean fashion, and find feasible solutions by means of a procedure that combines a greedy heuristic with a restricted neighborhood search. Also Cortinhal and Captivo (2003) obtain lower bounds dualizing constraints (3) whereas a local search procedure and a tabu search algorithm are proposed to compute upper bounds. Chen and Ting (2008) propose a hybrid algorithm that combines a Lagrangean heuristic with an ant colony system, and also a multiple ant colony system for the SSCFLP. For an overview of Lagrangean relaxation-based techniques for solving facility location problems we refer to Galvão and Marianov (2011), where special emphasis is given to the SSCFLP. Among the other heuristics proposed in the literature, we mention the tabu search heuristic for the SSCFLP introduced by Filho and Galvão (1998), and the heuristics proposed by Delmaire, Díaz, Fernández, and Ortega (1999), namely a reactive GRASP heuristic, a tabu search heuristic, and two different hybrid approaches that combine elements of the GRASP and of the tabu search methodologies. Rönnqvist, Tragantalerngsak, and Holt (1999) consider a repeated matching algorithm which essentially solves a series of matching problems until certain convergence criteria are satisfied. Ahuja, Orlin, Pallottino, Scaparra, and Scutellà (2004) present a Very Large-scale Neighborhood Search algorithm (VLNS) for the SSCFLP, whereas Contreras and Díaz (2008) develop a scatter search algorithm. To the best of our knowledge, the best heuristics are the VLNS developed by Ahuja et al. (2004) and the hybrid algorithm designed by Chen and Ting (2008). In both papers, computational results are reported for instances with up to 100 facilities and 1000 customers.

Among the exact methods, we mention the paper by Holmberg, Rönnqvist, and Yuan (1999) who propose an algorithm consisting of a Lagrangean dual heuristic (used to compute a lower bound) coupled with a strong primal heuristic (used to compute an upper bound), within a branch-and-bound framework. Computational results are reported for instances involving up to 30 facilities and 200 customers. Díaz and Fernández (2002) develop an exact algorithm in which a column generation procedure for finding upper and lower bounds for the SSCFLP is incorporated within a branch-and-price framework. They solve instances with up to 30 facilities and 90 customers. To the best of our knowledge, the most recent exact approach proposed in the literature is the cut-and-solve algorithm (see Climer & Zhang, 2006) introduced in Yang et al. (2012). The authors solve to optimality instances with up to 80 facilities and 400 customers.

The Kernel Search is a heuristic proposed for the solution of Mixed Integer Linear Programming (MILP) problems with binary variables. The general idea of the heuristic is to identify subsets of the decision variables and solve to optimality, whenever possible, the resulting restricted problems by means of a general-purpose MILP solver used as a black-box. In its original form, the heuristic was successfully applied to the multi-dimensional knapsack problem in Angelelli, Mansini, and Speranza (2010) and to a portfolio selection problem in Angelelli, Mansini, and Speranza (2012). Some improvements were introduced in Guastaroba and Speranza (2012a) where the Kernel Search was applied to the index tracking problem. Some further enhancements were proposed in Guastaroba and Speranza (2012b) where the Kernel Search was applied to the CFLP.

The idea of fixing the value for some of the variables inspired several successful algorithms recently proposed for the solution of MILP problems. The so called soft variable fixing mechanism (intuitively, variables to be fixed are not chosen explicitly but selected implicitly adding a linear constraint to the MILP model) leads to the local branching algorithm proposed by Fischetti and Lodi (2003). The Relaxation Induced Neighborhood Search (RINS) introduced by Danna, Rothberg, and Pape (2005) is based on the intuition that, given an incumbent solution, many variables take the same value in the linear relaxation and in the incumbent solution. When performing a tree search, at some nodes of the global branch-and-cut tree the RINS algorithm fixes the variables with the same values in the incumbent and in the current linear relaxation solution (hard variable fixing). Finally, in the variable neighborhood decomposition search proposed by Lazić, Hanafi, Mladenović, and Urošević (2010) the two approaches are combined: hard variable fixing is used in the main scheme, whereas soft variable fixing is adopted in the local search.

Contributions of the paper. The Kernel Search framework is applied in Angelelli et al. (2010) to a BILP problem with only one set of binary variables. It was used to solve MILP problems where a continuous variable is associated with each binary variable in Angelelli et al. (2012) and Guastaroba and Speranza (2012a). Guastaroba and Speranza (2012b) extended the heuristic to a MILP problem where a large number of continuous variables are associated with each binary variable.

In this paper, we further develop the Kernel Search framework in the following ways. Firstly, we extend the heuristic to a general BILP problem and apply it to the SSCFLP. Secondly, we propose two variants of the standard framework aiming at improving the efficiency of the algorithm. Both variants are based on hard variable fixing. The selection of the variables to be fixed is guided by the information provided by the optimal solution of the linear relaxation. Guastaroba and Speranza (2012a) proposed a refinement procedure that aims at improving the best solution found solving the sequence of restricted problems. The idea of the refinement procedure is to set to 1 some binary variables on the basis of their values in the optimal solutions of the restricted problems, and then solve to optimality the MILP problem restricted to the remaining variables. Different from Guastaroba and Speranza (2012a), the variants proposed in the present paper make use of the information retrieved from the optimal solution of the linear relaxation to fix some binary variables either to 1 or to 0. This allows us to perform variable fixing for all the restricted problems in the sequence.

An important contribution of the current paper is to the solution of the SSCFLP. Indeed, extensive computational experiments show that the Kernel Search significantly outperforms the best heuristics for the SSCFLP known in the literature. Optimal solutions to benchmark instances are provided in Holmberg et al. (1999) for 71 instances ranging from 10 to 30 facilities and from 50 to 200 customers. Optimal solutions are also reported in Yang et al. (2012) for 20 instances ranging from 30 to 80 facilities and from 200 to 400 customers. 12 large-scale benchmark instances, taken from the CFLP literature, were first solved in Hindi and Pienkosz (1999) with a heuristic algorithm. These instances comprise 100 facilities and 1000 customers and we solved them to optimality with a general-purpose MILP solver, namely CPLEX. As mentioned above, the most successful heuristics available in the literature are those proposed by Ahuja et al. (2004) and Chen and Ting (2008). Both papers report computational experiments for the 71 instances provided in Holmberg et al. (1999) and the 12 instances considered in Hindi and Pienkosz (1999). The average error with respect to the optimal solution reported in Holmberg et al. (1999) is 0.03% for the best VLNS implemented in Ahuja et al. (2004), whereas the hybrid algorithm designed in Chen and Ting (2008) achieved an average error of 0.02%. On the large-scale instances proposed in Hindi and Pienkosz (1999), the average error for the former two heuristics with respect to the optimal solution, computed with CPLEX, is 0.08% and 0.05%, respectively. The standard Kernel Search solves to optimality almost all the aforementioned benchmark instances within fractions of a second for the small and medium-scale instances, and within few minutes for the larger instances. The error computed for the only two instances that are not solved to optimality is 0.01%. The standard Kernel Search solves very large-scale instances ranging from 300 to 1000 facilities and from 300 to 1000 customers in approximately 1 hour of average computing time. Optimal solutions are found with CPLEX for 43 out of the 100 very large-scale instances. To the best of our knowledge, instances of this size have never been tested before for the SSCFLP. The standard Kernel Search solves to optimality 40 out of the latter 43 instances, whereas the average error on the remaining 3 instances is smaller than 0.08%. Computational results show that both the Kernel Search and the variants outperform CPLEX, tested with several different settings, both in terms of solution quality and computing time.

Structure of the Paper. In Section 2 the general Kernel Search framework for BILP problems is presented. A detailed description of the Kernel Search implemented for the SSCFLP along with its variants is provided in Section 3. Section 4 is devoted to the computational analysis. Finally, in Section 5 some concluding remarks are drawn.

2. The Kernel Search for BILP problems

In this section we provide a description of the Kernel Search that can be applied to any BILP problem. We also introduce the basic concepts and definitions that will be used in the rest of the paper.

We consider BILP problems with several sets of binary variables (for example, x, y and z) and assume, without loss of generality, that the BILP is a minimization problem. We refer to the BILP problem including all the binary variables as the *original* problem. We call *restricted* problem the BILP problem restricted to a subset of the binary variables.

We say that a binary variable is *promising* if it is likely that it takes value 1 in an optimal solution of the original problem. We refer to *kernel* as a set of promising variables. We distinguish between the kernel of each set of binary variables and the kernel of the original problem. The promising variables belonging to a given set (for example, x) compose its *individual kernel*. The kernel of the original problem is the union of the individual kernels of all the sets of variables and is referred to as the *global kernel*. All individual kernels, and therefore the global kernel, vary during the algorithm.

At the beginning of the Kernel Search the linear relaxation of the original problem is solved. The promising variables of each set compose its *initial individual kernel*. The union of the initial individual kernels of all sets of variables form the *initial global kernel*. All the variables of each set not included in its initial individual kernel are partitioned into ordered groups, called *individual buckets*.

The nodal part of the algorithm is the solution of a sequence of restricted problems. The first BILP problem in the sequence is restricted to the initial global kernel. Then, any subsequent BILP problem is restricted to the *current global kernel*, given by the union of the current individual kernels of all sets of variables, and one individual bucket for each set of variables. The *current individual kernel* of a set of variables is the previous current individual kernel from which some variables are dropped – those that are no longer promising – and some are added – those that have become promising.

Any feasible solution of a restricted BILP problem in the sequence is a heuristic solution of the original problem and provides an upper bound on its optimal solution that is used as a cut-off value for all the following restricted BILP problems. The Kernel Search stops when a given stopping criterion is met.

The general scheme of the Kernel Search for BILP problems is sketched in Algorithm 1.

The initial individual kernel may be chosen as the set of variables in the basis of the optimal solution of the linear relaxation. The sequence of buckets may be built by ordering the variables out of the basis using the reduced costs coefficients, from the smallest to the largest. The current individual kernel may be updated as follows. A variable belonging to an individual bucket that takes a positive value in the optimal solution of a restricted BILP problem may be considered to have become promising. Conversely, if a promising variable belonging to the current individual kernel has not taken a positive value in the optimal solution of a certain number of restricted BILP problems, then it may be considered to be no longer promising.

Algorithm 1. General Scheme of the Kernel Search for BILP problems.

1: Solve the linear relaxation.
2: For each set of variables, identify the initial individual kernel and the sequence of individual buckets.
3: Build the initial global kernel.
4: Solve a BILP problem on the initial global kernel.
5: Repeat the following until a stopping criterion is met
  (a) build the current global kernel;
  (b) solve a BILP problem on the current global kernel plus one individual bucket for each available set of variables;
  (c) for each set of variables, update its current individual kernel.

3. The Kernel Search for the SSCFLP

In this section we describe the standard Kernel Search, and some variants, we implemented for the SSCFLP.

3.1. The Standard Kernel Search

We consider the SSCFLP formulation (1)–(6). We denote as BILP($J, I \times J$) the BILP problem that takes into consideration the whole set of facilities J and, for each facility, the whole set of customers I (i.e., the original problem). We denote as LP($J, I \times J$) the linear relaxation of BILP($J, I \times J$), i.e. constraints (5) and (6) are substituted by $x_{ij} \in [0,1],\ i \in I,\ j \in J$, and by $y_j \in [0,1],\ j \in J$, respectively.

In the SSCFLP a set of two-index binary variables x_{ij} (assign customers to facilities) is associated with each one-index binary variable y_j (open the facility or not). This characteristic has a crucial importance in the design of the Kernel Search for the SSCFLP. Indeed, the (initial and current) individual kernel for set of variables x has to be consistent with the (initial and current) individual kernel for set of variables y. The same reasoning also applies to the individual buckets.

We denote as $K(y)$ and $K(x)$ the generic *individual kernel* for variables y and x, respectively. Each facility in $K(y)$ can serve only a subset of all the customers, and $K(x)$ does not contain any variable x_{ij} if y_j is not in $K(y)$. We denote as K a generic *global kernel*. Global kernel K is given by the union of the individual kernels $K(y)$ and $K(x)$. The first restricted BILP problem solved considers the variables in the initial global kernel only, as detailed later. We denote as $K \bigcup A$ the set of variables on which a generic subsequent BILP problem is solved that contains the current global kernel K and a set of additional variables A.

In the first phase of the Kernel Search, referred to as the *initialization phase*, problem LP($J, I \times J$) is solved. Let $(\mathbf{y}^{LP}, \mathbf{x}^{LP})$ denote its optimal solution. If $(\mathbf{y}^{LP}, \mathbf{x}^{LP})$ is integer, then it is an optimal solution to the original problem and the Kernel Search stops. Otherwise, the optimal solution of LP($J, I \times J$) provides a lower bound on the optimal cost and information that can be used to identify the promising variables. The Kernel Search continues sorting all the facilities in J. Even if alternative sorting criteria could be used, our experience showed that the following is the most effective one. Let $\hat{c}(y_j)$ be the reduced cost of variable y_j in the optimal solution of LP($J, I \times J$). The facilities are then sorted in non-increasing order of the total demand they serve in the optimal solution of LP($J, I \times J$), i.e. $\sum_{i \in I} d_i x_{ij}^{LP}$, and for those facilities not selected in the optimal solution, i.e. all j such that $y_j^{LP} = 0$, in non-decreasing order of $\hat{c}(y_j)$ values. This sorting criterion aims at creating a list L where the facilities that are most likely chosen in an optimal solution of the original problem are ranked in the first positions, whereas the least likely are in the last positions.

Afterwards, for each facility a subset of the customers is selected as follows. Let $\hat{c}(x_{ij})$ be the reduced cost of variable x_{ij} in the optimal solution of LP($J, I \times J$). The subset of customers associated with facility j is chosen by setting a threshold γ and then selecting all pairs (i,j) such that the reduced cost $\hat{c}(x_{ij})$ does not exceed γ.

Subsequently, global kernel K is initialized. Set $K(y)$ initially includes the variables corresponding to the first m facilities in list L, where m is a given parameter. Set $K(x)$ is composed, for each facility belonging to $K(y)$, of the variables x_{ij} associated with the corresponding subset of customers. The remaining $|J| - m$ facilities are partitioned into NB sets denoted as $B(y)_h,\ h = 1, \ldots, NB$. Given this partition, a sequence of *individual buckets* denoted as $\{B(y)_h\}_{h=1,\ldots,NB}$ is created for the vector $\mathbf{y}$. Particularly, we choose a priori parameter *lbuck* and then create a sequence of disjoint buckets, all with cardinality equal to *lbuck* except possibly the last one that may contain a smaller number of facilities. The number of buckets generated with this procedure is $NB := \left\lceil \frac{|J|-m}{lbuck} \right\rceil$. A sequence of NB individual buckets for the vector $\mathbf{x}$, denoted as $\{B(x)_h\}_{h=1,\ldots,NB}$, is created similarly to what has been done for the individual kernels. Hence, set $B(x)_h$ is composed, for each facility belonging to $B(y)_h$, of the variables x_{ij} associated with the corresponding subset of customers.

An example of the initial individual kernels and the sequences of individual buckets for a SSCFLP instance with 5 facilities and 6 customers is shown in Fig. 1. Note that the x_{ij} variables in the gray area are not considered by the Kernel Search.

As stopping criterion of the Kernel Search we adopt the maximum number of restricted problems to solve. Hence, we introduce parameter $\overline{NB} \leqslant NB$ representing the number of buckets to be analyzed by the Kernel Search.

Upper bound z^H is initialized by solving problem BILP(K) restricted to the variables corresponding to the initial global kernel. Before solving problem BILP(K) it is checked whether each customer in I is linked to at least one facility in $K(y)$. If the check fails, set $K(x)$ is modified by adding, for each facility in $K(y)$, the variables x_{ij} corresponding to each customer not served by any facility.

In the second phase, referred to as the *solution phase*, a sequence of $\overline{NB}$ restricted problems is solved. Specifically, at each iteration h, where $h = 1, \ldots, \overline{NB}$, set $K \bigcup A$ is created by adding the variables belonging to the current individual buckets $B(y)_h$ and $B(x)_h$ to the variables in the current global kernel K. The aforementioned procedure to check that each customer is linked to at least one facility in the restricted problem is run. Then, the restricted BILP($K \bigcup A$) problem is solved after the introduction of two supplementary constraints aiming at reducing computing times. The constraints set a cut-off value to the objective function and constrain the solution of the restricted problem to include at least one facility from the corresponding current individual bucket. In fact, we are interested in those solutions that improve upon the current upper bound z^H and include at least one new facility from the current

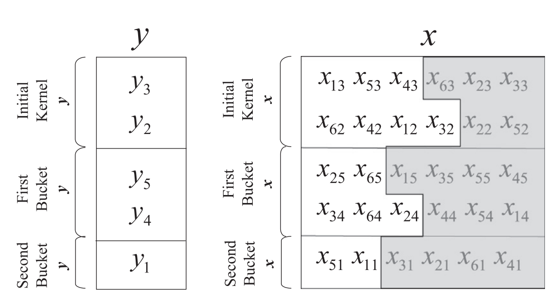

Fig. 1. Individual kernels and buckets for the SSCFLP.

individual bucket $B(y)_h$. If the restricted BILP($K \bigcup A$) problem is feasible, then its optimal solution improves the best found so far. In this case, at least one facility from its current individual bucket is selected in the optimal solution of BILP($K \bigcup A$), i.e. new promising variables have been identified, and the current individual kernel $K(y)$ is modified to include them. The set including these facilities is denoted as $B(y)_h^+$. Conversely, if a facility in the current individual kernel has not been selected in the optimal solution of BILP($K \bigcup A$) and also in p of the restricted problems solved since it has been added to the kernel, where p is a given parameter, then that facility is assumed to be no longer promising and is removed from its individual kernel. The set including these facilities is denoted as $B(y)_h^-$. Thus, at the end of iteration h the current individual kernel $K(y)$ is given by the individual kernel at the beginning of iteration h plus the facilities in $B(y)_h^+$, minus the facilities in $B(y)_h^-$. The current individual kernel $K(x)$ is updated similarly. If a new facility is added to $K(y)$, then the corresponding subset of customers is added to the current individual kernel $K(x)$. Conversely, when a facility is removed from $K(y)$, then the corresponding subset of customers is removed from the current individual kernel $K(x)$. When the last bucket (i.e., bucket $\overline{NB}$) has been analyzed, the algorithm ends.

The Kernel Search fails in two cases. The first case is when no feasible solution for LP($J, I \times J$) is found. This implies that no feasible solution for the original problem exists either. The second case is when no feasible solution for any restricted BILP problem is found. This means that either no feasible solution for the original problem exists or that the Kernel Search has not been able to find any of them. The latter case might occur when the capacity of the facilities included in each restricted problem is not sufficient to fulfill the demand of all the customers.

3.2. The Kernel Search with variable fixing to 1

The efficiency of the Kernel Search can be improved by means of variable fixing. Here, we see the case where some binary variables are fixed to 1. Once LP($J, I \times J$) is solved, set J is partitioned into two subsets $J(1)$ and $J \setminus J(1)$. Set $J(1)$ contains all the indices of variables y_j that took value 1 in the optimal solution of LP($J, I \times J$), whereas $J \setminus J(1)$ contains all the remaining indices. This first variant, denoted in the following as *Kernel Search(1)*, is based on the idea of reducing the number of y_j variables in all the restricted problems solved by the Kernel Search fixing to 1 those that took value 1 in the optimal solution of the linear relaxation. Hence, the sequence of restricted problems is obtained from (1)–(6) by fixing to 1 each y_j with $j \in J(1)$.

The rest of the algorithm described in the previous section remains unchanged. Note that even if $J(1) = J$, i.e. all y_j variables are fixed, the former problem is still $\mathcal{NP}$-hard as it reduces to a particular case of the GAP.

3.3. The Kernel Search with Variable Fixing to 0 and 1

In the second variant, referred to as *Kernel Search(0–1)*, once LP($J, I \times J$) is solved, set J is partitioned into three subsets: $J(1)$, $J(0)$ and $\mathcal{J}$. The indices of variables y_j that took value 0 in the optimal solution of the linear relaxation compose set $J(0)$, whereas set $\mathcal{J}$ includes all the indices not included either in $J(1)$ or in $J(0)$. In addition to fixing to 1 the y_j variables with indices in $J(1)$, in the Kernel Search(0–1) the y_j variables with indices in set $J(0)$ are set to 0. Therefore, the sequence of restricted problems is obtained from (1)–(6) by fixing to 1 each y_j with $j \in J(1)$, and to 0 each y_j variable with $j \in J(0)$.

Due to constraints (2) in the SSCFLP model, fixing one variable y_j to 0 implies a remarkable reduction in the number of binary variables of the optimization model, as the $|I|$ assignment variables x_{ij} associated with each of those facilities are forced to take value 0. Even if $\mathcal{J} = \emptyset$, i.e. all y_j variables are fixed either to 1 or to 0, the former problem remains $\mathcal{NP}$-hard as it reduces to a particular case of the GAP.

4. Experimental analysis

This section is devoted to presentation and discussion of the computational experiments. They were conducted on a PC Intel Xeon with 3.33 gigahertz 64-bit processor, 12.0 gigabyte of RAM and Windows 7 64-bit as Operating System. The algorithms were implemented in Java. The LP and BILP problems were solved with CPLEX 12.2. After preliminary experiments, we set the following CPLEX parameters. We chose the sifting algorithm as LP optimizer (parameter RootAlg), the pseudo reduced costs to drive the selection of the variable to branch on at a node (parameter VarSel), and we set the emphasis on finding optimal solutions with less effort applied to finding feasible solutions early (parameter MIPEmphasis). We decided to not perform probing (parameter Probe) in order to save computing time, and to generate flow cover (parameter FlowCovers) and mixed integer rounding cuts (parameter MIRCuts) moderately. All the other CPLEX parameters were set to their default values.

In order to provide further insights into the effectiveness of the proposed heuristics, we solved all the instances without an optimal solution available in the literature with CPLEX 12.2 using 3 different settings. We considered CPLEX with all the parameters set to their default values with the exception of parameter MIPEmphasis that was set to feasibility (this variant is referred to as *CPLEX Setting-A*, henceforth). *CPLEX Setting-B* has the same parameter settings of CPLEX Setting-A with the exception that the RINS heuristic is applied every 20 nodes (i.e., parameter RINSHeur was set equal to 20). *CPLEX Setting-C* differs from CPLEX Setting-A as the local branching heuristic is turned on (parameter LBHeur). The latter two settings are chosen as they correspond to two approaches available in CPLEX that, as mentioned in Section 1, use variable fixing. Finally, we set a time limit equal to 7200 seconds to the solution of each instance for any CPLEX setting.

The present section is organized as follows. In Section 4.1 we briefly describe the testing environment we used. In Section 4.2 we report some issues related to the implementation of the heuristics, whereas in Section 4.3 we motivate the introduction of the variants by means of some examples. Finally, Section 4.4 provides the computational results.

4.1. Testing environment

The proposed heuristics were tested on four groups of instances: three groups are composed of benchmark instances available in the literature, and one group is formed of benchmark instances taken from the literature on the CFLP. Altogether, the heuristics we implemented were tested on 227 instances, ranging from small-scale (i.e., 16 facilities and 50 customers) to very large-scale (i.e., 1000 facilities and 1000 customers). A summary of the instances tested in this paper is shown in Table 1. Instances in the same group are further classified into sub-groups according to their size. In column "References", we report the main references where optimal solutions (Opt.) or best upper bounds (Heur.) can be found.

The first group of instances is composed of a subset of those belonging to the OR-Library, publicly available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/capinfo.html. These instances were originally proposed for the CFLP, and some of them are not feasible when the single source constraint is introduced. Hence, we restricted our experiments to the set of feasible instances that is

Table 1
The benchmark instances tested.

Instances		# Inst.	$\lvert J\rvert$	$\lvert I\rvert$	References
OR-Library	OR1	8	16	50	
	OR2	8	25	50	Opt.: CPLEX 12.2
	OR3	8	50	50	
	OR4	12	100	1000	Opt.: CPLEX 12.2 Heur.: Ahuja et al. (2004), Chen and Ting (2008)
Holmberg	H1	12	10	50	Opt.: Holmberg et al. (1999),
	H2	12	20	50	Yang et al. (2012)
	H3	16	30	150	
	H4	15	10–30	70–100	Heur.: Ahuja et al. (2004),
	H5	16	30	200	Chen and Ting (2008)
Yang	Y1	5	30	200	Opt.: Yang et al. (2012)
	Y2	5	60	200	
	Y3	5	60	300	
	Y4	5	80	400	
TBED1	TB1	20	300	300	
	TB2	20	300	1500	
	TB3	20	500	500	Opt.: CPLEX 12.2
	TB4	20	700	700	
	TB5	20	1000	1000	

composed of 24 small-scale (data sets from OR1 to OR3) and 12 large-scale instances (data set OR4). Even though the 24 small-scale instances can be solved by CPLEX in negligible computing time, we performed experiments on these in order to have a large set of instances with known optimal solutions to assess the performance of the heuristics. Data set OR4 includes instances consisting of 100 facilities and 1000 customers that are, to the best our knowledge, the largest-scale instances tested so far in the literature. Best upper bounds can be found in Ahuja et al. (2004) and Chen and Ting (2008). Optimal solutions are not available from the literature for these instances but we solved them to optimality within few minutes, by using CPLEX.

The second group of benchmark instances comprises five sets of small/medium-scale instances randomly generated by Holmberg et al. (1999), each with different size and ratio between the total capacity of the facilities and the total demand of the customers. Optimal solutions for this group of instances are reported in Holmberg et al. (1999) and Yang et al. (2012), whereas heuristic solutions are published in Ahuja et al. (2004) and Chen and Ting (2008).

The third group is composed of 20 medium/large-scale instances randomly generated by Yang et al. (2012). They are classified into four sub-groups which differ in problem size and in the ratio of the total capacity of the facilities over the total demand of the customers. Optimal solution values can be found in Yang et al. (2012).

The fourth group considers the 100 instances in TBED1 tested in Avella and Boccia (2009) for the CFLP and publicly available at http://www.ing.unisannio.it/boccia/CFLP.htm. All of them turned out to be feasible for the SSCFLP. These are very large-scale instances ranging from 300 to 1000 facilities and from 300 to 1000 customers. As these instances have never been tested before for the SSCFLP, we solved them by means of CPLEX with the 3 settings mentioned above. Whenever CPLEX found an optimal solution, the latter was used to measure the quality of the solutions found by the heuristics. In all the remaining cases, the heuristics are validated using the optimal solutions reported in Avella and Boccia (2009) for the CFLP as lower bounds.

The four groups of instances, along with the optimal solution values, when available, or the best lower and upper bounds, are publicly available at the web page https://sites.google.com/site/orbrescia/instances/instances_sscflp.

4.2. Some implementation issues

The implementation of a Kernel Search framework involves an accurate calibration of its parameters. In Guastaroba and Speranza (2012b) a thorough analysis and computational comparison of different parameter settings is provided. In this paper, after extensive preliminary experiments and based upon the results reported in Guastaroba and Speranza (2012b), we made the following choices. All the settings concern both the Kernel Search algorithm and the Kernel Search(1) and Kernel Search(0–1) variants.

Set $K(y)$ is initialized selecting all the facilities with positive value in the optimal solution of LP($J, I \times J$) (i.e., parameter m). The subset of customers associated with each facility is selected as follows. Once LP($J, I \times J$) is solved, parameter γ is set equal to the median of the reduced costs of variables x_{ij} associated with the facilities composing the initial individual kernel $K(y)$, i.e. we set $\gamma := \text{median}\{\hat{c}(x_{ij}) | i \in I, j \in K(y)\}$. Then, all the customers such that $\hat{c}(x_{ij}) \leqslant \gamma$ compose the subset associated with facility j. After some preliminary experiments, we chose to adopt the median instead of the average of the reduced costs used in Guastaroba and Speranza (2012b) since, especially for the medium and large-scale instances of the SSCFLP, we realized that the distributions of the reduced costs were strongly skewed with some extreme values that affected the computation of the average. This led to an over-estimation of parameter γ and, as a consequence, an overly large number of variables x_{ij} considered in the restricted problems. Each individual bucket $B(y)_h$ has a cardinality equal to the number of facilities used to initialize set $K(y)$, i.e. we set $lbuck := m$. Therefore, the number of individual buckets built is equal to $NB := \left\lceil \frac{|J|-m}{m} \right\rceil$ and we decided to analyze all the buckets generated, i.e. we set $\overline{NB} := NB$. Finally, we set parameter p equal to 2.

During some preliminary tests conducted on the large-scale instances, we observed that often CPLEX quickly found the optimal (or a slightly sub-optimal) solution of the restricted problems and spent excessive computing efforts trying to prove the optimality of the solution at hand. Hence, we set a time limit equal to 900 seconds for the solution of each restricted problem. If CPLEX does not terminate within the allowed computing time, then the best solution found so far, if any, is used to update the individual kernels and the upper bound. The maximum computing time constraint is set on the solution of the restricted problems, only. No time limit is introduced for the solution of LP($J, I \times J$).

There are some issues related to the solution of LP($J, I \times J$) which deserve particular notice, since they may have a substantial impact on computing times. On the one hand, the presence of the redundant constraints (4) in the SSCFLP formulation (1)–(6) does yield a much tighter linear relaxation. Indeed, we found that for some of the small-scale instances in data sets OR1–OR3 the optimal solution of the linear relaxation was integer if constraints (4) were introduced. On the other hand, the number of constraints (4) to be included in the SSCFLP model increases quite rapidly as the size of the instance grows, leading to linear programming problems that CPLEX cannot solve within reasonable computing times. Therefore, we decided as follows. In our computational experiments, we solved LP($J, I \times J$) including constraints (4) for the OR-Library, Holmberg and Yang instances, whereas we did not include them for the solution of the TBED1 instances. The solution of any restricted BILP problem does include constraints (4), even for the TBED1 instances. In fact, given the parameter settings mentioned above, the size of all the restricted problems was manageable, thus not requiring the removal of constraints (4).

4.3. Motivations of the variants

In this section we report part of the preliminary analysis we conducted when designing the two variants described in Sections 3.2 and 3.3. The goal of the analysis was to identify a relationship between the optimal solution of the linear relaxation and the optimal solution of the original BILP problem. To this aim we randomly selected 10 instances in the Holmberg data set and compared the optimal values for variables y_j in the BILP problem and in its linear relaxations. Particular attention was paid to determine a relationship between those variables y_j that took value 0 or 1 in the linear relaxation and their values in the optimal solution of the original problem. The findings of this analysis are reported in Table 2.

In the fourth column of Table 2 we report, for each instance, the number of times a binary variable y_j that took value 1 in the optimal solution of the linear relaxation took a different value (i.e., took value 0) in the optimal solution of the original problem. Similarly, in the last column we show, for each instance, the number of times a variable y_j whose optimal value was 0 in the linear relaxation took value 1 in the optimal solution of the original problem. For only few variables y_j we found a difference between the optimal solution of the linear relaxation and that of the original problem. This suggests that the variants based on variable fixing may significantly improve the efficiency of the procedure with, hopefully, minor worsening of the solution quality.

Table 2
A preliminary analysis that motivates the variants.

Instances	$\|J\|$	$\|I\|$	# $y_j^{LP} = 1$ $y_j^* = 0$	# $y_j^{LP} = 0$ $y_j^* = 1$
p4	10	50	0	0
p19	20	50	1	1
p32	30	150	0	0
p45	20	80	0	0
p47	10	90	0	0
p49	30	70	3	2
p52	10	100	0	0
p53	20	100	0	0
p66	30	200	0	0
p71	30	200	0	0

4.4. Computational results

This section is devoted to the illustration and comment of the computational results. As the goal of this paper is to provide a tool for solving medium/large-scale instances that cannot be solved easily to optimality, the main part of this section concentrates on the medium/large-scale instances. Whenever possible, the quality of the solutions found by the heuristics proposed in this paper was validated by direct comparison with the optimal solution values, denoted as z^*, either available in the literature or found with CPLEX. When commenting the instances solved also by other heuristics, we compared the performance of the Kernel Search (and the two variants) with the upper bounds, denoted as z^{UB}, found by the best performing heuristic available in the literature. Finally, when optimal solutions were not available in the literature and could not be found with CPLEX, we used the optimal solution values for the CFLP as lower bounds, denoted as z^{LB}, to validate the performance of our heuristics.

In Tables 3, 4, 6 and 7 we provide a summary of the computational results for each group of instances. We decided to report in the present section only the detailed computational results for the OR4 data set (see Table 5) since it comprises benchmark instances for the SSCFLP without, to the best of our knowledge, optimal solutions previously published in the literature. All the detailed computational results are reported in Tables 8–11 in Appendix A.

For the OR-Library and Holmberg data sets we report only the performance of the Kernel Search, given the negligible computing times reported. For all the remaining instances the performance of the Kernel Search(1) and Kernel Search(0–1) is also shown.

In Table 3 we report the computational results for data sets OR1, OR2 and OR3. The performance of the Kernel Search is evaluated comparing the solution values found by the heuristic, denoted as z^H, with the optimal solution values computed with CPLEX. For each data set we report statistic *Opt. Gap %* that refers to the average error with respect to the optimal solution value. The error for each instance is computed as $100(z^H - z^*)/z^*$, and then averaged over all the instances belonging to the same data set to obtain statistic *Opt. Gap %*. Statistic *Worst Gap %* shows the worst error computed out of all the instances in the data set. Finally, statistic *CPU (seconds)* shows the average computating time in seconds. The results for these data sets show that the Kernel Search found the optimal solutions for these small-scale instances in negligible computing time (computing times for CPLEX are similar).

Table 4 compares the performance of the Kernel Search with that of the VLNS proposed in Ahuja et al. (2004) and the LH-ACS introduced in Chen and Ting (2008) for the OR4 data set and the Holmberg instances. For each data set we report the best implementation of the VLNS proposed in Ahuja et al. (2004). For each heuristic we report the number of times it found the optimal solution in column *# Opt*. For the Kernel Search we also report statistic *# Impr.* that counts, for each data set, the number of times that the Kernel Search found a better solution than the best upper bound available in the literature, i.e. the best solution among those found by the different VLNS implementations in Ahuja et al. (2004) and the LH-ACS in Chen and Ting (2008). The figures reported in Table 4 show that the Kernel Search outperforms the other heuristics.

Table 3
Average statistics for OR1–OR3 data sets.

Data Set	# Inst.	*Kernel Search*		
		Opt. Gap %	Worst Gap %	CPU (second)
OR1	8	0.00	0.00	0.286
OR2	8	0.00	0.00	0.385
OR3	8	0.00	0.00	0.617

Table 4
Average statistics for OR4 and Holmberg data sets.

Data Set	# Inst.	*VLNS*			*LH-ACS.*[d]			*Kernel Search*				
		# Opt.	Opt. Gap %	Worst Gap %	# Opt.	Opt. Gap %	Worst Gap %	# Opt.	# Impr.	Opt. Gap %	Worst Gap %	CPU (second)
OR4	12	3[a]	0.08	0.33	4	0.05	0.19	12	8	0.00	0.00	34.665
H1	12	12[b]	0.00	0.00	12	0.00	0.00	12	0	0.00	0.00	0.321
H2	12	12[b]	0.00	0.00	12	0.00	0.00	12	0	0.00	0.00	0.379
H3	16	10[b]	0.07	0.42	13	0.02	0.18	16	3	0.00	0.00	2.434
H4	15	13[b]	0.01	0.15	12	0.06	0.74	15	2	0.00	0.00	0.536
H5	16	11[c]	0.02	0.14	11	0.03	0.19	16	3	0.00	0.00	2.319

[a] *bfs* in Ahuja et al. (2004).
[b] *bs* in Ahuja et al. (2004).
[c] *dfs* in Ahuja et al. (2004).
[d] Chen and Ting (2008).

Table 5
A comparison with CPLEX on the OR4 data set.

Instance Details				*Kernel Search*		*CPLEX Setting-A*		*CPLEX Setting-B*		*CPLEX Setting-C*	
Name	$\lvert J\rvert$	$\lvert I\rvert$	z^*	Opt. Gap %	CPU (second)	Opt. Gap %	CPU (second)	Opt. Gap %	CPU (second)	Opt. Gap %	CPU (second)
capa1	100	1000	19241056.93	0.00	32.212	0.00	60.294	0.00	61.245	0.00	55.739
capa2	100	1000	18438329.78	0.00	20.837	0.00	51.464	0.00	52.432	0.00	54.366
capa3	100	1000	17765201.95	0.00	20.003	0.00	32.885	0.00	32.963	0.00	35.334
capa4	100	1000	17160612.23	0.00	14.793	0.00	23.619	0.00	23.790	0.00	23.416
capb1	100	1000	13657464.23	0.00	21.987	0.00	27.268	0.00	27.737	0.00	27.737
capb2	100	1000	13362529.34	0.00	75.761	0.00	328.787	0.00	293.202	0.00	321.641
capb3	100	1000	13199213.19	0.00	64.252	0.00	356.195	0.00	310.254	0.00	310.254
capb4	100	1000	13083203.74	0.00	43.692	0.00	186.156	0.00	177.809	0.00	189.431
capc1	100	1000	11647410.50	0.00	41.341	0.00	149.698	0.00	142.600	0.00	161.710
capc2	100	1000	11570437.68	0.00	36.507	0.00	73.040	0.00	73.273	0.00	76.565
capc3	100	1000	11519169.78	0.00	25.075	0.00	37.330	0.00	39.250	0.00	40.685
capc4	100	1000	11505861.86	0.00	19.523	0.00	22.558	0.00	22.448	0.00	25.132
Avg.				0.00	34.665	0.00	112.441	0.00	104.750	0.00	110.168

Indeed, the Kernel Search found the optimal solution for all the OR4 and Holmberg instances. The number of improvements achieved by the Kernel Search with respect to the best known upper bound is particularly remarkable for the OR4 data set, where the Kernel Search improves the upper bound for 8 out of the 12 instances in the data set. A smaller number of improvements is achieved for the Holmberg instances (8 improvements out of 71 instances) as for the remaining 63 instances the best upper bound corresponds to the optimal solution value. Average computing times are negligible for all the Holmberg instances, whereas it took less than 35 seconds, on average, to solve the instances in the OR4 data set. A thorough validation of the performance of the Kernel Search is provided for the OR4 data set in Table 5. We found that CPLEX with any of the tested settings could solve to optimality the instances in the OR4 data in less than 2 minutes, on average. The results do not show remarkable differences in terms of computing time among the different settings. Note that the Kernel Search found the optimal solution for the instances in the OR4 data set in, on average, slightly more than half a minute of computing time.

As the size of the instances in both the Yang and the TBED1 data sets is large, in Tables 6 and 7 we report also the performance of the Kernel Search(1) and Kernel Search(0–1) variants. Specifically, Table 6 reports the average results for the Yang data set. The Kernel Search found the optimal solution for 18 out of 20 instances. The error reported for the 2 remaining instances is approximately equal to 0.01% (see Table 10). Computing times are, on average, less than 17 minutes and 8 instances were solved within 1 minute and a half (see Table 10). We report three further statistics for the two variants. As the optimal solution for these instances is known, for each variant in columns *# Gap < 0.1%*, *# Gap < 0.5%* and *# Gap < 1%* we show the number of times it found a solution whose error with

Table 6
Average statistics for Yang data set.

Data Set	# Inst.	*Kernel Search*				*Kernel Search*(1)						
		# Opt.	Opt. Gap %	Worst Gap %	CPU (second)	# Opt.	Opt. Gap %	# Gap < 0.1%	# Gap < 0.5%	# Gap < 1%	Worst Gap %	CPU (second)
Y1	5	5	0.00	0.00	411.282	3	0.27	3	3	5	0.74	186.297
Y2	5	4	0.00	0.01	1640.424	2	0.33	3	4	4	1.49	1187.597
Y3	5	4	0.00	0.01	597.056	2	0.18	3	4	5	0.70	558.075
Y4	5	5	0.00	0.00	1409.110	3	0.01	5	5	5	0.04	1149.781

Data Set	# Inst.	*Kernel Search*(0–1)						
		# Opt.	Opt. Gap %	# Gap < 0.1%	# Gap < 0.5%	# Gap < 1%	Worst Gap %	CPU (second)
Y1	5	1	1.02	1	1	3	2.02	44.273
Y2	5	1	0.67	3	4	4	2.96	368.735
Y3	5	1	1.61	1	2	2	3.24	184.533
Y4	5	0	0.51	3	3	3	1.46	369.761

Table 7
Average statistics for TBED1 data set.

Data Set	# Inst.	*Kernel Search*			*Kernel Search*(1)			*Kernel Search*(0–1)		
		LB/Opt. Gap %	Worst Gap %	CPU (second)	LB/Opt. Gap %	Worst Gap %	CPU (second)	LB/Opt. Gap %	Worst Gap %	CPU (second)
TB1	20	0.56	2.22	2206.957	0.59	2.29	2110.348	0.78	2.29	408.213
TB2	20	0.00	0.00	334.705	0.00	0.00	299.765	0.21	0.74	186.527
TB3	20	0.66	2.04	4190.283	0.71	2.07	4050.817	0.79	2.09	673.563
TB4	20	0.90	2.29	5244.693	0.91	2.34	5169.958	1.00	2.70	854.165
TB5	20	1.07	3.11	6533.149	1.02	2.48	6509.264	1.10	2.67	968.126
		CPLEX Setting-A			*CPLEX Setting-B*			*CPLEX Setting-C*		
TB1	20	0.59	2.23	3159.606	0.56	2.22	3038.605	0.56	2.23	3231.517
TB2	20	0.00	0.00	186.670	0.00	0.00	184.328	0.00	0.00	200.760
TB3	20	0.74	2.15	6007.624	0.71	2.13	5764.971	0.83	2.42	5876.130
TB4	20	1.24	3.35	6865.528	1.10	2.99	6642.240	1.61	3.88	7072.458
TB5	20	2.44	6.47	7269.602	1.94	4.56	7244.478	2.95	8.15	7451.382

Table 8
Detailed computational results for OR1–OR3 data sets.

Instance Details				*Kernel Search*	
Name	$\|J\|$	$\|I\|$	z^*	Opt. Gap %	CPU (second)
cap61	16	50	932615.8	0.00	0.271
cap62	16	50	977799.4	0.00	0.270
cap63	16	50	1014100	0.00	0.280
cap64	16	50	1053197	0.00	0.417
cap71	16	50	932615.7	0.00	0.258
cap72	16	50	977799.4	0.00	0.254
cap73	16	50	1010641	0.00	0.270
cap74	16	50	1034977	0.00	0.270
cap91	25	50	796648.4	0.00	0.317
cap92	25	50	858109.3	0.00	0.359
cap93	25	50	900760.1	0.00	0.397
cap94	25	50	950608.4	0.00	0.640
cap101	25	50	796648.4	0.00	0.308
cap102	25	50	854704.2	0.00	0.317
cap103	25	50	893782.1	0.00	0.324
cap104	25	50	928941.8	0.00	0.419
cap121	50	50	793439.6	0.00	0.540
cap122	50	50	854900.5	0.00	0.514
cap123	50	50	898266.1	0.00	0.736
cap124	50	50	950608.4	0.00	0.940
cap131	50	50	793439.6	0.00	0.546
cap132	50	50	851495.3	0.00	0.448
cap133	50	50	893076.7	0.00	0.589
cap134	50	50	928941.8	0.00	0.625
Avg.				0.00	0.430

respect to the optimal solution value was smaller than 0.1%, 0.5% and 1%, respectively. As expected, the quality of the solutions found by the Kernel Search(1) variant is slightly worse than the Kernel Search, but the improvements in terms of computing times are remarkable. Specifically, the Kernel Search(1) variant found the optimal solution for half of the instances composing the data set. Moreover, the average optimality gap was always smaller than 0.34% and statistic *Opt. Gap %* took a value smaller than 0.1% for 14 instances out of 20, a value smaller than 0.5% for 16 instances out of 20 and a value smaller than 1% for 19 instances out of 20 (see Table 10 for further details). Statistic *Opt. Gap%* for the only instance which reported an error larger than 1% was equal to 1.49%. On the other hand, savings in terms of computing times are valuable. Indeed, the average computing time spent by the Kernel Search(1) variant is approximately 25% smaller than the processing time required by the Kernel Search. More importantly, in several cases the Kernel Search(1) variant found the optimal solution (or a near-optimal) in much less computing times than the Kernel Search. For example, see the figures related to instances 30_200_4, 60_200_4, 60_300_4 and 80_400_2 in Table 10. The

Table 9
Detailed computational results for OR4 and Holmberg data sets.

Instance Details					*Kernel Search*	
Name	$\|J\|$	$\|I\|$	z^*	z^{UB}[a] %	Opt. Gap %	CPU (second)
capa1	100	1000	19241056.93	0.01	0.00	32.212
capa2	100	1000	18438329.78	0.01	0.00	20.837
capa3	100	1000	17765201.95	0.00	0.00	20.003
capa4	100	1000	17160612.23	0.00	0.00	14.793
capb1	100	1000	13657464.23	0.01	0.00	21.987
capb2	100	1000	13362529.34	0.02	0.00	75.761
capb3	100	1000	13199213.19	0.16	0.00	64.252
capb4	100	1000	13083203.74	0.02	0.00	43.692
capc1	100	1000	11647410.50	0.03	0.00	41.341
capc2	100	1000	11570437.68	0.00	0.00	36.507
capc3	100	1000	11519169.78	0.04	0.00	25.075
capc4	100	1000	11505861.86	0.00	0.00	19.523
p1	10	50	8848	0.00	0.00	0.291
p2	10	50	7913	0.00	0.00	0.361
p3	10	50	9314	0.00	0.00	0.287
p4	10	50	10714	0.00	0.00	0.268
p5	10	50	8838	0.00	0.00	0.274
p6	10	50	7777	0.00	0.00	0.347
p7	10	50	9488	0.00	0.00	0.344
p8	10	50	11088	0.00	0.00	0.313
p9	10	50	8462	0.00	0.00	0.309
p10	10	50	7617	0.00	0.00	0.250
p11	10	50	8932	0.00	0.00	0.399
p12	10	50	10132	0.00	0.00	0.409
p13	20	50	8252	0.00	0.00	0.454
p14	20	50	7137	0.00	0.00	0.336
p15	20	50	8808	0.00	0.00	0.352
p16	20	50	10408	0.00	0.00	0.417
p17	20	50	8227	0.00	0.00	0.327
p18	20	50	7125	0.00	0.00	0.337
p19	20	50	8886	0.00	0.00	0.569
p20	20	50	10486	0.00	0.00	0.392
p21	20	50	8068	0.00	0.00	0.352
p22	20	50	7092	0.00	0.00	0.308
p23	20	50	8746	0.00	0.00	0.341
p24	20	50	10273	0.00	0.00	0.362
p25	30	150	11630	0.00	0.00	0.904
p26	30	150	10771	0.00	0.00	0.952
p27	30	150	12322	0.00	0.00	1.141
p28	30	150	13722	0.00	0.00	1.297
p29	30	150	12371	0.00	0.00	1.224
p30	30	150	11331	0.18	0.00	8.754
p31	30	150	13331	0.10	0.00	8.360
p32	30	150	15331	0.07	0.00	10.088
p33	30	150	11629	0.00	0.00	0.830
p34	30	150	10632	0.00	0.00	0.789
p35	30	150	12232	0.00	0.00	0.744
p36	30	150	13832	0.00	0.00	0.791

Table 9 (*continued*)

Instance Details					*Kernel Search*	
Name	$\|J\|$	$\|I\|$	z^*	$z^{UB[a]}$ %	Opt. Gap %	CPU (second)
p37	30	150	11258	0.00	0.00	0.667
p38	30	150	10551	0.00	0.00	0.816
p39	30	150	11824	0.00	0.00	0.954
p40	30	150	13024	0.00	0.00	0.628
p41	10	90	6589	0.00	0.00	0.459
p42	20	80	5663	0.00	0.00	0.581
p43	30	70	5214	0.00	0.00	0.493
p44	10	90	7028	0.00	0.00	0.466
p45	20	80	6251	0.00	0.00	0.689
p46	30	70	5651	0.00	0.00	0.594
p47	10	90	6228	0.00	0.00	0.273
p48	20	80	5596	0.00	0.00	0.808
p49	30	70	5302	0.00	0.00	0.700
p50	10	100	8741	0.00	0.00	0.442
p51	20	100	7414	0.15	0.00	0.932
p52	10	100	9178	0.02	0.00	0.315
p53	20	100	8531	0.00	0.00	0.551
p54	10	100	8777	0.00	0.00	0.273
p55	20	100	7654	0.00	0.00	0.463
p56	30	200	21103	0.08	0.00	1.921
p57	30	200	26039	0.14	0.00	10.146
p58	30	200	37239	0.00	0.00	6.236
p59	30	200	27282	0.00	0.00	3.072
p60	30	200	20534	0.00	0.00	0.889
p61	30	200	24454	0.00	0.00	1.203
p62	30	200	32643	0.00	0.00	2.317
p63	30	200	25105	0.00	0.00	1.173
p64	30	200	20530	0.00	0.00	0.838
p65	30	200	24445	0.00	0.00	0.883
p66	30	200	31415	0.04	0.00	2.176
p67	30	200	24848	0.00	0.00	1.239
p68	30	200	20538	0.00	0.00	1.030
p69	30	200	24532	0.00	0.00	1.223
p70	30	200	32321	0.00	0.00	1.675
p71	30	200	25540	0.00	0.00	1.084
Avg.				0.01	0.00	6.126

[a] Best Upper Bound among Ahuja et al. (2004) and Chen and Ting (2008).

trade-off between the efficiency of the procedure and the quality of the solution found is even more evident observing the results reported for the Kernel Search(0–1) variant. With respect to both Kernel Search and Kernel Search(1), the deteriorations in terms of solution quality are noteworthy, but the algorithm is quite faster. Particularly, the Kernel Search(0–1) variant found the optimal solution for only 3 instances out of 20. However, the average error is less than 1% (see Table 10), statistic *Opt. Gap %* is never larger than 3.25% and for 12 instances out of 20 it is smaller than 1%. Computing times are, on average, 76% and 68% smaller than those reported by the Kernel Search and the Kernel Search(1), respectively. We also highlight that for some instances the Kernel Search(0–1) variant found a solution much faster than the other two heuristics with almost negligible deteriorations in terms of solution quality (see the figures related to instances 60_200_2, 60_200_4, 60_200_5, 60_300_2, and 80_400_5 in Table 10).

In Table 7 we report the average statistics for the TBED1 data set. Detailed computational results are shown in Table 11. As for this set of instances neither optimal solutions nor upper bounds are available in the literature, the performance of the heuristics is assessed as follows. 43 out of 100 instances were solved to proven optimality within the computing time limit of 7200 seconds by CPLEX (with one or more of the tested settings). They are denoted with the symbol (*) in Table 11. For each of the latter instances we computed statistic *Opt. Gap %*. For all the remaining instances, we computed the gap with respect to lower bound z^{LB} as $100(z^H - z^{LB})/z^{LB}$. We then averaged over all the instances in the same data set to calculate statistic *LB/Opt. Gap %* reported in Table 7. The Kernel Search found the optimal solution for 40 out of the 43 instances with proven optimal solution (see Table 11). The average error for the 3 remaining instances is roughly equal to 0.08%. Particularly, the Kernel Search has a very good performance for the instances in groups TB1 and TB2 where it always found the optimal solution, when available. The figures in Table 7 show that statistic *LB/Opt. Gap %* is, on average, smaller than 1.07% for every group of instances. Additionally, for 58 instances out of 100 the LB/Opt. gap is smaller than 0.5%, and 76 instances were solved within a 1% LB/Opt. gap (see Table 11). Average computing times are quite larger than for the previous instances, but still reasonable given the size of the instances. The performance reported by the Kernel Search(1) variant is quite sim-

Table 10
Detailed computational results for Yang data set.

Instance Details				*Kernel Search*		*Kernel Search*(1)		*Kernel Search*(0–1)	
Name	$\|J\|$	$\|I\|$	z^*	Opt. Gap %	CPU (second)	Opt. Gap %	CPU (second)	Opt. Gap %	CPU (second)
30_200_1	30	200	30181	0.00	964.378	0.00	683.796	0.61	153.863
30_200_2	30	200	28923	0.00	1021.459	0.74	227.997	1.67	38.673
30_200_3	30	200	28131	0.00	11.232	0.60	11.799	0.81	26.193
30_200_4	30	200	28152	0.00	55.520	0.00	6.084	2.02	0.936
30_200_5	30	200	27646	0.00	3.822	0.00	1.809	0.00	1.701
60_200_1	60	200	27977	0.00	1637.051	1.49	373.418	2.96	1.654
60_200_2	60	200	29704	0.01	2702.611	0.02	2702.313	0.04	900.637
60_200_3	60	200	27993	0.00	28.564	0.12	26.685	0.33	1.482
60_200_4	60	200	27691	0.00	1132.063	0.00	133.084	0.00	39.234
60_200_5	60	200	29195	0.00	2701.831	0.00	2702.486	0.01	900.668
60_300_1	60	300	35648	0.01	2238.729	0.00	2747.289	0.31	906.050
60_300_2	60	300	35474	0.00	72.368	0.00	20.499	0.00	8.034
60_300_3	60	300	33872	0.00	85.987	0.12	7.722	2.38	2.979
60_300_4	60	300	33096	0.00	577.528	0.09	11.934	3.24	2.901
60_300_5	60	300	30918	0.00	10.670	0.70	2.933	2.14	2.699
80_400_1	80	400	39318	0.00	616.419	0.04	278.663	1.46	26.769
80_400_2	80	400	37076	0.00	987.061	0.00	49.608	1.03	10.873
80_400_3	80	400	43859	0.00	2702.830	0.01	2702.252	0.03	901.307
80_400_4	80	400	37344	0.00	35.366	0.00	14.539	0.03	8.378
80_400_5	80	400	43508	0.00	2703.875	0.00	2703.844	0.01	901.479
Avg.				0.00	1014.468	0.20	770.438	0.95	241.826

Table 11
Detailed computational results for TBED1 data set.

Instance Details				*Kernel Search*		*Kernel Search*(1)		*Kernel Search*(0–1)		*CPLEX Setting-A*		*CPLEX Setting-B*		*CPLEX Setting-C*	
Name	|J|	|I|	z^{LB}/z^{*}[a]	LB/Opt. Gap %	CPU (second)	LB/Opt. Gap %	CPU (second)	LB/Opt. Gap %	CPU (second)	LB/Opt. Gap %	CPU (second)	LB/Opt. Gap %	CPU (second)	LB/Opt. Gap %	CPU (second)
i300_1	300	300	16350.66	1.71	5512.161	1.85	5545.239	1.88	904.520	1.98	7212.205	1.98	7333.183	1.73	7353.806
i300_2	300	300	15948.44	1.52	5490.640	1.85	5553.599	1.84	903.085	1.71	7284.999	1.52	7357.506	1.84	7394.008
i300_3	300	300	15474.84	1.51	5541.068	1.45	5432.474	1.47	902.587	1.78	7358.975	1.51	7337.751	1.48	7353.788
i300_4	300	300	17989.97	2.22	4602.425	2.29	4645.545	2.29	906.112	2.23	7355.811	2.22	7349.149	2.23	7267.157
i300_5	300	300	18037.61	1.88	4635.414	1.88	4609.580	1.92	904.099	1.84	7210.140	1.75	7210.677	1.84	7284.627
i300_6	300	300	11251.19	0.77	3325.762	0.67	2752.727	1.03	907.251	0.67	7206.616	0.67	7203.861	0.67	7218.179
i300_7	300	300	11392.52	0.71	2799.664	1.04	2867.857	1.04	902.430	0.68	7200.937	0.75	7203.919	0.68	7201.987
i300_8	300	300	11449.67(*)	0.00	1346.933	0.00	1755.865	0.00	296.946	0.00	1625.651	0.00	520.492	0.00	1476.683
i300_9	300	300	10932.88(*)	0.00	2909.397	0.00	2163.102	0.00	218.993	0.00	1361.067	0.00	1005.685	0.00	1925.262
i300_10	300	300	11232.77	0.82	3777.251	0.82	3976.203	1.22	902.586	0.83	7201.420	0.82	7204.003	0.82	7216.416
i300_11	300	300	10046.94(*)	0.00	33.528	0.00	33.350	0.05	13.994	0.00	115.679	0.00	49.397	0.00	53.929
i300_12	300	300	9359.64(*)	0.00	144.820	0.00	93.880	0.02	13.151	0.00	70.307	0.00	62.990	0.00	77.049
i300_13	300	300	10103.49(*)	0.00	3281.720	0.00	2094.150	0.16	243.298	0.00	1492.235	0.00	607.804	0.00	2220.523
i300_14	300	300	9738.05(*)	0.00	318.277	0.00	308.792	0.03	65.146	0.00	131.574	0.00	80.730	0.00	126.500
i300_15	300	300	9902.26(*)	0.00	282.974	0.00	245.839	0.62	39.093	0.00	205.096	0.00	89.933	0.00	261.753
i300_16	300	300	9168.08(*)	0.00	15.808	0.00	16.297	0.00	5.007	0.00	20.494	0.00	20.533	0.00	23.260
i300_17	300	300	9181.07(*)	0.00	27.436	0.00	27.006	0.64	8.612	0.00	16.252	0.00	16.247	0.00	17.409
i300_18	300	300	9581.95(*)	0.00	55.957	0.00	49.810	0.91	17.192	0.00	71.738	0.00	66.455	0.00	99.840
i300_19	300	300	9062.16(*)	0.00	17.490	0.00	17.444	0.47	4.883	0.00	24.831	0.00	24.562	0.00	28.922
i300_20	300	300	9078.22(*)	0.00	20.407	0.00	18.210	0.00	5.273	0.00	26.099	0.00	27.222	0.00	29.250
i3001500_1	300	1500	154999.14(*)	0.00	503.239	0.00	497.285	0.17	186.092	0.00	194.209	0.00	199.263	0.00	202.492
i3001500_2	300	1500	159438.60(*)	0.00	302.723	0.00	263.051	0.08	158.684	0.00	141.550	0.00	117.485	0.00	156.203
i3001500_3	300	1500	157300.15(*)	0.00	661.213	0.00	472.069	0.14	266.355	0.00	145.567	0.00	128.666	0.00	235.404
i3001500_4	300	1500	157796.28(*)	0.00	1159.057	0.00	836.755	0.13	344.433	0.00	673.667	0.00	667.967	0.00	719.832
i3001500_5	300	1500	161306.00(*)	0.00	387.133	0.00	295.793	0.35	180.539	0.00	262.273	0.00	254.797	0.00	339.862
i3001500_6	300	1500	156669.28(*)	0.00	257.302	0.00	234.788	0.18	165.439	0.00	158.318	0.00	158.669	0.00	164.069
i3001500_7	300	1500	157031.55(*)	0.00	241.983	0.00	235.370	0.31	161.382	0.00	156.629	0.00	156.629	0.00	163.586
i3001500_8	300	1500	157802.83(*)	0.00	256.516	0.00	254.100	0.35	174.034	0.00	160.972	0.00	161.893	0.00	167.610
i3001500_9	300	1500	156968.46(*)	0.00	247.816	0.00	239.538	0.02	172.209	0.00	159.359	0.00	160.379	0.00	165.645
i3001500_10	300	1500	157764.39(*)	0.00	247.468	0.00	235.407	0.10	169.760	0.00	157.903	0.00	156.903	0.00	162.681
i3001500_11	300	1500	150015.13(*)	0.00	235.815	0.00	233.700	0.00	165.439	0.00	156.875	0.00	157.284	0.00	157.627
i3001500_12	300	1500	154937.67(*)	0.00	257.706	0.00	255.677	0.16	182.770	0.00	156.487	0.00	155.815	0.00	161.885
i3001500_13	300	1500	151608.42(*)	0.00	237.140	0.00	235.771	0.74	177.154	0.00	151.606	0.00	151.690	0.00	153.071
i3001500_14	300	1500	151848.05(*)	0.00	239.336	0.00	242.388	0.12	174.205	0.00	158.579	0.00	158.975	0.00	159.405
i3001500_15	300	1500	156480.89(*)	0.00	243.642	0.00	253.636	0.00	176.328	0.00	153.281	0.00	153.361	0.00	154.928
i3001500_16	300	1500	155495.62(*)	0.00	223.736	0.00	222.255	0.35	171.600	0.00	150.402	0.00	150.165	0.00	150.747
i3001500_17	300	1500	156038.04(*)	0.00	252.710	0.00	251.273	0.32	172.724	0.00	147.634	0.00	147.117	0.00	149.405
i3001500_18	300	1500	156799.93(*)	0.00	240.778	0.00	240.066	0.22	178.636	0.00	152.044	0.00	152.187	0.00	152.509
i3001500_19	300	1500	155947.13(*)	0.00	246.482	0.00	242.342	0.13	178.121	0.00	149.267	0.00	150.143	0.00	149.530
i3001500_20	300	1500	156426.14(*)	0.00	252.310	0.00	254.036	0.28	174.642	0.00	146.769	0.00	147.175	0.00	148.703
i500_1	500	500	26412.41	1.56	5650.400	1.79	5506.064	1.84	915.331	2.09	7233.951	1.79	7225.589	1.89	7216.463
i500_2	500	500	28130.74	1.85	4702.949	2.07	4215.468	2.09	915.332	1.98	7528.300	2.13	7485.727	2.22	7206.354
i500_3	500	500	27904.51	1.80	5804.406	1.64	5652.558	2.09	911.151	1.82	7383.188	1.68	7295.847	2.42	7471.796
i500_4	500	500	28159.03	2.04	6002.189	2.06	5564.510	2.06	913.880	2.15	7300.679	2.04	7414.084	2.36	7721.647
i500_5	500	500	24702.77	1.73	5803.310	1.82	5550.934	1.85	913.522	1.82	7215.880	1.53	7387.514	1.73	7203.219
i500_6	500	500	15756.82	0.64	6319.431	0.61	6317.071	0.64	908.623	0.64	7202.849	0.66	7204.223	0.75	7204.514
i500_7	500	500	16109.28	0.60	6321.427	0.60	5849.287	0.62	908.780	0.75	7240.570	1.33	7461.936	1.62	7203.094
i500_8	500	500	16041.73	0.70	6327.848	0.70	6317.143	0.80	913.319	0.62	7211.912	0.53	7298.383	0.62	7222.579
i500_9	500	500	16327.71	0.44	5050.178	0.65	6371.211	0.65	908.374	0.59	7227.798	0.58	7211.382	0.82	7203.625
i500_10	500	500	15815.13	0.73	6474.079	0.84	6409.783	0.85	910.090	0.97	7214.917	0.81	7210.424	0.76	7330.234
i500_11	500	500	13497.71(*)	0.00	2678.694	0.00	2570.429	0.28	281.425	0.00	1141.234	0.00	984.325	0.00	1263.274
i500_12	500	500	14675.02	0.42	4257.617	0.64	2821.077	0.75	913.148	0.71	7221.495	0.42	7203.768	0.67	7204.576
i500_13	500	500	13666.25	0.36	4866.336	0.36	4698.959	0.39	911.229	0.36	7216.827	0.39	7204.097	0.39	7227.321

i500_14	500	500	13629.54(*)	0.03	2956.496	0.03	3183.265	0.03	909.185	0.00	7215.451	0.00	6746.850	0.00	6235.617
i500_15	500	500	13896.76	0.36	5425.913	0.36	5413.755	0.44	913.724	0.36	7209.418	0.36	7204.368	0.36	7210.286
i500_16	500	500	12618.68(*)	0.00	1212.711	0.00	645.665	0.00	80.730	0.00	4606.009	0.00	4886.145	0.00	4722.815
i500_17	500	500	13386.17(*)	0.00	2388.633	0.03	2364.661	0.03	232.596	0.00	5369.779	0.00	3468.454	0.00	6370.708
i500_18	500	500	12852.52(*)	0.00	447.683	0.00	448.352	0.15	46.738	0.00	976.325	0.00	859.975	0.00	471.994
i500_19	500	500	13521.52(*)	0.00	996.535	0.00	994.733	0.05	37.799	0.00	6258.143	0.00	3392.119	0.00	3681.716
i500_20	500	500	12362.26(*)	0.00	118.831	0.00	121.405	0.25	26.286	0.00	177.762	0.00	154.207	0.00	150.774
i700_1	700	700	36905.93	2.29	6102.502	2.02	5547.183	2.39	926.267	3.15	7444.411	2.98	7431.978	3.52	7445.877
i700_2	700	700	34311.71	2.23	6003.572	2.34	5534.519	2.70	923.350	3.35	7461.674	2.99	7477.811	3.71	7440.090
i700_3	700	700	34294.63	1.99	5866.431	2.33	5573.378	2.38	924.676	2.34	7461.441	2.65	7203.416	2.89	7343.338
i700_4	700	700	38090.90	2.02	5739.139	2.01	5623.046	2.11	925.627	2.63	7505.590	2.73	7520.027	3.88	7622.330
i700_5	700	700	37802.10	1.74	5530.161	1.89	5567.745	2.04	924.926	2.54	7484.668	2.40	7520.341	3.82	7561.099
i700_6	700	700	19910.67	0.95	6828.499	1.03	6460.036	1.03	927.484	1.42	7435.196	1.18	7402.321	2.29	7560.382
i700_7	700	700	21297.30	0.93	6672.663	0.71	6521.385	0.83	926.377	1.59	7391.119	0.95	7911.521	2.00	7475.315
i700_8	700	700	20659.96	0.79	6462.144	1.13	6447.699	1.13	928.841	1.78	7386.222	1.22	7544.769	1.93	7459.637
i700_9	700	700	20979.88	0.59	7526.146	0.66	7404.635	0.67	925.799	1.45	7436.185	0.67	7549.102	2.39	7368.813
i700_10	700	700	22055.41	1.19	6589.162	1.13	6808.542	1.26	920.527	1.74	7246.606	1.23	7504.672	2.30	7203.484
i700_11	700	700	17120.15	0.44	6257.306	0.41	6354.786	0.55	925.814	0.42	7203.262	0.64	7203.870	0.72	7204.482
i700_12	700	700	18130.42	0.69	6338.953	0.67	6341.005	0.70	922.180	0.56	7203.343	0.56	7398.052	0.83	7551.100
i700_13	700	700	17239.96	0.32	5441.263	0.31	5438.058	0.31	921.634	0.26	7242.526	0.30	7252.229	0.33	8060.269
i700_14	700	700	17337.63	0.29	6349.248	0.29	4550.247	0.29	953.642	0.27	7212.306	0.27	7213.284	0.27	7228.288
i700_15	700	700	18145.49	0.51	5469.249	0.52	5502.446	0.52	924.099	0.78	7203.736	0.78	7203.866	0.76	7203.921
i700_16	700	700	16029.55(*)	0.00	1149.214	0.00	1144.166	0.05	221.084	0.00	4645.075	0.00	4237.805	0.00	6408.803
i700_17	700	700	16199.55(*)	0.18	2968.712	0.18	2970.135	0.18	927.796	0.00	3943.623	0.00	4145.014	0.00	6664.004
i700_18	700	700	16443.07(*)	0.02	2468.256	0.00	3318.523	0.02	840.716	0.00	5641.572	0.00	4067.726	0.00	3696.224
i700_19	700	700	16399.79(*)	0.00	1421.483	0.00	1447.388	0.03	262.892	0.00	5559.224	0.00	1853.696	0.00	5748.033
i700_20	700	700	15434.21	0.78	3709.754	0.60	4844.247	0.78	929.574	0.44	7202.782	0.38	7203.298	0.58	7203.671
i1000_1	1000	1000	49509.81	3.11	5977.808	2.48	5775.407	2.47	967.623	4.95	7319.003	4.56	7318.409	5.07	7635.371
i1000_2	1000	1000	50688.57	1.95	5743.155	2.16	5564.515	2.35	972.787	5.06	7206.469	3.34	7213.423	8.15	7221.253
i1000_3	1000	1000	47202.64	1.99	5702.181	1.99	5568.926	2.06	966.734	4.34	7274.134	3.43	7317.833	5.17	7206.339
i1000_4	1000	1000	48868.54	2.78	6300.522	2.13	5603.420	2.67	970.541	3.64	7205.779	3.40	7206.715	5.26	7228.850
i1000_5	1000	1000	50743.54	2.13	5872.481	2.21	5642.671	2.56	973.863	3.76	7257.229	3.92	7259.896	4.03	9379.430
i1000_6	1000	1000	27823.84	0.98	7461.927	1.16	7541.224	1.17	966.733	6.47	7205.361	2.52	7205.688	4.69	7268.692
i1000_7	1000	1000	27252.32	0.99	7502.515	1.02	7633.843	1.21	971.242	2.82	7206.235	2.09	7222.279	2.86	7281.734
i1000_8	1000	1000	27375.37	1.06	9025.487	1.03	8557.247	0.96	966.562	2.62	7206.472	1.93	7206.639	2.62	7248.850
i1000_9	1000	1000	26857.09	1.23	7612.117	1.25	7738.354	1.15	970.306	3.61	7206.017	2.10	7206.148	3.88	7497.030
i1000_10	1000	1000	27186.99	1.17	7347.191	1.04	7465.657	1.04	962.163	3.34	7209.333	2.57	7218.305	3.73	7242.718
i1000_11	1000	1000	22180.33	0.53	7416.488	0.63	7421.549	0.65	966.328	1.01	7206.027	1.89	7205.593	2.64	7205.887
i1000_12	1000	1000	22160.39	0.33	6412.984	0.32	6429.246	0.34	960.291	1.10	7362.431	0.92	7376.055	1.88	7206.761
i1000_13	1000	1000	22657.09	0.55	6597.131	0.49	6457.703	0.55	959.027	1.29	7209.034	1.28	7205.575	1.42	7494.035
i1000_14	1000	1000	22312.01	0.43	6461.072	0.43	6497.133	0.53	959.620	1.56	7508.508	2.46	7285.166	2.66	7206.557
i1000_15	1000	1000	22629.44	0.43	7385.738	0.41	7368.705	0.52	969.448	1.28	7396.739	0.96	7309.944	2.66	7206.666
i1000_16	1000	1000	21331.81	0.40	6420.705	0.38	6585.080	0.38	964.924	0.39	7466.203	0.27	7274.794	0.46	7486.250
i1000_17	1000	1000	21188.89	0.22	5787.048	0.22	5590.041	0.22	968.855	0.48	7239.019	0.27	7206.649	0.61	8009.023
i1000_18	1000	1000	20713.43	0.40	5665.803	0.40	5581.119	0.40	971.663	0.19	7283.909	0.19	7213.486	0.19	7237.991
i1000_19	1000	1000	20537.45	0.43	5642.936	0.40	5605.308	0.46	974.970	0.58	7213.642	0.35	7207.516	0.63	7553.377
i1000_20	1000	1000	21560.86	0.27	4327.692	0.32	5558.133	0.32	978.840	0.40	7210.504	0.43	7229.443	0.45	7210.832
Avg.				0.64	3701.957	0.65	3628.030	0.78	618.119	1.00	4697.806	0.86	4574.924	1.19	4766.450

[a] Optimal solution values are denoted with the symbol (*).

ilar. Indeed, the average *LB/Opt. Gap %* is slightly larger for almost all the data sets, and computing times are slightly smaller. The not remarkable saving in terms of average computing times is due to the fact that, even after fixing some of the y_j variables to 1, the resulting problem is still hard to solve. It is worth highlighting the performance of the Kernel Search(1) variant for the TB5 group of instances in terms of reduction of the *LB/Opt. Gap %* and especially of the *Worst Gap %* compared to the Kernel Search. Finally, even though the gaps reported by the Kernel Search(0–1) variant are worse than for the other heuristics, they are actually quite small. In fact, the average *LB/Opt. Gap %* is never larger than 1.10%, 70 instances out of 100 were solved within a 1% LB/Opt. gap (see Table 11), and the *Worst Gap %* is always smaller than 2.70%. Kernel Search(0–1) found the optimal solution for only 7 out of the 43 instances with proven optimality, but the average error for the remaining instances is relatively small (it is approximately equal to 0.23%). On the other hand, computing times are much smaller than both the Kernel Search and the Kernel Search(1) variant with an average reduction larger than 80%.

As far as a comparison with CPLEX is considered, any heuristic reported an average *LB/Opt. Gap %*, computed over all the instances in TBED1 data set, smaller than that achieved by CPLEX with any setting (see Table 11). It is clear from Table 7 that Kernel Search outperforms CPLEX in terms of statistics *LB/Opt. Gap %* and *Worst Gap %*, especially for data sets TB4 and TB5. A similar performance is achieved by the Kernel Search(1) variant, with the exception of slightly larger average gaps reported for data set TB1. The Kernel Search(0–1) variant is outperformed by CPLEX for data sets TB1 and TB2. Conversely, the latter heuristic variant significantly outperforms CPLEX, with any setting, for data sets TB4 and TB5. As far as computing times are taken into consideration, each heuristic, in particular the Kernel Search(0–1) variant, is significantly faster than CPLEX with any tested setting.

5. Conclusions

The Kernel Search presented in this paper is a heuristic with a very simple and general structure, applicable to any Binary Integer Linear Programming problem. The heuristic relies on the high performance of commercial softwares for the solution of Linear Programming and Mixed Integer Linear Programming problems and is based on the solution of subproblems restricted to subsets of variables. The heuristic has been applied to the Single Source Capacitated Facility Location Problem that is considered to be a hard problem in the class of facility location problems because of the restriction that each customer can be served by one facility only. The results show that, although the heuristic does not include any step that explicitly considers the specific structure of the problem, the Kernel Search performs extremely well. Extensive computational experiments on benchmark instances show that the Kernel Search outperforms the best heuristics for the Single Source Capacitated Facility Location Problem available in the literature. The Kernel Search found the optimal solution for 165 out of 170 instance with a proven optimum. The error in the remaining instances is negligible. The proposed heuristic is able to solve new very large-scale instances, with up to 1000 facilities and 1000 customers. We also introduced variants of the heuristic based on variable fixing. Computational results showed that the variants may improve the efficiency of the algorithm with minor deteriorations of the solution quality.

Simpler versions of the Kernel Search have been shown to perform extremely well on other hard combinatorial problems. The heuristic has a strong potential for applicability to real world problems, due to the small implementation effort required and the implicit flexibility.

Acknowledgments

The authors would like to express their appreciation for the valuable comments made by two anonymous reviewers.

Appendix A. Detailed computational results

See Tables 8–11.

References

Ahuja, R. K., Orlin, J. B., Pallottino, S., Scaparra, M. P., & Scutellà, M. G. (2004). A multi-exchange heuristic for the single-source capacitated facility location problem. *Management Science, 50*(6), 749–760.
Angelelli, E., Mansini, R., & Speranza, M. G. (2010). Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers and Operations Research, 37*(11), 2017–2026.
Angelelli, E., Mansini, R., & Speranza, M. G. (2012). Kernel search: A new heuristic framework for portfolio selection. *Computational Optimization and Applications, 51*(1), 345–361.
Avella, P., & Boccia, M. (2009). A cutting plane algorithm for the capacitated facility location problem. *Computational Optimization and Applications, 43*(1), 39–65.
Chen, C. H., & Ting, C. J. (2008). Combining lagrangian heuristic and ant colony system to solve the single source capacitated facility location problem. *Transportation Research Part E: Logistics and Transportation Review, 44*(6), 1099–1122.
Climer, S., & Zhang, W. (2006). Cut-and-solve: An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence, 170*(8), 714–738.
Contreras, I. A., & Díaz, J. A. (2008). Scatter search for the single source capacitated facility location problem. *Annals of Operations Research, 157*(1), 73–89.
Cortinhal, M. J., & Captivo, M. E. (2003). Upper and lower bounds for the single source capacitated location problem. *European Journal of Operational Research, 151*(2), 333–351.
Danna, E., Rothberg, E., & Pape, C. L. (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming Series A, 102*(1), 71–90.
Delmaire, H., Díaz, J. A., Fernández, E., & Ortega, M. (1999). Reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *Infor-Information Systems and Operational Research, 37*(3), 194–225.
Devine, M. D., & Lesso, W. G. (1972). Models for the minimum cost development of offshore oil fields. *Management Science, 18*(8), 378–387.
Díaz, J. A., & Fernández, E. (2002). A branch-and-price algorithm for the single source capacitated plant location problem. *Journal of the Operational Research Society, 53*(3), 728–740.
Eiselt, H. A., & Marianov, V. (Eds.). (2011). *Foundations of location analysis. International series in operations research & management science* (Vol. 155). New York: Springer.
Filho, V. J. M. F., & Galvão, R. D. (1998). A tabu search heuristic for the concentrator location problem. *Location Science, 6*(1–4), 189–209.
Fischetti, M., & Lodi, A. (2003). Local branching. *Mathematical Programming, Series B, 98*(1), 23–47.
Fisher, M. L., Jaikumar, R., & Van Wassenhove, L. N. (1986). A multiplier adjustment method for the generalized assignment problem. *Management Science, 32*(9), 1095–1103.
Galvão, R. D., & Marianov, V. (2011). Lagrangean relaxation-based techniques for solving facility location problems. In H. A. Eiselt & V. Marianov (Eds.), *Foundations of location analysis* (pp. 391–420). New York: Springer.
Guastaroba, G., & Speranza, M. G. (2012a). Kernel search: An application to the index tracking problem. *European Journal of Operational Research, 217*(1), 54–68.
Guastaroba, G., & Speranza, M. G. (2012b). Kernel search for the capacitated facility location problem. *Journal of Heuristics, 18*(6), 877–917.
Hindi, K. S., & Pienkosz, K. (1999). Efficient solution of large scale, single-source, capacitated plant location problems. *Journal of the Operational Research Society, 50*(3), 268–274.
Holmberg, K., Rönnqvist, M., & Yuan, D. (1999). An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research, 113*(3), 544–559.
Klose, A., & Drexl, A. (2005). Facility location models for distribution system design. *European Journal of Operational Research, 162*(1), 4–29.
Lazić, J., Hanafi, S., Mladenović, N., & Urošević, D. (2010). Variable neighborhood decomposition search for 0–1 mixed integer programs. *Computers & Operations Research, 37*(6), 1055–1067.
Mirchandani, P., & Francis, R. (Eds.). (1990). *Discrete location theory*. New York: John Wiley and Sons.
Pirkul, H. (1987). Efficient algorithms for the capacitated concentrator location problem. *Computers & Operations Research, 14*(3), 197–208.
Rönnqvist, M., Tragantalerngsak, S., & Holt, J. (1999). A repeated matching heuristic for the single-source capacitated facility location problem. *European Journal of Operational Research, 116*(1), 51–68.
Yang, Z., Chu, F., & Chen, H. (2012). A cut-and-solve based algorithm for the single-source capacitated facility location problem. *European Journal of Operational Research, 221*(3), 521–532.