Primal-Dual Variable Neighborhood Search for the Simple Plant-Location Problem

Pierre Hansen
GERAD and HEC Montréal, Montréal, Québec H3T 2A7, Canada, pierre.hansen@gerad.ca

Jack Brimberg
GERAD and Royal Military College of Canada, Kingston, Ontario K7K 5L0, Canada, jack.brimberg@rmc.ca

Dragan Urošević
Mathematical Institute, SANU, Belgrade, Serbia, draganu@mi.sanu.ac.yu

Nenad Mladenović
School of Mathematics, Brunel University, West London, Uxbridge UB8 3PH, United Kingdom, nenad.mladenovic@brunel.ac.uk

1. Introduction
The simple plant-location problem (SPLP), also known as the uncapacitated facility-location problem, is one of the fundamental and most studied models in facility-location theory. The objective is to choose, from a set of potential facility-locations on a network, which ones to open to minimize the sum of opening (or fixed) costs and service (or variable) costs to satisfy the known demands from a set of customers. Although the origins of the plant-location problem go back to the pioneering work of Weber (1909), the actual formulation of SPLP may be attributed to Stollsteimer (1963), Kuehn and Hamburger (1963), and Balinski (1965). Since that time numerous articles have been published that deal with the properties and solution of the mathematical model, e.g., see the surveys in Krarup and Pruzan (1983), Labbé et al. (1995), Labbé and Louveaux (1997), and the books by Mirchandani and Francis (1990), Francis et al. (1992), and Daskin (1995). Despite the inherent assumptions of the model that may limit its practicality, the SPLP has gained considerable importance as a basic model in several combinatorial problems dealing, e.g., with vehicle dispatching, set covering, set partitioning, assortment, and, more recently, information retrieval and data mining (Pentico 1976, 1988; Jones et al. 1995; Tripathy et al. 1999). The SPLP is also used in multicriteria extensions (Brimberg and ReVelle 1998, 2000; Myung et al. 1997), and is shown to be an embedded problem in a number of types of location problems (ReVelle and Laporte 1996).

Most articles dealing with SPLP are concerned with solving the mathematical program. An exact branch-and-bound procedure was first proposed by Efroymson and Ray (1966), and later by Khumawala (1972). A dual-based model, the well-known DUALOC algorithm, was developed by Erlenkotter (1978) and a similar version by Bilde and Krarup (1977). The main idea here is to solve a reduced nonlinear form of the dual-based model heuristically by a simple ascent and adjustment procedure that often produces the optimal dual solution, which in turn often corresponds directly to the optimal primal integer solution. Otherwise, a branch-and-bound procedure is implemented to complete the solution. Refinements to the dual approach allowed Körkel (1989) to solve exactly much larger instances than previously attempted with sizes on the order of $1,500 \times 1,500$. A primal-dual algorithm by Galvão and Raggi (1989) alternates between the primal and dual problems solving each in turn heuristically. Again a branch-and-bound procedure is used as required to close the process. More recently an exact algorithm has been proposed by Goldengorin.
et al. (2003a) that is based on a pseudo-Boolean representation of the problem due originally to Hammer (1968). A data-correcting algorithm by Goldengorin et al. (2003b) may be used for an exact or approximate solution of the problem; however, the instances tested are relatively small.

A direct solution approach to SPLP involves solving the primal LP relaxation followed by branch and bound on the fractional values as in Morris (1978). The relaxation is known often to yield an integer solution (or zero duality gap), a property referred to as the integer friendliness of the model structure (ReVelle 1993). Tests in Brimberg and ReVelle (2000) on problem instances of up to 100 facility nodes × 300 customer nodes revealed that, in the majority of cases where fractional solutions were obtained, the optimal solution was found after branching off a single variable. However, as the SPLP is NP-hard (Cornuéjols et al. 1990), the number of branchings tends to grow exponentially with problem size. Nonetheless, the latest general mixed integer LP solvers such as CPLEX 8.1 are now able to handle problem sizes up to 1,000 × 1,000 by using built-in efficient branch-and-cut routines.

For NP-hard problems such as SPLP, it becomes necessary to use approximate methods to solve large instances. Earlier heuristics typically involve a local search on the primal problem using add, drop, and interchange moves (Kuehn and Hamburger 1963, Feldman et al. 1966, Teitz and Bart 1968, Manne 1964). See Cornuéjols et al. (1977) for an analysis of the worst case behavior of such methods. Later methods include Lagrangian-based heuristics (Beasley 1993), a projection method (Conn and Cornuèjols 1990), and more recently the application of metaheuristics such as Tabu search (Goncharov and Kochetov 1999, 2000; Ghosh 2003), a worst case behavior of such methods. Later methods include Lagrangian-based heuristics (Beasley 1993), a projection method (Conn and Cornuèjols 1990), and more recently the application of metaheuristics such as Tabu search (Goncharov and Kochetov 1999, 2000; Ghosh 2003). See Cornuèjols et al. (2003b) that is based on a pseudo-Boolean representation of the problem due originally to Hammer (1968). A data-correcting algorithm by Goldengorin et al. (2003b) may be used for an exact or approximate solution of the problem; however, the instances tested are relatively small.

A direct solution approach to SPLP involves solving the primal LP relaxation followed by branch and bound on the fractional values as in Morris (1978). The relaxation is known often to yield an integer solution (or zero duality gap), a property referred to as the integer friendliness of the model structure (ReVelle 1993). Tests in Brimberg and ReVelle (2000) on problem instances of up to 100 facility nodes × 300 customer nodes revealed that, in the majority of cases where fractional solutions were obtained, the optimal solution was found after branching off a single variable. However, as the SPLP is NP-hard (Cornuéjols et al. 1990), the number of branchings tends to grow exponentially with problem size. Nonetheless, the latest general mixed integer LP solvers such as CPLEX 8.1 are now able to handle problem sizes up to 1,000 × 1,000 by using built-in efficient branch-and-cut routines.

For NP-hard problems such as SPLP, it becomes necessary to use approximate methods to solve large instances. Earlier heuristics typically involve a local search on the primal problem using add, drop, and interchange moves (Kuehn and Hamburger 1963, Feldman et al. 1966, Teitz and Bart 1968, Manne 1964). See Cornuéjols et al. (1977) for an analysis of the worst case behavior of such methods. Later methods include Lagrangian-based heuristics (Beasley 1993), a projection method (Conn and Cornuèjols 1990), and more recently the application of metaheuristics such as Tabu search (Goncharov and Kochetov 1999, 2000; Ghosh 2003), a genetic algorithm (Kratika et al. 2001) and volume algorithms with random rounding (Barahona and Chudak 2000).

In the next section we review the formulation of the SPLP, which is given by a mixed binary integer LP, and discuss several known linear and nonlinear formulations of the relaxed dual that will be required later. In Section 3 the rules of our variable neighborhood search (VNS) applied to the SPLP are outlined, followed by an explanation of the steps of a reduced VNS (RVNS) and the decomposition approach (VND), two variants of VNS designed for solving large problem instances. Guaranteed bounds of the heuristic solution may be obtained by solving the relaxed LP, either in the primal or dual space. However, for very large problem instances, it is not possible to store all variables in memory, or solve the LP in reasonable time. We may obtain approximate bounds (Erlenkotter 1978), but these will be less effective when a branch-and-bound phase is used later to find the optimal solution. Hence, in Section 4, we proceed as follows: (i) an initial dual solution (not necessarily feasible) is obtained from the primal VNDS solution using the complementary slackness conditions; (ii) a variable neighborhood descent (VND) is applied in the dual space to improve the solution. Section 5 then solves the problem exactly as follows: (iii) an exact solution of the dual is obtained using the last solution as the starting point and a new sliding simplex method developed by us that is able to reduce the size of the dual considerably; (iv) finally, reverting back to the primal problem, a branch-and-bound algorithm is used, strengthened by a tight lower bound from the dual and upper bound from the earlier heuristic (VNDs) solution.

Computational results are reported in Section 6 on a series of randomly-generated problems, including instances that are much larger than previously tested in the literature. The ability to solve very large SPLPs is becoming more important in view of the size of facility-location problems being tackled in practice today, and the fact that the SPLP is finding other applications in such areas as cluster analysis, computer and telecommunications network design, information retrieval, and data mining. Section 7 summarizes the main results and offers possible directions of future research.

2. Mathematical Model
Let \( I = \{1, 2, \ldots, m\} \) denote a set of potential facilities, and \( J = \{1, 2, \ldots, n\} \) a set of users or customers on a network. The SPLP is

\[
\min_{x, y} \quad z_P = \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{c \in C} c_{ij} x_{ij} \tag{1}
\]

s.t.

\[
\sum_{c \in C} x_{ij} = 1, \quad \forall j \in J; \tag{2}
\]

\[
x_{ij} - y_i \leq 0, \quad \forall i \in I, \forall j \in J; \tag{3}
\]

\[
y_i \in \{0, 1\}, \quad \forall i \in I; \tag{4}
\]

\[
x_{ij} \geq 0, \quad \forall i \in I, \forall j \in J, \tag{5}
\]

where

- \( f_i \) denotes the fixed cost for opening facility \( i \);
- \( c_{ij} \) is the distribution cost for satisfying the demand of user \( j \) from facility \( i \);
- \( y_i \) is a boolean variable equal to 1 if facility \( i \) is opened, and 0 otherwise;
- \( x_{ij} \) is the fraction of demand of user \( j \) satisfied from facility \( i \).

This problem has \( mn + m \) variables and \( mn + n \) constraints. The objective function expresses that the sum of fixed costs to open facilities and distribution costs
for these facilities to serve the users must be minimized. The \( f_i \) and \( c_{ij} \) are assumed to be positive. Constraint (2) imposes that the demand at each customer is satisfied. Constraint (3) opens facility \( i \) by setting \( y_i = 1 \) if there is a flow from this facility to any customer \( j (x_{ij} > 0) \). Once the \( y_i \) are fixed, the values of the \( x_{ij} \) are easily determined; for each \( j, x_{ij} = 1 \) for that open facility with minimum distribution cost \( c_{ij} \) (ties may be broken arbitrarily); all other \( x_{ij} = 0 \).

If the binary constraints in (4) are relaxed to 0 \( \leq y_i \leq 1, \forall i \in I \), one gets the strong linear programming relaxation. Another relaxation, called the weak linear programming relaxation, may be obtained by also replacing the constraints in (3) by \( \sum_{i,j} x_{ij} \leq ny_i, \forall i \in I \), so that the number of original constraints is reduced from \( mn + n \) to \( m + n \). But this latter relaxation is less tight than the strong one, and does not appear to be of interest. The strong LP relaxation is known to be integer-friendly (ReVelle 1993, Brimberg and ReVelle 2000); i.e., most variables \( y_i \) take an integer value in the optimal vector, and thus, the duality gap is also small and often zero.

A particular class of test problems has been studied in depth (Barahona and Chudak 2000) and often used in empirical studies. Potential sites for facilities coincide with the given locations of the users, and are points taken from a uniform distribution on the unit square. Distribution costs \( c_{ij} \) are Euclidean distances between \( i \) and \( j \); fixed costs are equal for all facilities, and set at \( \sqrt{n}/10, \sqrt{n}/100, \) or \( \sqrt{n}/1000 \). It is proven that any branch-and-bound algorithm using the strong relaxation (without further cutting planes) will require a number of branches that increases exponentially with \( m \) (or \( n \)). Nevertheless, near-optimal solutions may be readily obtained for fairly large instances. For example, Barahona and Chudak (2000) recently solved, with an instance-dependent error of at most 1%, problems with \( m = n \) up to 3,000.

2.1. Dual Formulations

The dual of the strong LP relaxation is

\[
\max_{v, w, x} \left( \sum_{j \in J} v_j - \sum_{i \in I} t_i \right) \tag{6}
\]

s.t. \( \sum_{j \in J} w_{ij} - t_i \leq f_i, \forall i \in I \) \tag{7}
\( v_j - w_{ij} \leq c_{ij}, \forall i \in I, \forall j \in J \) \tag{8}
\( t_i, w_{ij} \geq 0, \forall i \in I, \forall j \in J. \) \tag{9}

Note that the variables \( v_j \) are not restricted in sign. However, because the equality sign in (2) may be replaced by \( \geq \) without affecting the optimal solution, the \( v_j \) will be nonnegative. This problem has \( mn + m + n \) variables and \( mn + m \) constraints. Thus, as in the primal, it is large in both dimensions. Fortunately, the dual may be simplified in various ways. First observe that each variable \( t_i \) appears only in the objective function, with a negative sign, and in a single constraint in (7). Further examination shows that the \( t_i \) may be reduced one at a time without reducing the objective function’s value. Using (7) and (9), we have

\[
t_i = \max \left\{ \sum_{j \in J} w_{ij} - f_i, 0 \right\} = \left( \sum_{j \in J} w_{ij} - f_i \right)^+, \tag{10}
\]

where \( a^+ = \max\{a, 0\} \). It follows that in the optimal solution the \( t_i \) should all be zero, yielding the simpler LP formulation of the dual that usually appears in the literature:

\[
\max_{v, w} z_D = \sum_{j \in J} v_j \tag{11}
\]

s.t. \( \sum_{j \in J} w_{ij} \leq f_i, \forall i \in I \) \tag{12}
\( v_j - w_{ij} \leq c_{ij}, \forall i \in I, \forall j \in J \) \tag{13}
\( w_{ij} \geq 0, \forall i \in I, \forall j \in J. \) \tag{14}

Erlenkotter (1978) observed that for any fixed vector of \( v_i \)'s, the \( w_{ij} \) may be reduced without affecting feasibility, i.e., the \( w_{ij} \) may be made as small as possible. Thus, constraints (13) and (14) imply that we should set

\[
w_{ij} = \max[v_j - c_{ij}, 0] = (v_j - c_{ij})^+, \forall i, j. \tag{15}
\]

Now substitute (15) into (12) to get a nonlinear-programming formulation in \( n \) variables with \( m \) constraints, known as the restricted dual:

\[
\max_{v} \sum_{j \in J} v_j \tag{16}
\]

s.t. \( \sum_{i \in I} (v_j - c_{ij})^+ \leq f_i, \forall i \in I. \) \tag{17}

Another way to obtain a restricted dual is to substitute both (10) and (15) into the standard LP formulation (6)–(9) to get the unconstrained nonlinear program in \( n \) variables (Spielberg 1969, Conn and Cornuéjols 1990):

\[
\max_v F(v) = \sum_{j \in J} v_j - \sum_{i \in I} \left( \max\left\{ \sum_{j \in J} (v_j - c_{ij})^+ - f_i, 0 \right\} \right). \tag{18}
\]

Note that this formulation is equivalent to one suggested by Karkazis (1985),

\[
\max_v F(v) = \sum_{j \in J} v_j + \sum_{i \in I} \left( \min\left\{ f_i - \sum_{j \in J} (v_j - c_{ij})^+, 0 \right\} \right)
\]

since \( -\max\{x\} = \min\{-x\} \) always holds.

It is well known that \( F(v) \) is a piecewise linear concave objective function in \( n \) variables, and several
nonlinear-programming methods that use specifics of
the problem have been suggested to maximize it. For
example, in Karkazis (1985) and Klose (1995) sub-
gradient methods have been derived, and in Conn
and Cornuéjols (1990) a gradient-projection method is
proposed.

3. Solving the Primal Problem by
Variable Neighborhood Search
VNS is a recent metheuristic (see Mladenović and
Hansen 1997, and the surveys in Hansen and Mladenović 2001, 2003), whose basic idea is to use systematically
different neighborhoods, both in descent phases
and to jump out of local optimum traps.

To apply VNS to our problem, the same data structure
is used as suggested in Voss (1996) and in Hansen
and Mladenović (1997) for the $p$-median problem:
- Indices of facilities in the current and in the best
  known solution are stored in arrays $s$ and $s^*$.
- Indices of closest and second-closest open facilities
  for each user and associated costs are stored in
  arrays $\xi$ and $\xi^*$, respectively.

3.1. A VNS Heuristic for SLP
To apply VNS, a neighborhood structure must be
defined on the solution space. Let $S$ denote any subset
of open facilities ($S \subseteq I$); the solution space $x$ may
then be defined as all such possible subsets. The total
number of solutions in $x$ is $2^n - 1$. To define the
neighborhoods, we use a distance function. Let $S_1$, $S_2$
be any two solutions in $x$; the distance between
them is defined by $\rho(S_1, S_2) = |(S_1 \setminus S_2) \cup (S_2 \setminus S_1)|$. If $S_2$ is obtained from $S_1$ by closing one facility $i_1 \in S_1$
and opening an $i_2 \in I \setminus S_1$ (an interchange: $S_2 = (S_1 \setminus \{i_1\}) \cup \{i_2\}$), $\rho(S_1, S_2) = 2$; if $S_2 = S_1 \setminus \{i_1\}$ (a drop) or $S_2 = S_1 \cup \{i_2\}$ (an add), $\rho(S_1, S_2) = 1$. The $k$th neighborhood
of a current solution $S$ is defined as the set of all possible
solutions $S'$ derived from $S$ by any combination
of exactly $k$ total-interchange, drop, or add moves.

The basic steps of VNS consist of a repetitive sequence of (i) shaking to a $k$th neighborhood of the
incumbent solution; (ii) conducting a local search
from the perturbed solution using the first neighborhood;
and (iii) moving to a better local optimum if one is
found.

(i) Shaking. To get a random point $S'$ in the $k$th neighborhood of the incumbent solution $S$ (which corresponds to distance at most $2k$), the following steps are repeated $k$ times:
- Choose a facility $i_1$ at random from $S$ equiprobably.
- Choose a facility $i_2$ at random from $I \setminus S$ equiprobably.
- Generate a uniform random number $r$ from the interval $(0, 1)$.
- If $r < 0.2$, delete $i_1$ from the solution ($p \leftarrow
  p - 1$); if $r \geq 0.8$, add $i_2$ to the solution ($p \leftarrow p + 1$);
- If $r \in (0.2, 0.8)$, interchange positions $i_1$ and $i_2$ in $s$, i.e.,
close facility $i_1$ and open facility $i_2$.
- Update the arrays for first and second closest facilities w.r.t. the new open facilities.

(The values $0.2$ and $0.8$ used in choosing the move
have been obtained empirically by comparing results
for threshold values distant of $0.1$ at the time. Values
$0.2$ and $0.8$ imply interchanges are more frequent
in shaking than opening or closing facilities yet the
number of these may vary.)

(ii) Local search. A local search is conducted from
the perturbed solution $S'$ using the first neighborhood,
$\mathcal{N}_1(S')$. In the best-improvement version of the
local search that we are using, all $p(m - p) + m$
solutions from $\mathcal{N}_1(S')$ are visited, and a move made
to the best among them only if its objective-function
value is smaller than that of $S'$. A fast-interchange
version, as proposed in Whitaker (1983) and Hansen
and Mladenović (1997) for solving the $p$-median, is
applied. The local search is repeated after each down-
ward move until a local minimum is reached.

(iii) Move or not. The simplest acceptance criterion
for basic VNS is used. A move is made only if the
local search in (ii) obtains a better solution than the
incumbent $S$. Each time a move is made, $k$ is reset to
$k_{\min}$ (a parameter typically $1$); otherwise $k$ is changed
typically augmented by one until a parameter $k_{\max}$
is reached, after which it is reset to $k_{\min}$, and the cycle
is repeated. The search is terminated after a stopping
 criterion, such as a limit on execution time or on
the number of iterations without an improvement, is
reached.

3.2. Variable Neighborhood Decomposition
Search
Reduced variable neighborhood search (RVNS) and
variable neighborhood decomposition search (VNDS)
are two variants of VNS devoted to solving large
problem instances. In RVNS, we simply skip the local-
search phase of the basic VNS.

Our procedure first obtains an initial solution with
RVNS. Two parameters are specified for RVNS: the
maximum neighborhood distance, $k'_{\max}$, for the shaking
operation, and a stopping criterion based on the
maximum number of iterations $i_{\max}$ allowed between
two improvements. A suitable compromise between
speed and quality was found experimentally to be
$k'_{\max} = 2$ and $i_{\max} = 30$ or 20 seconds elapsed time.
This shows that shaking must remain moderate to get
an improved solution without a descent phase. Once
RVNS is executed, we proceed with our decomposition
heuristic as outlined below:

1. Initialization. Choose values for the two param-
eters $\ell_{\max}$ (maximum number of open facilities to
be selected from the incumbent solution) and \( t_{\text{max}} \) (maximum computing time for the heuristic). Set the incumbent solution \( S \) to be the set of open facilities obtained by RVNS, and let \( p = |S| \). Set the size of the decomposed problem, \( \ell = 2 \).

2. **Constructing the decomposed problem.** (i) Determine the \((p-1) \times p\) matrix \( R = [r_{ij}] \) of ordered network distances where column \( j \) is assigned the \( j \)th facility listed in \( S \), row \( i \) is reserved for the \( i \)th closest facility in \( S \) to each facility \( j \), \( i = 2, \ldots, p \), and \( r_{ij} \) is the corresponding network distance. (Note: Also save facility indices in \( R \).)

(ii) Determine \( r_{ij} = \min_{1 \leq \ell \leq p} \{r_{ij}\} \).

(iii) The subproblem and its initial solution \((D)\) are defined as follows:

- The open facilities are given by the facility assigned column \( j^* \) and the 2nd, 3rd, \ldots, \( \ell \)th closest facilities to it (the first \( \ell - 1 \) entries in column \( j^* \)).
- The subset of users consists of the \( n \) ones assigned in the incumbent solution \( S \) to the subset of \( \ell \) open facilities just identified.
- Additional potential facility sites are added by a subroutine.

3. **Solving the decomposed problem.** If the total number of facility sites (open or closed) in the subproblem, \( m' \leq 1,000 \), solve it by VNS; if \( 1,000 < m' \leq 1,500 \), solve it by RVNS; else set \( \ell = 2 \) and return to step 2 (the decomposed problem is too big).

4. **Move or not.** If the new solution \( D' \) is better than \( D \), proceed to step 5; else if \( \ell = \ell_{\text{max}} \), set \( \ell = 2 \); else set \( \ell = \ell + 1 \). If \( t < t_{\text{max}} \), return to step 2; else stop.

5. **Adjusting for boundary effect.** Add the new decomposed solution \( D' \) to the fixed portion of \( S \) \((S \leftarrow (S,D) \cup D')\). Conduct a local search from the new solution to obtain a local optimum \( S' \). Set \( S = S' \). If \( t < t_{\text{max}} \), set \( \ell = 2 \) and return to step 2; else stop.

Note that the new set of facilities obtained in the subproblem may influence users not considered in the subproblem, i.e., some users may change assignment with respect to this new solution. Such a boundary effect is accounted for by updating arrays \( \zeta \) and \( \tilde{\zeta} \) in the whole space each time a new improved solution in the subproblem is found.

4. **A VNS Heuristic for the Dual**

4.1. **Initial Dual Solution**

Guaranteed performance of the primal heuristic may be determined if a lower bound on the objective function value is known. To that end the standard approach is to relax the integrality condition on the \( y_i \) variables. The well-known integer-friendliness property ensures that the strong LP relaxation for the SPLP gives a small duality gap between the optimal integer and relaxed solutions. We may then evaluate the existing gap as a percentage: \( 100 \times (z_h - z_r) / z_r \), where \( z_h \) denotes the solution obtained by the heuristic and \( z_r \), the solution of the strong LP relaxation, to determine the maximum error obtained by the heuristic solution.

Let us consider the dual of the strong LP relaxation, \((11)–(14)\). For large problems (say \( n = m = 1,500 \)) finding the exact solution of the primal or dual by some general LP solver such as CPLEX would be impossible or, at best, very time consuming. Thus, we first develop some procedures that will take into account the primal solution \((y, x)\) found by the heuristic, and avoid solving completely the dual problem at this stage.

The complementary-slackness conditions for the SPLP are

\[
\begin{align*}
\sum_{j \in J} x_{ij} - 1 &= 0, \quad \forall j \in J \\
\sum_{j \in J} w_{ij} (y_i - x_{ij}) &= 0, \quad \forall i \in I, \forall j \in J \\
f_i - \sum_{j \in J} w_{ij} y_i &= 0, \quad \forall i \in I \\
(c_{ij} - v_j + w_{ij}) x_{ij} &= 0, \quad \forall i \in I, \forall j \in J,
\end{align*}
\]

where \((y, x)\) and \((v, w)\) denote the associated primal and dual solutions (feasible or not), respectively.

The strong duality theorem \((z^*_p = z^*_d)\) is obtained by summing first each of \((19)–(22)\) and then summing their left- and right-hand sides. In this proof all four complementary slackness conditions \((19)–(22)\) are needed. However, \((20), (21), \text{and} (22)\) are not necessarily true if we add integrality constraints on the primal variables \(y_i\), and that is the source of the duality gap \(z_p - z_d\). Since the primal solution is feasible, \((19)\) is automatically satisfied.

Mladenović et al. (2006) prove following proposition. Here \(I^+\) denotes the set of open facilities, and \(I^- = I \setminus I^+\), the set of closed ones, in the heuristic solution \((y, x)\).

**Proposition 1.** If \(|I^+| \geq 2\) and a feasible primal solution is such that

\[
\sum_{j \in I^+} (c_{ij} - v_j)^+ \leq f_i, \quad \forall i \in I^-,
\]

then \((y, x)\) is an optimal solution of the strong LP relaxation of SPLP.

Therefore, having a set of open facilities \(I^+\) and associated vector of second-closest distances \(\tilde{\zeta}\) obtained by VNDS, we first check if \((23)\) is satisfied, and if that is the case, the incumbent solution solves SPLP optimally and no further work is required.

Otherwise, our next objective is to derive an approximate dual solution from the primal (heuristic) solution. What makes our procedure new is that the
dual solution does not have to be feasible. In effect, we take advantage of two expected conditions: (a) the primal VNDS solution is very close to optimum; (b) the duality gap is small. Thus, by finding a dual solution with the same objective-function value as the primal, we expect to be close in the dual solution space to the optimal (feasible) dual solution. To accomplish this, the complementary-slackness conditions must be satisfied.

**Proposition 2.** For a given primal solution \( y \), let \( v \) be a corresponding dual solution such that

\[
\sum_{j \in J} v_j = f_i + \sum_{j \in J^i} \xi_j, \quad \forall i \in I^+, \tag{24}
\]

where \( J \) denotes the subset of users assigned to open facility \( i \). Then \( z_D(v) = z_p(y) \).

**Proof.**

\[
z_D(v) = \sum_{j \in J} v_j = \sum_{i \in J^+} \sum_{j \in J_i} v_j = \left( \sum_{i \in J^+} f_i + \sum_{j \in J^i} \xi_j \right) = z_p(y). \quad \Box
\]

We use the previous proposition to find an initial dual solution \( v \) that belongs to the intersection of the \( p = |I^+| \) hyperplanes in (24). If, in addition, we impose the condition

\[
\xi_j \leq v_j \leq \bar{c}_j, \quad \forall j \in J, \tag{25}
\]

it follows that all the complementary slackness conditions will be satisfied (see Mladenović et al. 2006). Thus, a dual solution that satisfies (24) and (25) also satisfies the four complementary slackness conditions (19)–(22). Also note that conditions (21) and (22) are used exclusively to get the dual solution for a given dual (Erlenkotter 1978), and that the transformation of a primal to a corresponding dual is usually done by setting \( v_j = \xi_j \), which is known to produce feasible but bad initial dual solutions (Galvão and Raggi 1989). Proposition 2 is capable of providing a better initial dual, but, since such a solution is not unique, alternative procedures must be investigated. We consider the following approaches.

(i) **Proportional formula.**

\[
v_{i^j} = \xi_j + \frac{f_i(\bar{c}_j - \xi_j)}{\sum_{i \in J^i}(\bar{c}_i - \xi_i)}, \quad \forall j \in J, \quad i = i^j, \tag{26}
\]

where \( i^j \) denotes the closest open facility to \( j \) (i.e., the facility assigned to \( j \)). Summing the left and right sides of (26) over \( j \in J \), it follows that (24) holds, \( \forall i \in I^+ \). We may also show that (25) is satisfied. Since the primal solution is a local minimum, we have (Mladenović et al. 2006) \( \sum_{i \in J^i}(\bar{c}_i - \xi_i) \geq f_i, \forall i \in I^+ \) and thus, \( \xi_j \leq v_j \leq \bar{c}_j, \forall j \in J \).

(ii) **Projection formula.** Given any dual solution \( v'_j \in \mathbb{R}^n \), we may find its closest point that belongs to the manifold defined in (24) by

\[
v_j = v'_j - \frac{1}{|J|} \left( \sum_{j \in J} v'_j - f_i - \sum_{i \in J} \xi_j, \quad j \in J, \quad i = i^j. \right.
\]

For example, we could select \( v'_j = (\bar{c}_j + \xi_j)/2 \), \( \forall j \in J \); i.e., take a point \( (v') \) in the middle of the hypercube \( H = \prod_{j \in J} [\xi_j, \bar{c}_j] \). Another possibility would be \( v' = \max\{\xi_j, \min\{\bar{c}_j, \xi_j\}\}, \forall j \in J \). In the last expression, \( \bar{c}_j \) is defined as \( \bar{c}_j = \min_{i \in I} |c_i|, \forall j \in J \) (see Mladenović et al. 2006 for details).

4.2. **Improving the Dual Solution**

As seen above, the initial dual solution is easily obtained by closed formula; however, it will most likely be infeasible. To reduce this infeasibility, we consider the unconstrained dual function in (20):

\[
F(v) = \sum_{j \in J} v_j - \sum_{i \in I^+} \left( \sum_{j \in J_i} (v_j - c_i)^+ - f_i \right)^+,
\]

where the second term in the right side is the sum of infeasibilities. To maximize this function, we devise a powerful local search that uses variable neighborhood descent (VND) rules and four neighborhood structures designed for this purpose.

The first two neighborhoods represent windows around the current \( v_j \) in the ranked matrix \( \{c_i\} \). Letting \( i_j \) denote the lower index of the window, we obtain \( c_{i_j, j} \geq v_j \geq c_{i_{j+1}, j}, \forall j \in J \). To simplify the notation, denote the last inequalities that define the window around the current dual value by \( v_j \leq i_j \leq v_j \). The first neighborhood \( N_1(v) \) is constructed by replacing \( v_j \) with \( a_j \), i.e.,

\[
N_1(v) = \{(a_1, v_2, \ldots, v_n), (v_1, a_2, \ldots, v_n), \ldots, (v_1, v_2, \ldots, a_n)\}.
\]

In the same way, neighborhood \( N_2(v) \) is obtained by replacing \( v_j \) with its upper window (one at the time):

\[
N_2(v) = \{(b_1, v_2, \ldots, v_n), (v_1, b_2, \ldots, v_n), \ldots, (v_1, v_2, \ldots, b_n)\}.
\]

The cardinality of each of these two neighborhoods equals \( n \).

In the third neighborhood \( N_3(v) \), the value of some variable \( v_j \) is increased by \( \Delta v_j = \min\{b_j - v_j, \min_{i \in I^+, c_j \leq f_i} (f_i - \sum_{j \in J} (v_j - c_j)^+)^+\} \), where \( \Delta f_i = (f_i - \sum_{j \in J} (v_j - c_j)^+)^+ \). A move in \( N_3 \) will improve \( F(v) \) without increasing the infeasibility of the solution.

In \( N_4(v) \) the value of some variable \( v_j \) is decreased by

\[
\Delta v_j = \min\left( \min_{i \in I^+} \left( \sum_{j \in J} (v_j - c_i)^+ - f_i \right)^+, v_j - \xi_j \right). \tag{27}
\]
The last formula needs to be explained in more detail. When \( v_j \) is reduced by some amount, then, in order to get a larger \( F(v) \), we need to reduce at least two members of the sum \( \sum_{i \in I} (v_j - c_{ij})^+ \). Those two members should then satisfy the conditions \( v_j > c_{ij} \) and \( \sum_{j \in I} (v_j - c_{ij})^+ > f_i \). Thus, it may be possible to increase \( F(v) \) by decreasing \( v_j \) according to (27).

The VND procedure first makes best improvement moves in the \( N_i \) neighborhood of the current solution by examining all \( n \) points in that neighborhood. Once stalled, the procedure moves to the next neighborhood in the sequence \( (N_1, N_2, N_3) \), always reverting to \( N_1 \) when an improvement is found. The iterations end when no improvement is found consecutively in each of the four neighborhoods. Note that the output solution may still be infeasible.

5. Exact Solution Methods

5.1. Sliding Simplex for Exact Dual Solution

Earlier we saw that Erlenkotter’s observation in (17) leads to a restricted dual ((18), (19)) with only \( n \) variables \( v_j \) and \( m \) constraints. However, the constraints are nonlinear due to the max operator. Combining a vector \( v = (v_j) \) with relation (17) gives a dual solution \( (v, w) \) satisfying constraints (15) and (16), but not necessarily (14). Instead of trying to solve the restricted dual, we rewrite the original (linear) dual in a reduced form by taking advantage of the fact that (a) many of the constraints in (15) are nonbinding and may be eliminated; (b) for those that are binding, the \( w_j \) may be eliminated by direct substitution. The reduced linear dual and its solution by a new sliding simplex approach is discussed next.

5.1.1. Reduced Dual. Suppose that

\[
\xi_j \leq v_j \leq \tilde{c}_j, \quad \forall j \in J.
\] (28)

Let us then divide the set of users \( J \) into three subsets for each facility \( i \in I \):

\[
J_1 = \{ j \in J \mid c_{ij} < \xi_j \}, \quad J_2 = \{ j \in J \mid \xi_j \leq c_{ij} \leq \tilde{c}_j \}, \quad J_3 = \{ j \in J \mid \tilde{c}_j < c_{ij} \}.
\]

Using (15), it is immediately seen that \( w_{ij} = 0 \) for all users \( j \in J_3 \). Also from \( w_j = (v_j - c_{ij})^+ \), it holds that \( w_j = v_j - c_{ij} \), for all \( j \in J_1, i \in I \), since \( v_j \geq \xi_j \). Therefore, model (11)–(14) is reduced as follows:

\[
\begin{align*}
\max_{v, w} & \quad z_D = \sum_{j \in I} v_j \\
\text{s.t.} & \quad \sum_{j \in J_1} v_j + \sum_{j \in J_2} w_j \leq f_i + \sum_{j \in J_1} c_{ij}, \quad \forall i \in I \\
& \quad v_j - w_j \leq c_{ij}, \quad \forall i \in I, j \in J_2 \\
& \quad w_j = 0, \quad \forall i \in I, j \in J_2.
\end{align*}
\] (29)

In our sliding simplex method, the \( w_j \) variables corresponding to \( c_{ij} \notin [\tilde{c}_j, \xi_j] \) are removed with their constraints as in the above formulation, but now the bounds \( \xi_j \) and \( \tilde{c}_j \) are allowed to vary during the solution process to move towards the optimal solution while keeping a reasonable dimension on the problem size. To this end, it is necessary to rank the \( c_{ij} \) by nondecreasing values for each \( j \). Using a second-level index for ranking, we have \( c_{ij}^{(1)} \leq c_{ij}^{(2)} \leq \cdots \leq c_{ij}^{(n)} \), \( \forall j \in J \). Consider a value of \( v_j \in [c_{ij}^{(k)}, c_{ij}^{(k-1)}] \), and let \( k \) denote the largest index such that \( c_{ij}^{(k)} \leq v_j \). Then, we define the \( \ell \)-interval of \( v_j \) to be

\[
[c_{ij}^{(k)}, c_{ij}^{(k-1)}],
\] (33)

which contains the following values of the \( c_{ij} \)’s: \( c_{ij}^{(1)} \cdots c_{ij}^{(k-1)} \). Note that there may be an adjustment necessary for border effect; i.e., some end terms are obviously omitted if \( k - \ell < 1 \) or \( k + \ell > m \).

Setting \( \xi_j = c_{ij}^{(1)} \) and \( \tilde{c}_j = c_{ij}^{(k)} \), \( \forall j \in J \), one gets the reduced \( \ell \)-dual associated with vector \( v \) as given in (29)–(32) with the subsets \( J_{1\ell}, J_{2\ell}, J_{3\ell} \), \( i = 1, \ldots, m, \) updated appropriately.

The steps of the sliding simplex are as follows:

1. **Initialization.** For each \( j \in J \) rank the \( c_{ij} \) in order of nondecreasing values (ties being broken arbitrarily). Record the values and corresponding indices as \( c_{ij}^{(p)} \) and \( i_p \), for \( p = 1, \ldots, |I| \), and all \( j \in J \). Choose a value for parameter \( \ell \).

2. **Initial solution.** Obtain a vector \( v \) which corresponds to a feasible or infeasible solution \((v, w)\) of the dual. Set up the first reduced \( \ell \)-dual from \( v \).

3. **Solution of the \( \ell \)-dual.** Solve the current \( \ell \)-dual using the simplex algorithm (e.g., with CPLEX) and the latest dual solution as starting solution to obtain a vector \( v^* \).

4. **Optimality test.** Check for each \( j \in J \), that the following condition holds: \( v_j^* = c_{ij}^{(k)} \) or \( c_{ij}^{(k)} < v_j^* < c_{ij}^{(k-1)} \) or \( v_j^* = c_{ij}^{(1)} \). If for some \( j \) it is not the case, go to step 5; otherwise go to step 6.

5. **Updating of the reduced \( \ell \)-dual.** Update the index \( k \) and window in (33) for each \( j \) as required; reformulate the \( \ell \)-dual accordingly and return to step 3.

6. **Output.** An optimal solution of the dual is given by \((v^*, w^*)\) where \( w^*_j = \max(v^*_j - c_{ij}, 0), \forall i \in I, \forall j \in J \); its value is \( \sum_{j \in I} w^*_j \).

The sliding simplex method bears some resemblance to the BOXSTEP method of Marsten et al. (1975). Indeed, BOXSTEP proceeds by solving a sequence of problems with additional constraints defining a box around the current solution; if the solution of the current such problem is on the boundary, the box is translated. The sliding simplex method has some differences too: (i) it adds interval constraints on a small part of the variables only; (ii) the role
of these constraints is to eliminate a large part of the original constraints and variables; and (iii) the problem obtained is a linear program solved by simplex instead of a nonlinear one solved by subgradient optimization.

Theorem 1. The sliding simplex algorithm solves the dual of SPLP.

Proof. From sensitivity analysis, one may add to the dual (11)–(14) without changing the optimal solution the set of constraints

\[ c_{i,j} \leq v_j \leq c_{i,j}, \quad \forall j \in J \]  

(34)
as at least one \( y_i \) must be equal to 1 in any feasible solution. Then the current reduced \( \ell \)-dual is equivalent to this problem with the additional constraints

\[ c_{k-i,j} \leq v_j \leq c_{i+j}, \quad \forall j \in J, \]  

(35)
where we assume that redundant constraints obtained when \( c_{k-j} = c_{i}, \) or \( c_{h-j} = c_{i} \) are omitted.

The optimal solution \((v^*, w^*)\) of (11)–(14), (34), (35) is such that none of the constraints (35) are tight, as otherwise the condition of step 4 would not hold. So it remains optimal if those constraints are removed, i.e., for (11)–(14), (34) and hence for (11)–(14).

Furthermore, since there are a finite number of combinations of windows for \( v_i \) and each combination may be encountered at most once, the algorithm must terminate after a finite number of iterations. □

5.2. Exact Primal Solution

At this stage we have a primal solution obtained by the VNDS heuristic that provides an upper bound, and an exact dual solution by sliding simplex that provides a lower bound for the optimal solution of the SPLP. If the two bounds are equal, the VNDS solution must be optimal. Otherwise, a classical branch-and-bound procedure is initiated. The tightness of the upper and lower bounds will be useful in keeping the number of branchings to a minimum. The main features of the branch-and-bound algorithm are as follows:

(a) For branching, the fractional primal variables (duals of the dual) are first identified, and those that correspond to open facilities in the heuristic solution are closed, one at a time, by a depth-first strategy.

(b) At each node of the branch-and-bound tree, a relaxed dual is solved by the sliding simplex method, described in the previous section using the solution of the parent node as the starting point.

(c) As in Erlenkotter (1978), to keep facility \( i \) closed, the fixed cost \( f_i \) is temporarily set to \(+\infty\); to keep facility \( i \) open, the fixed cost \( f_i \) is set to 0.

(d) An elementary backtracking scheme with last-in, first-out is applied.

(e) Pruning of nodes in the branch-and-bound tree is either by bounding (relaxed solution is worse than upper bound) or by obtaining a primal integer solution (no fractional dual values of the dual problem).

Because the sliding simplex method may call the LP solver many times at each node as the windows on the \( v_i \) change, it is advisable after each call to check if the node can be fathomed by bounding, before proceeding further to the exact lower bound.

6. Computational Experience

In this section we first explain what type of test instances are used. They are available in the Online Supplement to this paper on the journal’s website. Then we verify the capacity of the latest CPLEX 8.1 as a general mixed integer solver applied to the SPLP. Our main computer results on the specialized heuristics and algorithms described above are then given.

6.1. Barahona-Chudak Instances

Our procedure is tested on similarly-constructed instances from Barahona and Chudak (2000). That is, both facility and user points are assumed to be the same random uniformly-distributed set of vertices in the unit square. The fixed costs are the same for all facilities, and the transportation costs correspond to the Euclidean distances separating pairs of points in the plane. Several interesting properties of such test problems are described in Ahn et al. (1988), e.g.: (i) for \( n \leq 500 \), the problems are easy to solve; (ii) when \( n \) is large, any enumerative method based on LP relaxation requires the exploration of an exponentially-increasing number of solutions; and (iii) the value of the LP relaxation is about 0.998 of the optimal value.

Three different types of instances based on different magnitudes of fixed cost are considered. These problems provide a wide range of structural diversity: (i) Type I, \( f_i = \sqrt{n}/10, \forall i \in I \); (ii) Type II, \( f_i = \sqrt{n}/100, \forall i \in I \); and (iii) Type III, \( f_i = \sqrt{n}/1,000, \forall i \in I \). To avoid numerical problems, all data entries are made integer by rounding them to four significant digits. As noted in Barahona and Chudak (2000), the DUALOC heuristic seems to benefit most from such rounding. However, their results show that their volume algorithm together with random rounding (V&RRWC for short) outperforms significantly the dual ascent and dual adjustment heuristics of Erlenkotter (1978) on Type I and Type II instances. On Type III instances dual adjustment (DA) was the best. Experiments were performed on relatively large instances for the time, with \( m = n \leq 3,000 \). For example, comparing the \% gap (or guaranteed instance-dependent bounds) of DA and V&RRWC on all three types of instances for the maximum problem size considered \( m = n = 3,000 \), the following average results were reported: Type I: 16.79\% for DA vs. 0.71\% for V&RRWC; Type II: 4.27\% vs. 0.93\%; Type III: 0.62\% vs. 0.85\%.
6.2. CPLEX Integer Solver

One solution approach, of course, is to use a general off-the-shelf mixed integer program solver such as CPLEX. Table 1 shows the results obtained by the latest CPLEX 8.1 on various sizes of the three types of problems considered (in the usual form (1)–(5)). Columns 1 and 2 give the fixed cost and problem size, respectively, of each instance; the next two columns provide the optimal solution value and the execution time obtained by running CPLEX on a PC Pentium 4; the last two columns provide the corresponding results of our VNDS heuristic on the same machine.

The experiments show that execution time and memory requirements of CPLEX increase rapidly for moderately-sized problems. Type I problems also appear the hardest to solve. It is clear that the solution of large problems with CPLEX is still not a viable option. Meanwhile, comparing the results in the table from VNDS, we see that high-quality solutions are obtained in a fraction of the time by this heuristic. The optimal solution was found in all Type III instances, in 9 out of 11 Type II, and 5 out of 7 Type I. The Type III instances appeared very easy, requiring VNDS on average around 1 sec. to find the optimal solution.

6.3. Main Computational Results

Here we examine the computational results obtained on problem instances generated by the procedure of Barahona and Chudak (2000) described earlier. We cover the same range of instances, and go far beyond, testing problem instances as large as 15,000 × 15,000.

All programs are coded in C++ and run on two machines depending on problem size: for $n \leq 7,000$, we use a 1,800 MHz PC Pentium 4, and for $n > 7,000$, a SUN Enterprise 10,000 (with 400 Mhz clock and 64 gigabyte of RAM) that is slower but has sufficient memory to handle the larger problems.

Tables 2–4 summarize our computational results for the three types of problems considered and the problem sizes ($n = m$) indicated in the first column; for a given size the same problem is used but the fixed cost ($f_i$) at each node is adjusted according to the type I, II, or III. The value $p$ gives the number of open facility sites in the optimal (if available) or best-known solution. The next four columns give objective-function values obtained, respectively, by our branch-and-bound (optimal value), sliding simplex for exact solution of the dual, reduced variable neighborhood search (RVNS) for the first stage of the heuristic procedure, and variable neighborhood decomposition search (VNDS) for the final stage. The computation times are reported in sequence in the next four columns, followed by time totals, best representing the total time spent until the best heuristic solution was found and all, the actual time spent. The last two columns give the gap calculated as a percentage as follows:

$$\text{Gap}(B&B) = \left(\frac{z^*_B - z^*_D}{z^*_D}\right) \times 100,$$

$$\text{Gap}(Sliding) = \left(\frac{z^*_B - z^*_D}{z^*_D}\right) \times 100,$$

where $z^*_B$, $z^*_p$, and $z^*_D$ are the objective-function values obtained, respectively, by VNDS, B&B, and sliding simplex. Optimal solutions of the primal problem are not shown where computation times of the B&B exceeded an imposed limit resulting in premature termination of the algorithm.

Parameter settings selected for the variable neighborhood search procedures are noted as follows (also see Section 3):

- **VNDS**: $l_{max} = \min(p, 25)$, where $p$ is the number of open facilities in the current solution; stopping condition: 20 sequences of $\ell_{max}$ iterations without improvement (Type III), 10 (Type I and II).

- **VNS subroutine**: $k_{min} = 1$, step size = 1, $k_{max} = 20$ (Type III), 10 (Type I and II); stopping condition: 20 sequences of $k_{max}$ iterations without improvement (Type III), 10 (Type I and II).
Table 2  Main Results: Type I Instances

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>Objective values</th>
<th>Time (sec.)</th>
<th>Time total</th>
<th>% gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B&amp;B  Sliding</td>
<td>RVNS</td>
<td>VNDs</td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td>15</td>
<td>1,431,038</td>
<td>35,3</td>
<td>0.1</td>
<td>15.9</td>
</tr>
<tr>
<td>1,100</td>
<td>16</td>
<td>1,555,369</td>
<td>35,3</td>
<td>0.1</td>
<td>21.8</td>
</tr>
<tr>
<td>1,300</td>
<td>18</td>
<td>1,789,843</td>
<td>49,9</td>
<td>38.2</td>
<td>0.1</td>
</tr>
<tr>
<td>1,400</td>
<td>18</td>
<td>1,904,195</td>
<td>42.9</td>
<td>35.3</td>
<td>0.1</td>
</tr>
<tr>
<td>1,500</td>
<td>18</td>
<td>2,023,870</td>
<td>205.8</td>
<td>0.1</td>
<td>94.0</td>
</tr>
<tr>
<td>2,000</td>
<td>20</td>
<td>2,581,964</td>
<td>842.5</td>
<td>0.1</td>
<td>98.0</td>
</tr>
<tr>
<td>2,500</td>
<td>21</td>
<td>3,101,007</td>
<td>1,923.0</td>
<td>0.6</td>
<td>56.0</td>
</tr>
<tr>
<td>3,000</td>
<td>23</td>
<td>3,602,388</td>
<td>3,073.1</td>
<td>0.3</td>
<td>153.3</td>
</tr>
<tr>
<td>3,500</td>
<td>23</td>
<td>4,099,448</td>
<td>7,943.7</td>
<td>0.3</td>
<td>113.9</td>
</tr>
<tr>
<td>4,000</td>
<td>25</td>
<td>4,581,458</td>
<td>15,927.2</td>
<td>0.5</td>
<td>245.1</td>
</tr>
<tr>
<td>4,500</td>
<td>26</td>
<td>5,047,959</td>
<td>31,321.9</td>
<td>0.9</td>
<td>183.5</td>
</tr>
<tr>
<td>5,000</td>
<td>29</td>
<td>5,517,681</td>
<td>52,734.9</td>
<td>1.2</td>
<td>118.4</td>
</tr>
</tbody>
</table>

Table 3  Main Results: Type II Instances

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>Objective values</th>
<th>Time (sec.)</th>
<th>Time total</th>
<th>% gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B&amp;B  Sliding</td>
<td>RVNS</td>
<td>VNDs</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>62</td>
<td>1,328,235</td>
<td>1.0</td>
<td>1.0</td>
<td>2.44</td>
</tr>
<tr>
<td>1,000</td>
<td>77</td>
<td>1,611,100</td>
<td>5.9</td>
<td>4.9</td>
<td>11.6</td>
</tr>
<tr>
<td>1,500</td>
<td>85</td>
<td>1,872,278</td>
<td>124.4</td>
<td>9.8</td>
<td>171.6</td>
</tr>
<tr>
<td>2,000</td>
<td>92</td>
<td>1,122,577</td>
<td>495.8</td>
<td>24.2</td>
<td>138.3</td>
</tr>
<tr>
<td>2,500</td>
<td>104</td>
<td>1,366,092</td>
<td>105.5</td>
<td>1.2</td>
<td>465.7</td>
</tr>
<tr>
<td>3,000</td>
<td>107</td>
<td>1,595,895</td>
<td>75.5</td>
<td>65.3</td>
<td>315.6</td>
</tr>
<tr>
<td>3,500</td>
<td>111</td>
<td>1,819,686</td>
<td>357.4</td>
<td>1.9</td>
<td>1,106.8</td>
</tr>
<tr>
<td>4,000</td>
<td>113</td>
<td>2,042,313</td>
<td>500.2</td>
<td>0.9</td>
<td>2,366.6</td>
</tr>
<tr>
<td>4,500</td>
<td>119</td>
<td>2,255,880</td>
<td>502.4</td>
<td>1.5</td>
<td>1,767.6</td>
</tr>
<tr>
<td>5,000</td>
<td>124</td>
<td>2,468,883</td>
<td>1,054.5</td>
<td>2.3</td>
<td>1,360.2</td>
</tr>
</tbody>
</table>

Table 4  Main Results: Type III Instances

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>Objective values</th>
<th>Time (sec.)</th>
<th>Time total</th>
<th>% gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B&amp;B  Sliding</td>
<td>RVNS</td>
<td>VNDs</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>347</td>
<td>99,794</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
</tr>
<tr>
<td>1,000</td>
<td>391</td>
<td>223,206</td>
<td>1.1</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>1,500</td>
<td>410</td>
<td>323,750</td>
<td>8.6</td>
<td>2.6</td>
<td>3.0</td>
</tr>
<tr>
<td>2,000</td>
<td>453</td>
<td>438,574</td>
<td>68.0</td>
<td>4.8</td>
<td>30.7</td>
</tr>
<tr>
<td>2,500</td>
<td>498</td>
<td>542,203</td>
<td>86.1</td>
<td>7.3</td>
<td>34.8</td>
</tr>
<tr>
<td>3,000</td>
<td>519</td>
<td>642,321</td>
<td>78.3</td>
<td>8.8</td>
<td>152.1</td>
</tr>
<tr>
<td>3,500</td>
<td>542</td>
<td>741,097</td>
<td>555.9</td>
<td>13.3</td>
<td>87.1</td>
</tr>
<tr>
<td>4,000</td>
<td>570</td>
<td>839,922</td>
<td>205.3</td>
<td>21.5</td>
<td>162.2</td>
</tr>
<tr>
<td>4,500</td>
<td>582</td>
<td>932,428</td>
<td>5,156.1</td>
<td>23.4</td>
<td>113.7</td>
</tr>
<tr>
<td>5,000</td>
<td>600</td>
<td>1,028,249</td>
<td>1,266.3</td>
<td>32.2</td>
<td>239.6</td>
</tr>
<tr>
<td>6,000</td>
<td>637</td>
<td>1,211,889</td>
<td>4,030.7</td>
<td>47.6</td>
<td>761.4</td>
</tr>
<tr>
<td>7,000</td>
<td>668</td>
<td>1,392,127</td>
<td>823.3</td>
<td>981.5</td>
<td>0.035</td>
</tr>
<tr>
<td>8,000</td>
<td>701</td>
<td>1,569,292</td>
<td>2,784.4</td>
<td>3,090.1</td>
<td>0.030</td>
</tr>
<tr>
<td>9,000</td>
<td>724</td>
<td>1,742,075</td>
<td>3,818.9</td>
<td>4,182.9</td>
<td>0.018</td>
</tr>
<tr>
<td>10,000</td>
<td>752</td>
<td>1,915,065</td>
<td>4,261.9</td>
<td>4,740.1</td>
<td>0.0259</td>
</tr>
<tr>
<td>11,000</td>
<td>768</td>
<td>2,079,253</td>
<td>5,094.2</td>
<td>5,686.6</td>
<td>0.0175</td>
</tr>
<tr>
<td>12,000</td>
<td>789</td>
<td>2,245,167</td>
<td>7,784.1</td>
<td>8,504.6</td>
<td>0.0215</td>
</tr>
<tr>
<td>13,000</td>
<td>802</td>
<td>2,411,167</td>
<td>11,855.0</td>
<td>12,716.4</td>
<td>0.0109</td>
</tr>
<tr>
<td>14,000</td>
<td>827</td>
<td>2,570,329</td>
<td>11,016.8</td>
<td>11,935.4</td>
<td>0.0180</td>
</tr>
<tr>
<td>15,000</td>
<td>844</td>
<td>2,733,373</td>
<td>13,346.8</td>
<td>14,500.1</td>
<td>0.0221</td>
</tr>
</tbody>
</table>
From the summary results in Tables 2–4, we see:

- The VNDS heuristic provides high-quality solutions over a wide range of problem sizes and types. This includes much larger problem instances than currently considered in the literature. For Type III instances up to $15,000 \times 15,000$, the largest gap obtained is on the order of 0.03%. For Type II, the maximum gap is 0.06% for problem sizes up to $5,000 \times 5,000$, and Type I, 0.58%. These results present a significant improvement in the state-of-the-art given in Barahona and Chudak (2000), where gaps of 1% are reported on problem sizes up to $3,000 \times 3,000$. Meanwhile, the computation time for VNDS is very reasonable considering the problem sizes investigated. For example, problems up to $3,000 \times 3,000$ take only a few minutes; the largest one ($15,000 \times 15,000$) ran for 1.8 hours. Type III instances, the easiest for our VNDS, were the hardest for Barahona and Chudak’s V&RWC.

- The sliding simplex method is capable of solving the dual exactly for the large problems investigated. This is quite impressive considering that the largest problem solved has $n + m = 225,015,000$ dual variables. By obtaining a tight starting solution (which may be infeasible), and then using our sliding simplex method, a substantial reduction in problem size and number of simplex iterations is obtained. Computation times for sliding simplex are also seen to be reasonable, although Type I instances took significantly longer.

- By using the entire package proposed here, namely, a heuristic solution of the primal problem by VNDS, followed by exact solution of the dual with sliding simplex, and then closing the gap with branch and bound, exact solution of large SPLPs is achieved. Our largest problem solved ($7,000 \times 7,000$) set a new record (soon to be beaten, as shown below).

A referee suggested that we test the exact algorithm on instances with different fixed costs. Consequently, we combined a series of instances of type IV, in which fixed costs are drawn randomly from a uniform distribution on the interval $[\sqrt{n}/1,000, \sqrt{n}/10]$. Results are presented in Table 5; they show the following:

- Instances with different fixed costs are easier to solve than are those with uniform fixed costs. Indeed, all problems with sizes up to $15,000$ could be solved exactly, again setting a new record.

- Reduced VNS does not give good results if it is allocated a small computing time as in the previous experiments; VNDS, with the new stopping rule as before, takes more time but obtains excellent results, close to or equal to those of the sliding simplex algorithm; the value of the LP relaxation obtained by the sliding simplex algorithm is optimal in 12 cases out of 22, including those with 12,000, 13,000, and 14,000 users; the branch-and-bound algorithm has less work to do than with instances of type III, II, and especially I, although some computing time is required even when there is no duality gap to obtain an integer solution.

Finally, Table 6 studies the effect of the window size $\ell$ in the sliding simplex. Here an instance of size $n = 1,500$ is generated, and the three problem types investigated. As expected, a smaller window results
in more subproblems or calls to CPLEX. However, the effect on computation time appears to be the reverse for types I and II; computation time appears to be insensitive to the parameter $\ell$ for type III.

7. Conclusions
This paper develops a new methodology for solving the SPLP. In the first stage a heuristic based on variable neighborhood search (VNS) is used to obtain a near-optimal solution. We show that VNS with decomposition is a very powerful technique for large-scale problems, up to 15,000 facilities x 15,000 users. In the second phase, our approach is to find an exact solution of the relaxed dual problem. This is accomplished in three stages: (i) find an initial dual solution (generally infeasible) using the primal heuristic solution and complementary slackness conditions; (ii) improve the solution by applying VNS on the unconstrained nonlinear form of the dual; and (iii) finally, solve the dual exactly using a customized sliding simplex algorithm that applies windows on the dual variables to reduce the size of the problem substantially. In all problems tested, including instances much larger than previously reported, our procedure was able to find the exact dual solution in reasonable computing time. In the third and final phase, armed with tight upper and lower bounds obtained respectively from the heuristic primal solution in one and the exact dual solution in phase two, we apply a standard branch-and-bound algorithm to find an optimal solution of the original problem. The lower bounds are updated with the dual sliding simplex method and the upper bounds whenever new integer solutions are obtained at the nodes of the branching tree. In this way we were able to solve exactly problem instances with up to $7,000 \times 7,000$ for uniform fixed costs and $15,000 \times 15,000$ otherwise. This advances the record considerably.

Future directions include further experimenting with, and fine-tuning of, our primal-dual variable neighborhood search methodology. Adapting the exact solution method proposed here to extensions of the SPLP should also be investigated.

Acknowledgments
The first author was supported by NSERC Grant 105574-02 and the second author by NSERC Grant OGP205041. The third and fourth authors were partially supported by Serbian Ministry of Science, Project 1583. The authors thank Yuri Kotchetov for useful remarks that helped to improve this paper’s presentation.

References


