

# ***Recent results obtained with Primal-Dual VNS algorithms***

Pierre Hansen\*

[pierre.hansen@gerad.ca](mailto:pierre.hansen@gerad.ca)

Jack Brimberg

Dragan Urošević

Nenad Mladenović

# Outline:

- Introduction
- Problem statement
- Heuristic resolution of the primal
- Heuristic resolution of the dual
- Exact solution methods
- Computational experiments
- Concluding remarks

# Introduction

- Metaheuristics are frameworks to build heuristics for specific problems. As is well known, they are very successful in practice (and little developed in theory).
- Usually heuristics for combinatorial and global optimization exploit local search in the primal. Duality and complementary slackness are little or not exploited.

# Introduction

- Yet, they prove to be very useful to solve exactly difficult problems, or to solve them approximately but with a guarantee of quality of the solution obtained.
- We discuss here primal-dual heuristics for the simple plant location problem (SPLP) and the related  $p$ -median problem (PMP) based upon the variable neighborhood search metaheuristic.

# SPLP - Problem statement

- The objective is to choose from a set of  $m$  potential facility locations on a network which ones to open in order to minimize the sum of opening (or fixed) costs and service (or variable) costs to satisfy the known demands from a set of  $n$  costumers.

# SPLP - Problem statement

$$\min_{x,y} z_P = \sum_{i \in I} f_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \quad (1)$$

fixed cost for opening facility  $i$   
 equal to 1 if facility  $i$  is opened, and 0 otherwise  
 distribution cost for satisfying the demand of user  $j$  from facility  $i$

$$\sum_{i \in I} x_{ij} = 1, \quad \forall j \in J; \quad (2)$$

$$x_{ij} - y_i \leq 0, \quad \forall i \in I, \quad \forall j \in J; \quad (3)$$

$$y_i \in \{0, 1\}, \quad \forall i \in I; \quad (4)$$

$$x_{ij} \geq 0, \quad \forall i \in I, \quad \forall j \in J. \quad (5)$$

# SPLP - Problem statement

- The LP relaxation is known to be *integer-friendly* (ReVelle (1993) and Brimberg & ReVelle (2000)).
- For randomly generated instances, discussed later, it has been proved that any branch-and-bound algorithm using only the LP relaxation will require a number of branches that increases exponentially with  $m$ .

# SPLP - Problem statement

- Nevertheless, near optimal solutions may be readily obtained for fairly large instances.
- Barahona and Chudak (2005) recently solved with an instance-dependent error of at most 1%, problems with  $m = n$  up to 3000.



# SPLP - Problem statement

## Dual formulation

$$\max_{v,w,t} \left( \sum_{j \in J} v_j - \sum_{i \in I} t_i \right) \quad (8)$$

$$\sum_{j \in J} w_{ij} - t_i \leq f_i, \quad \forall i \in I \quad (9)$$

$$v_j - w_{ij} \leq c_{ij}, \quad \forall i \in I, \quad \forall j \in J \quad (10)$$

$$t_i, w_{ij} \geq 0, \quad \forall i \in I, \quad \forall j \in J. \quad (11)$$

# SPLP - Problem statement

- Each variable  $t_i$  appears only in the objective function, with negative sign, and in a single constraint. Then, we have

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

- For any fixed vector of  $v_j$ 's, the  $w_{ij}$  may be made as small as possible without affecting feasibility

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

# SPLP - Problem statement

- By substitution, an unconstrained nonlinear programming formulation of the dual can be stated as

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

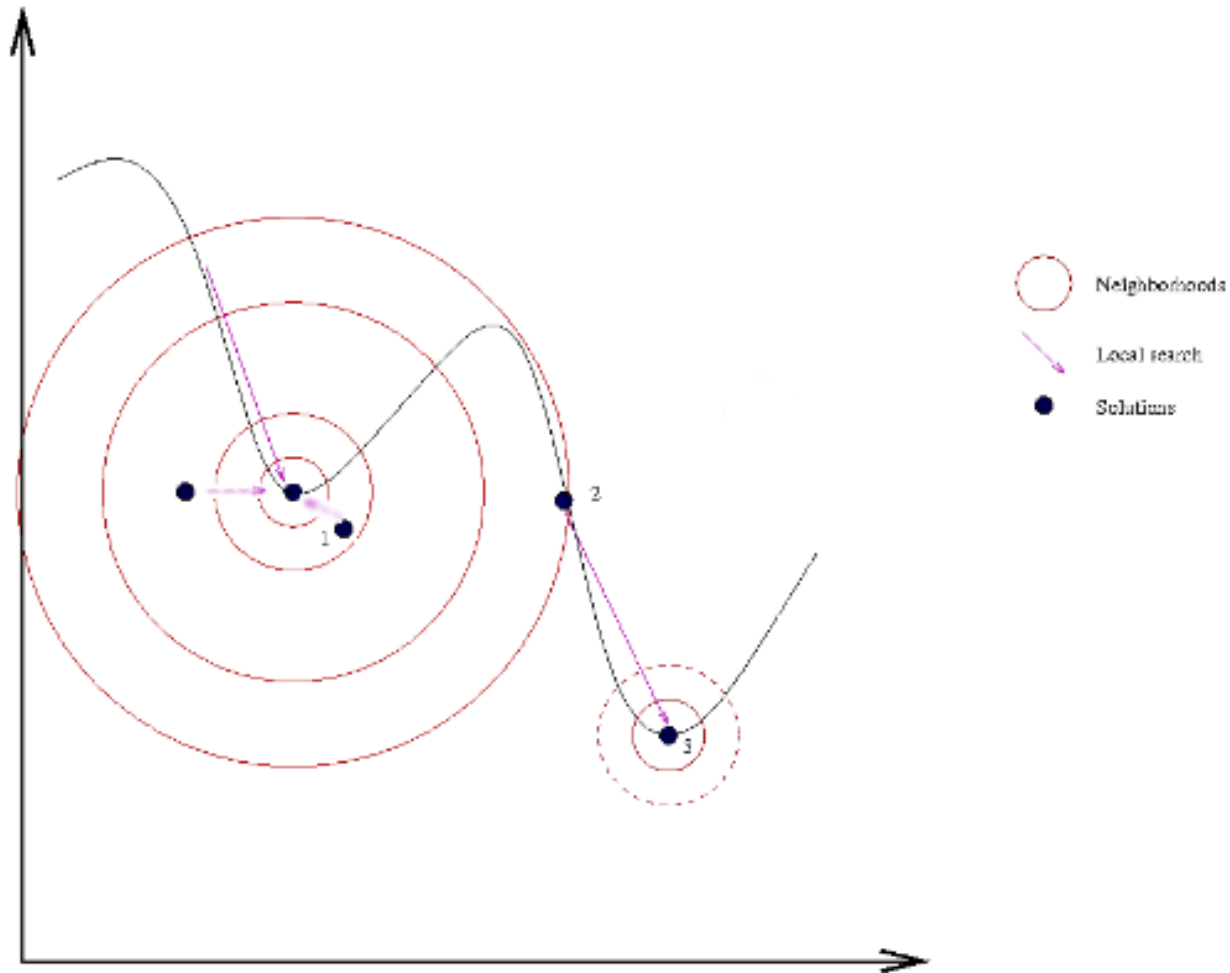
- It is a piecewise linear concave objective function in  $n$  variables.

# Heuristic resolution of the primal

- Variable neighborhood search (VNS) is a fairly recent metaheuristic (Mladenović & Hansen (1997)) whose basic idea is to use systematically different neighborhoods, both in descent phase and to “jump” out of local minimum traps.

# Example

## Variable neighborhood search



# Variable neighborhood search

---

Initialization. Select the set of neighborhood structures  $\mathcal{N}_k$ , for  $k = 1, \dots, k_{max}$ , that will be used in the search; find an initial solution  $x$ ; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

(1) Set  $k \leftarrow 1$ ;

(2) Repeat the following steps until  $k = k_{max}$ :

(a) Shaking. Generate a point  $x'$  at random from the  $k^{th}$  neighborhood of  $x$  ( $x' \in \mathcal{N}_k(x)$ ); in other words, let  $y$  be a set of  $k$  solution attributes present in  $x'$  but not in  $x$  ( $y = x' \setminus x$ ).

(b) Local search. Find the local optimum in the space of  $y$  either by inspection or by some heuristic; denote the best solution found with  $y'$  and with  $x''$  the corresponding solution in the whole space  $S$  ( $x'' = (x' \setminus y) \cup y'$ );

(c) Move or not. If the solution thus obtained is better than the incumbent, move there ( $x \leftarrow x''$ ), and continue the search with  $\mathcal{N}_1$  ( $k \leftarrow 1$ ); otherwise, set  $k \leftarrow k + 1$ ;

---

# Heuristic resolution of the primal

- In order to apply VNS, a neighborhood structure must be defined.
- Let  $S$  denote any subset of open facilities ( $S \subseteq I$ ).
- The  $k^{\text{th}}$  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.

# Heuristic resolution of the primal

- The basic steps of VNS consist of a repetitive sequence of

## (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  $S$ ;
- choose a facility  $i_2$  at random from  $I \setminus S$ ;
- generate a uniform random number  $rnd$  from the interval  $(0,1)$ ;
- if  $rnd \leq 0.2$ , delete  $i_1$  from the solution ( $p \leftarrow p - 1$ ); if  $rnd \geq 0.8$ , add  $i_2$  to the solution ( $p \leftarrow p + 1$ ); if  $rnd \in (0.2, 0.8)$ , 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.



# Heuristic resolution of the primal

## i) Local search.

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  $p(m) + m$  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, proposed in Whitaker (1983) and Hansen and Mladenovic (1997) for solving the  $p$ -Median problem, is applied. The local search is repeated after each downward move until a local minimum is reached.

## ii) 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.

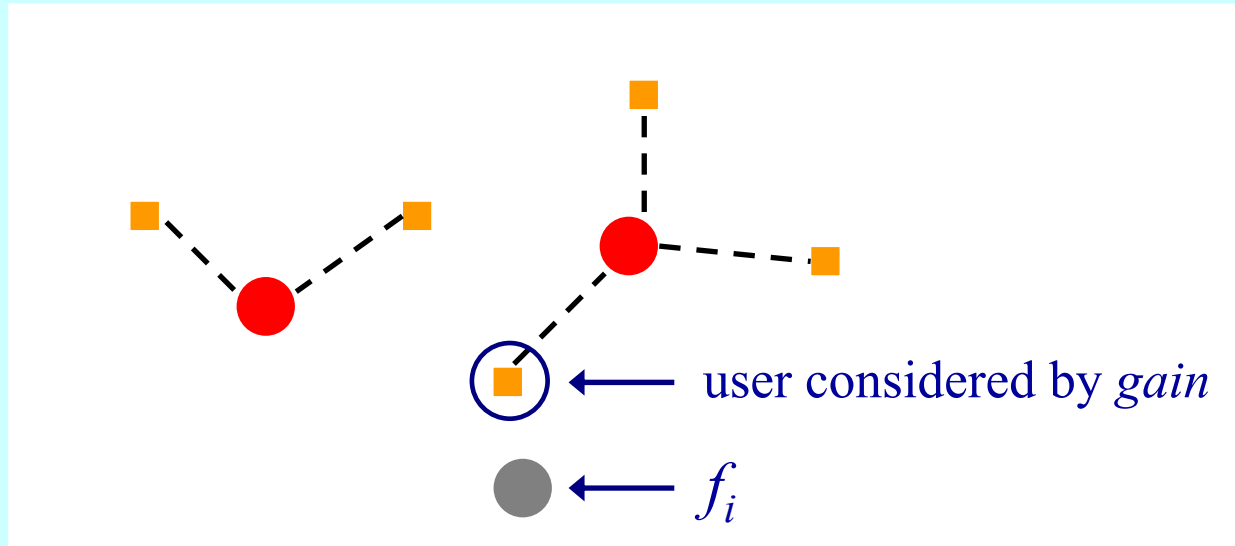
# Whitaker fast swap heuristic

- Published in 1983, even though not widely used until 1997 when Hansen and Mladenović applied it as a subroutine of a VNS procedure.
- The key aspect of this implementation is its ability to find in  $O(n)$  time the best facility to close, given a certain facility to open.

# Whitaker fast swap heuristic

- What makes this procedure fast is the observation that the variation in the solution cost can be decomposed into two components, which we call *gain* and *netloss*.
- Component *gain* accounts for all users who would benefit from the insertion of  $f_i$  into the solution. Each is closer to  $f_i$  than to the facility it is currently assigned to.

# Whitaker fast swap heuristic

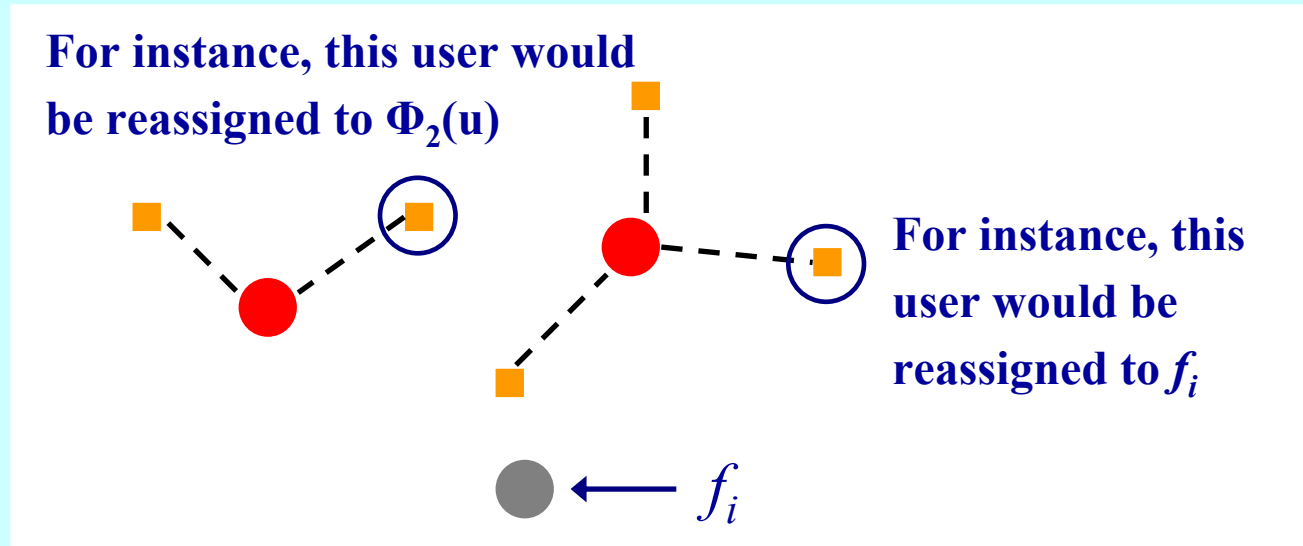


The difference between the distances is the amount by which the cost of serving that particular user will be reduced if  $f_i$  is inserted.

# Whitaker fast swap heuristic

- The second component, *netloss*, accounts for all other users, those that would not benefit from the insertion of  $f_i$  into the solution.

# Whitaker fast swap heuristic



If the facility that is closest to the user  $u$  is removed,  $u$  would have to be reassigned either to  $\Phi_2(u)$  (its current second closest facility) or to  $f_i$ .

# Whitaker fast swap heuristic

```
function findOut ( $S, f_i, \phi_1, \phi_2$ )
1    $gain \leftarrow 0$ ; /* gain resulting from the addition of  $f_i$  */
2   forall ( $f \in S$ ) do  $netloss(f) \leftarrow 0$ ; /* loss resulting from removal of  $f$  */
3   forall ( $u \in U$ ) do
4       if ( $d(u, f_i) \leq d_1(u)$ ) then /* gain if  $f_i$  is close enough to  $u$  */
5            $gain \leftarrow^+ [d_1(u) - d(u, f_i)]$ ;
6       else /* loss if facility that is closest to  $u$  is removed */
7            $netloss(\phi_1(u)) \leftarrow^+ \min\{d(u, f_i), d_2(u)\} - d_1(u)$ ;
8       endif
9   endforall
10   $f_r \leftarrow \operatorname{argmin}_{f \in S} \{netloss(f)\}$ ;
11   $profit \leftarrow gain - netloss(f_r)$ ;
12  return ( $f_r, profit$ );
end findOut
```

# Whitaker fast swap heuristic

- Given this  $O(n)$ -time function, it is trivial to implement the swap-based local search procedure in  $O(mn)$  time per iteration: simply call **findOut** once for each of the  $m-p$  candidates for insertion and pick the most profitable one.
- If the best swap is profitable, the move is performed, the values of  $\Phi_1(u)$  and  $\Phi_2(u)$  are updated, and the algorithm proceeds to the next iteration.



# Resende & Werneck fast swap heuristic

- Auxiliary data structures are used to speed up the local search procedure.
- The Whitaker algorithm tries to find the best pair  $(f_i, f_r)$  which maximizes

$$profit(f_i, f_r) = gain(f_i) - netloss(f_i, f_r)$$

# Resende & Werneck fast swap heuristic

- The algorithm differs from Whitaker's in the computation of *netloss*.
- For every facility  $f_r$  in the solution,  $loss(f_r)$  is defined as the increase in solution value that results from the removal of  $f_r$  from the solution (assuming that no facility is inserted).

$$loss(f_r) = \sum_{u: \phi_1(u)=f_r} [d_2(u) - d_1(u)]$$

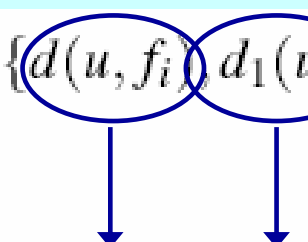
# Resende & Werneck fast swap heuristic

- As defined, *gain* and *loss* are capable of determining the net effect of a single insertion or a single deletion, but not of a swap.
- To compute *netloss* from *loss*, another function is defined, *extra*( $f_i, f_r$ ), defined so that the following is true for all pairs ( $f_i, f_r$ ):

$$netloss(f_i, f_r) = loss(f_r) - extra(f_i, f_r)$$

# Resende & Werneck fast swap heuristic

- An expression for  $extra(f_i, f_r)$  can be algebraically derived as

$$extra(f_i, f_r) = \sum_{\substack{u: [\phi_1(u)=f_r] \wedge \\ [d(u, f_i) < d_2(u)]}} [d_2(u) - \max\{d(u, f_i), d_1(u)\}]$$


It will have a net loss of  $d_2(u) - d_1(u)$  if  $d(u, f_i) < d_1(u)$  is not satisfied. If  $d(u, f_i) > d_1(u)$ , then  $d_1(u)$  is lost and  $d(u, f_i)$  is inserted. In this case, the net loss is  $d(u, f_i) - d_1(u)$ . If  $d(u, f_i) = d_1(u)$ , then no change occurs.

# Resende & Werneck fast swap heuristic

- Given the expressions of *gain*, *loss* and *extra*, the *profit* associated with each move is obtained in a very simple manner

$$profit(f_i, f_r) = gain(f_i) - loss(f_r) + extra(f_i, f_r)$$

- The interesting aspect of this decomposition is that the only term that depends on both the facility to be inserted and the one to be removed is *extra*.

# Resende & Werneck fast swap heuristic

- At first, Resende & Werneck implementation seems to be a complicated alternative to that of Whitaker with same worst-case complexity.
- Yet, additional memory is used to store *extra* as a matrix.
- However, this structure allows for significant accelerations as we will see.

# Resende & Werneck fast swap heuristic

- When a facility  $f_r$  is replaced by a new facility  $f_i$ , certain entries in *gain*, *loss*, *extra*,  $\Phi_1$ , and  $\Phi_2$  become inaccurate.
- A recomputation of these structures considering all users is the straightforward way to update them.

# Resende & Werneck fast interchange heuristic

```
function updateStructures ( $S, u, loss, gain, extra, \phi_1, \phi_2$ )  
1    $f_r \leftarrow \phi_1(u)$ ;  
2    $loss(f_r) \leftarrow^+ [d_2(u) - d_1(u)]$ ;  
3   forall ( $f_i \notin S$ ) do  
4       if ( $d(u, f_i) < d_2(u)$ ) then  
5            $gain(f_i) \leftarrow^+ \max\{0, d_1(u) - d(u, f_i)\}$ ;  
6            $extra(f_i, f_r) \leftarrow^+ [d_2(u) - \max\{d(u, f_i), d_1(u)\}]$ ;  
7       endif  
8   endfor  
end updateStructures
```



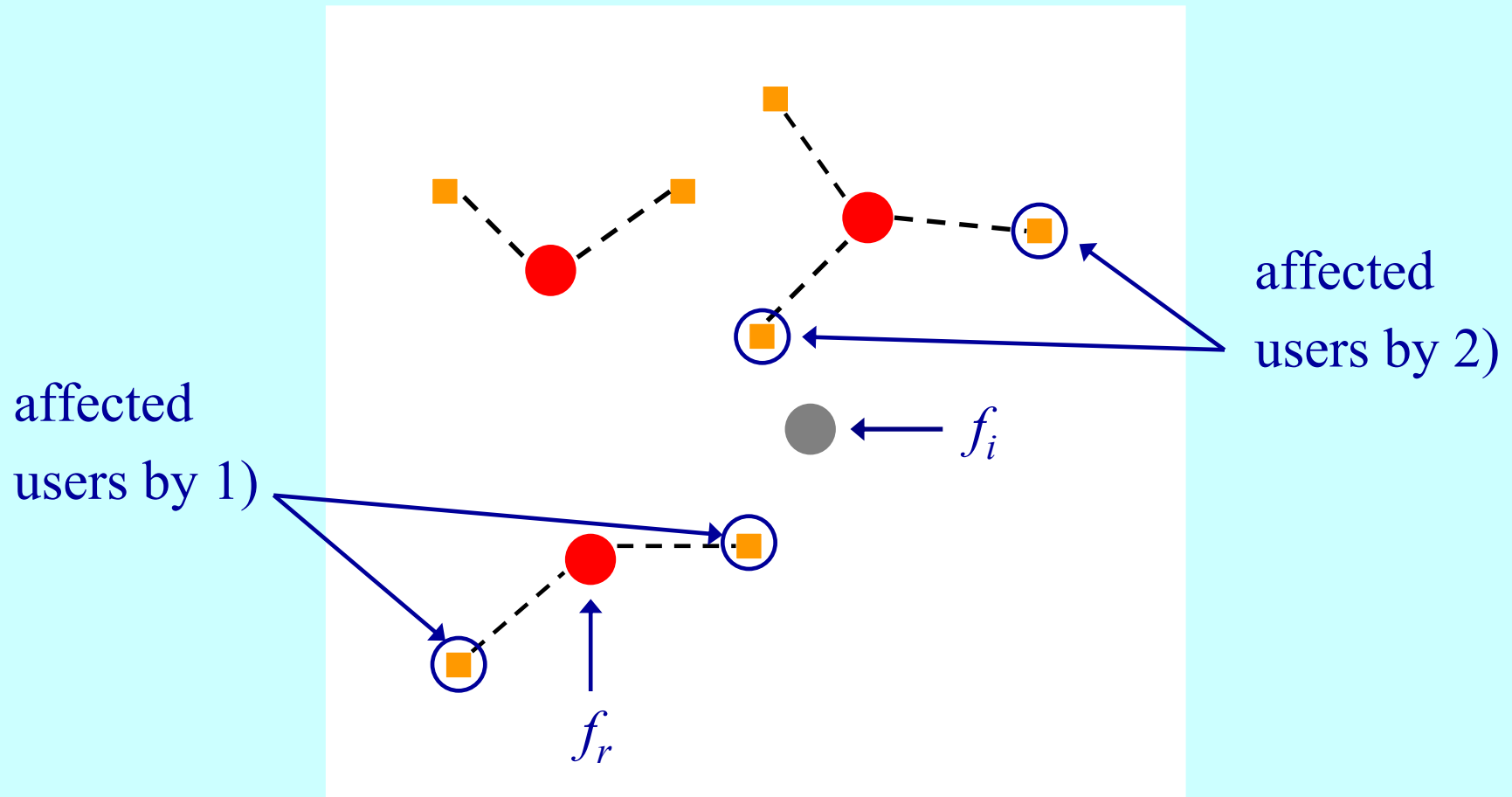
# Resende & Werneck fast swap heuristic

- A downside of this approach is that no information gathered in one iteration is used in subsequent ones.
- In fact, the actions performed by `updateStructures` depend only on  $u$ ,  $\Phi_1(u)$ , and  $\Phi_2(u)$ ; no value is read from other structures.
- If  $\Phi_1(u)$  and  $\Phi_2(u)$  do not change from one iteration to another,  $u$ 's contribution to *gain*, *loss*, and *extra* will not change either.

# Resende & Werneck fast swap heuristic

- All this means that there is no need to call **updateStructures** again for all users.
- To deal with such cases, the notion of *affected users* is introduced. Sufficient conditions for  $u$  to be affected after a swap between  $f_i$  and  $f_r$  are:
  - 1) either  $\Phi_1(u)$  or  $\Phi_2(u)$  is  $f_r$ , the facility removed; or
  - 2)  $f_i$  (the facility inserted) is closer to  $u$  than the original  $\Phi_2(u)$  is.

# Resende & Werneck fast interchange heuristic



# Resende & Werneck fast swap heuristic

- Contributions to *loss*, *gain*, and *extra* need only be updated for affected users.
- As they are often a few after a swap is performed, a significant acceleration is obtained by the entire local search procedure.

# Heuristic resolution of the primal

- 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.
- VNDS uses decomposition to enhance the efficiency of VNS when solving large instances.

# Heuristic resolution of the primal

- The proposed procedure first obtains an initial solution with RVNS.
- Two parameters are specified:
  - The maximum neighborhood distance  $k'_{\max}$ , for the shaking operation.
  - Stopping criterion based on the maximum number of iterations allowed between two improvements,  $i_{\max}$ .

# Heuristic resolution of the primal

- Once RVNS is executed, we proceed with the decomposition heuristic:

## 1. Initialization:

Choose values for the two parameters,  $\ell_{max}$  (maximum number of open facilities to be selected from the incumbent solution) and  $t_{max}$  (maximum computing time for the heuristic). Set the incumbent solution  $S$  to be the set of open facilities obtained by RVNS(2,1000), and let  $p = |S|$ . Set the size of the decomposed problem,  $\ell=2$ .

# Heuristic resolution of the primal

## 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  $j$ ,  $i = 2, \dots, p$ , and  $r_{ij}$  is the corresponding network distance. (Note: also save facility indices in  $R$ .)
- (ii) Determine  $r_{\ell j^*} = \min_{1 \leq j \leq p} \{r_{\ell j}\}$ .
- (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  $2^{nd}$ ,  $3^{rd}$ ,  $\dots$ ,  $\ell^{th}$  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. (In our computational experiments, the set of users and the set of potential facilities are always the same however for the general case, some routine to select the subset of potential facilities for the sub-problem is needed.)



# Heuristic resolution of the primal

## **Solving the decomposed problem:**

If the total number of facility sites (open or closed) in the subproblem,  $m' \leq 1000$ , solve it by VNS; if  $1000 < m' \leq 1500$ , solve it by RVNS; else set  $\ell = 2$  and return to step 2 (the decomposed problem is too big).

## **Move or not:**

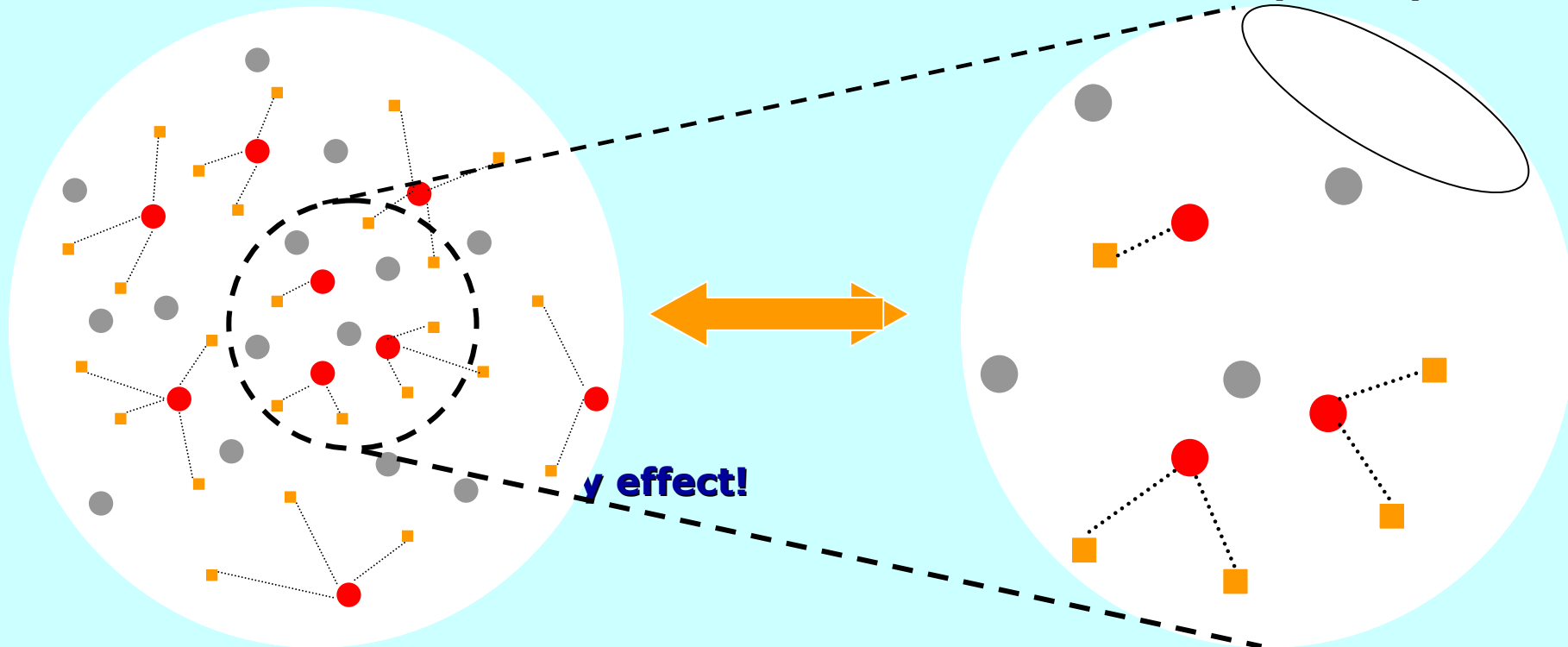
If the new solution  $D'$  is better than  $D$ , proceed to step 5; else if  $\ell = \ell_{max}$ , set  $\ell = \ell_{max} - 1$ ; else set  $\ell \leftarrow \ell + 1$ . If  $t < t_{max}$ , return to step 2; else stop.

## **Adjusting for boundary effect:**

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

# Decomposition

**These two points are not assigned to open facilities in the decomposed problem**



**Then, VNS is executed over this subproblem**

# Heuristic resolution of the dual

- **Initial dual solution**

- Guaranteed performance of primal heuristics may be determined if lower bounds are known.
- The *integer-friendliness* property ensures that the strong LP relaxation for the SPLP gives a small duality gap between optimal integer and relaxed solutions.

# Heuristic resolution of the dual

- For large instances (say  $n = m = 1500$ ) finding directly the exact solution of the primal or dual would be very time consuming.
- Thus, procedures which take into account the primal solution were developed in order to avoid solving completely the dual problem at this stage.
- Let us consider the dual formulation of the SPLP that usually appears in the literature.

# Heuristic resolution of the dual

- Let us consider the dual formulation of the SPLP that usually appears in the literature:

$$\max_{v,w} z_D = \sum_{j \in J} v_j \quad (13)$$

$$\sum_{j \in J} w_{ij} \leq f_i, \quad \forall i \in I \quad (14)$$

$$v_j - w_{ij} \leq c_{ij}, \quad \forall i \in I, \quad \forall j \in J \quad (15)$$

$$w_{ij} \geq 0, \quad \forall i \in I, \quad \forall j \in J. \quad (16)$$

# Heuristic resolution of the dual

- The *complementary slackness* conditions for the SPLP are:

$$v_j \left( \sum_{i \in I} x_{ij} - 1 \right) = 0, \forall j \in J \quad (22)$$

$$w_{ij}(y_i - x_{ij}) = 0, \forall i \in I, \forall j \in J \quad (23)$$

$$\left( f_i - \sum_{j \in J} w_{ij} \right) y_i = 0, \forall i \in I \quad (24)$$

$$(c_{ij} - v_j + w_{ij})x_{ij} = 0, \forall i \in I, \forall j \in J, \quad (25)$$

# Heuristic resolution of the dual

- The *strong duality* theorem ( $z^*_P = z^*_D$ ) is obtained by summing first each of (22), (23), (24), (25), and then summing their left and right hand sides.
- In the proof, all four complementary slackness conditions are needed.
- However, (23), (24) and (25) are not necessarily true if we add the integrality constraints on the primal variables  $y_i$ , and that is the source of the duality gap.

# Heuristic resolution of the dual

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

$$\sum_{j \in J} (\bar{c}_j - c_{ij})^+ \leq f_i, \forall i \in I^- \quad (26)$$

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

- If (26) is satisfied by the primal heuristic solution, it solves SPLP optimally and no further work is required.
- Otherwise, an approximate dual solution from the primal heuristic solution must be derived.



# Heuristic resolution of the dual

- The dual solution does not have to be feasible!
- Two expected conditions are exploited:
  - The primal VNDS solution is very close to optimum;
  - 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 space to the optimal (feasible) dual solution.
- To accomplish this, the complementary slackness condition must be satisfied.

# Heuristic resolution of the dual

**Proposition 2** For a given primal solution  $y$ , let  $v$  be a corresponding dual solution such that

$$\sum_{j \in J_i} v_j = f_i + \sum_{j \in J_i} \underline{c}_j, \quad \forall i \in I^+, \quad (27)$$

where  $J_i$  denotes the subset of users assigned to open facility  $i$ . Then  $z_D(v) = z_P(y)$ .

Proof.

$$z_D(v) = \sum_{j \in J} v_j = \sum_{i \in I^+} \sum_{j \in J_i} v_j = \sum_{i \in I^+} (f_i + \sum_{j \in J_i} \underline{c}_j) = z_P(y).$$

- If in addition, we impose the condition:

$$\underline{c}_j \leq v_j \leq \bar{c}_j, \quad \forall j \in J \quad (28)$$

it follows that all the complementary slackness conditions will be satisfied (see Mladenović *et al.* 2003).

# Heuristic resolution of the dual

- Proposition 2 is capable of providing a good initial dual solution, but, since such a solution is not unique, alternative procedures were devised.

# Heuristic resolution of the dual

(i) Proportional formula:

$$v_j = \underline{c}_j + \frac{f_i(\bar{c}_j - \underline{c}_j)}{\sum_{\ell \in J_i} (\bar{c}_\ell - \underline{c}_\ell)}, \quad \forall j \in J, \quad i = i_j^+, \quad (29)$$

where  $i_j^+$  denotes the closest open facility to  $j$  (i.e., the facility assigned to  $j$ ). Summing the left and right hand sides of (29) over  $j \in J_i$ , it follows that (27) holds,  $\forall i \in I^+$ . We may also show that (28) is satisfied. Since the primal solution is a local minimum, we have (see Mladenović *et al.* (2003)),

$$\sum_{j \in J_i} (\bar{c}_j - \underline{c}_j) \geq f_i, \quad \forall i \in I^+, \quad (30)$$

and thus,  $\underline{c}_j < v_j \leq \bar{c}_j$ ,  $\forall j \in J$ .

# Heuristic resolution of the dual

## (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 (27) by:

$$v_j = v'_j - \frac{1}{|J_i|} \left( \sum_{\ell \in J_i} v'_\ell - f_i - \sum_{\ell \in J_i} \underline{c}_\ell \right), \quad j \in J, \quad i = i_j^+. \quad (31)$$

For example, we could select  $v'_j = (\bar{c}_j + \underline{c}_j)/2, \forall j \in J$ ; i.e., take a point  $(v'_j)$  in the middle of the hypercube

$$H = \prod_{j=1}^n [\underline{c}_j, \bar{c}_j].$$

Another possibility would be:

$$v'_j = \max\{\underline{c}_j, \min\{\tilde{c}_j, \bar{c}_j\}\}, \quad \forall j \in J. \quad (32)$$

In the last expression,  $\tilde{c}_j$  is defined as  $\tilde{c}_j = \min_{i \in I^-} \{c_{ij}\}, \forall j \in J$  (see Mladenović *et al.* (2003) for details).

# Heuristic resolution of the dual

## Improving the dual solution

The initial dual solution obtained will most likely be infeasible. To reduce infeasibility, we consider the unconstrained dual function

$$F(v) = \sum_{j \in J} v_j - \sum_{i \in I} \left( \sum_{j \in J} (v_j - c_{ij})^+ - f_i \right)^+$$

where the second term in the right hand side is the sum of infeasibilities.

# Heuristic resolution of the dual

- To maximize this function, a powerful local search that uses variable neighborhood descent (VND) rules was devised as well as four neighborhoods structures designed for this purpose.

# Heuristic resolution of the dual

- The first two neighborhoods representing *windows* around the current  $v_j$  in the ranked matrix  $[c_{ij}]$ .
- Letting  $i_j$  denote the lower index of the window, we obtain:  $c_{i_j,j} \leq v_j \leq c_{i_j+1,j}, \quad \forall j \in J$ .
- To simplify the notation, let us denote the last inequalities that define the window around the current dual value by  $a_j \leq v_j \leq b_j$ .



# Heuristic resolution of the dual

- The first neighborhood  $N_1(v)$  is constructed by replacing  $v_j$  with  $a_j$ , i.e.

$$N_1(v) = \{(a_1, v_2, \dots, v_n), (v_1, a_2, \dots, v_n), \dots, (v_1, v_2, \dots, a_n)\}.$$

- In the same way, neighborhood  $N_2(v)$  is obtained by replacing  $v_j$  with its upper window  $b_j$ ,

$$N_2(v) = \{(b_1, v_2, \dots, v_n), (v_1, b_2, \dots, v_n), \dots, (v_1, v_2, \dots, b_n)\}.$$

- The cardinality of each of these neighborhoods equals  $n$ .

# Heuristic resolution of the dual

- In the third neighborhood  $N_3(v)$ , the value of some variable  $v_j$  is increased by

$$\Delta v_j = \min\{b_j - v_j, \min_{i \in I, c_{ij} \leq v_j} \Delta f_i\},$$

where

$$\Delta f_i = \left( f_i - \sum_{j \in J} (v_j - c_{ij})^+ \right)^+.$$

- A move in  $N_3$  will improve  $F(v)$  without increasing the infeasibility of the solution.

# Heuristic resolution of the dual

- In  $N_4(v)$ , the value of some variable  $v_j$  is decreased by

$$\Delta v_j = \min \left( \min_{i: v_j > c_{ij}} \left( \sum_{j \in J} (v_j - c_{ij})^+ - f_i \right)^+, v_j - \underline{c_j} \right).$$

first-closest open facility to us

- When  $v_j$  is reduced by some amount, then, in order to get a larger  $F(v)$ , at least two members of the following sum need to be reduced.

$$\sum_{i \in I} \left( \sum_{j \in J} (v_j - c_{ij})^+ - f_i \right)^+$$

# Heuristic resolution of the dual

- Those two members should then satisfy the conditions  $v_j > c_{ij}$  and  $\sum_{j=1}^n (v_j - c_{ij})^+ > f_i$ . Thus, it may be possible to increase  $F(v)$  by decreasing  $v_j$  as described before.
-

# Heuristic resolution of the dual

- The VND procedure first makes best improvement moves in the  $N_1$  neighborhood of the current solution by examining all  $n$  points in that neighborhood.
- Once stalled, the procedure moves to the next neighborhood in sequence ( $N_2, N_3, N_4$ ), 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.
- The output solution may still be infeasible.

# Exact solution methods

## Sliding simplex for exact dual solution

The original (linear) dual

$$\max_{v,w} z_D = \sum_{j \in J} v_j \quad (13)$$

$$\sum_{j \in J} w_{ij} \leq f_i, \quad \forall i \in I \quad (14)$$

$$v_j - w_{ij} \leq c_{ij}, \quad \forall i \in I, \quad \forall j \in J \quad (15)$$

$$w_{ij} \geq 0, \quad \forall i \in I, \quad \forall j \in J. \quad (16)$$

is rewritten in a reduced form, taking advantage of two facts:

# Exact solution methods

(a) many of the constraints

$$v_j - w_{ij} \leq c_{ij}, \quad \forall i \in I, \quad \forall j \in J \quad (15)$$

are nonbinding and may be eliminated;

(b) for those that are binding, the  $w_{ij}$  may be eliminated by direct substitution.

# Exact solution methods

Reduced dual. Suppose that

**first-closest open facility to user  $j$**

$$\underline{c}_j \leq v_j \leq \bar{c}_j \quad \forall j \in J. \quad (3)$$

**second-closest open facility to user  $j$**

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

$$J_{i1} = \{j \in J \mid c_{ij} < \underline{c}_j\}, \quad (3)$$

$$J_{i2} = \{j \in J \mid \underline{c}_j \leq c_{ij} \leq \bar{c}_j\}, \quad (3)$$

$$J_{i3} = \{j \in J \mid \bar{c}_j < c_{ij}\}. \quad (3)$$



# Exact solution methods

Using

$$w_{ij} = \max\{v_j - c_{ij}, 0\} = (v_j - c_{ij})^+, \forall i, j. \quad (17)$$

It is immediately seen that  $w_{ij} = 0$  for all users  $j$  that belong to the set  $J_{i3}$ . Also from (17), it holds that  $w_{ij} = v_j - c_{ij}$  for all  $j \in J_{i1}$ ,  $i \in I^-$  (set of closed facilities), since  $v_j \geq \underline{c}_j$ .

# Exact solution methods

Therefore, the dual model is reduced as follows:

$$\max_{v,w} z_D = \sum_{j \in J} v_j \quad (38)$$

$$\sum_{j \in J_{i1}} v_j + \sum_{j \in J_{i2}} w_{ij} \leq f_i + \sum_{j \in J_{i1}} c_{ij}, \quad \forall i \in I \quad (39)$$

$$v_j - w_{ij} \leq c_{ij}, \quad \forall i \in I, j \in J_{i2} \quad (40)$$

$$w_{ij} \geq 0, \quad \forall i \in I, j \in J_{i2}. \quad (41)$$

# Exact solution methods

In the *sliding simplex method* proposed, the  $w_{ij}$  variables, whose corresponding  $c_{ij} \notin [\underline{c}_j, \overline{c}_j]$ , are removed with their constraints as in the above formulation.

Only at this moment the bounds  $\underline{c}_j$  and  $\overline{c}_j$  are allowed to vary during the solution process, in order to move towards the optimal solution while keeping a reasonable dimension on problem size.

# Exact solution methods

To this end, it is necessary to rank the  $c_{ij}$  by nonincreasing values for each  $j$ . Using a second-level index for ranking, we have

$$c_{i_1j} \leq c_{i_2j} \leq \cdots \leq c_{i_mj}, \forall j \in J. \quad (42)$$

Consider a value of  $v_j \in [c_{i_1j}, c_{i_mj}]$ , and let  $k$  denote the largest index such that  $c_{i_kj} \leq v_j$ . Then, the  *$\ell$ -interval* of  $v_j$  is defined to be

$$[c_{i_{k-\ell}j}, c_{i_{k+\ell}j}] \quad (43)$$

# Exact solution methods

which contains the following values of the  $c_{ij}$ :

$$c_{i_{k-\ell}j}, c_{i_{k-\ell+1}j}, \dots, c_{i_{kj}}, c_{i_{k+1}j}, \dots, c_{i_{k+\ell}j}. \quad (44)$$

Setting  $\underline{c}_j = c_{i_{k-\ell}j}$  and  $\bar{c}_j = c_{i_{k+\ell}j} \quad \forall j \in J$ , one gets the **reduced  $\ell$ -dual** associated with vector  $v$ , as given in (38)-(41) and (34) with the subsets  $J_{i1}, J_{i2}, J_{i3}, i = 1, \dots, m$ , updated appropriately.

# Exact solution methods

The steps of the **sliding simplex** are now summarized as follows:

## 1. Initialization:

For each  $j \in J$  rank the  $c_{ij}$  in order of non-decreasing values (ties being broken arbitrarily). Record the values and corresponding indices as  $c_{i_p j}$  and  $i_p$  for  $p = 1, \dots, |I|$  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  $v^*$ .

# Exact solution methods

## . Optimality test:

Check for each  $j \in J$ , that the following condition holds:

$$v_j^* = c_{i_1j} \text{ or } c_{i_{k-\ell}j} < v_j^* < c_{i_{k+\ell}j} \text{ or } v_j^* = c_{i_mj}.$$

If for some  $j$  it is not the case, go to step 5; otherwise go to step 6.

## . Updating of the reduced $\ell$ -dual:

Update the index  $k$  and window in (43) for each  $j$  as required; reformulate the  $\ell$ -dual accordingly and return to step 3.

## . Output:

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

# Exact solution methods

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

*Proof.* From sensitivity analysis, one may add to the dual (13) - (16) without changing the optimal solution the set of constraints

$$c_{i_1j} \leq v_j \leq c_{i_mj}, \forall j \in J \quad (45)$$

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

$$c_{i_{k-\ell}j} \leq v_j \leq c_{i_{k+\ell}j}, \forall j \in J \quad (46)$$

where we assume that redundant constraints obtained when  $c_{i_{k-\ell}j} = c_{i_1j}$  or  $c_{i_{k+\ell}j} = c_{i_mj}$  are omitted.

The optimal solution  $(v^*, w^*)$  of (13) - (16), (45), (46) is such that none of the constraints (46) are tight, as otherwise the condition of Step 4 would not hold. So it remains optimal if those constraints are removed, i.e., for (13) - (16), (45) and hence for (13) - (16).

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



# Exact solution methods

## Exact primal solution

At this stage, we have:

- (a) a primal solution obtained by the VNDS heuristic.
- (b) an exact dual solution by sliding simplex.

If the two bounds are equal, the VNDS solution must be optimal.

Otherwise, a classical branch-and-bound is initiated.

The tightness of the upper and lower bounds will be useful in keeping the number of branchings to a minimum.

# Exact solution methods

The main features of the branch and bound algorithm are summarized below:

(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 at  $+\infty$ ; to keep facility  $i$  open, the fixed cost  $f_i$  is set at 0.

# Exact solution methods

(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 duals of the dual).

Since the sliding simplex method may call the LP solver many times at each node as the windows on the  $v_j$  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.

# Computational experiments

The solution procedure is tested on similarly-constructed instances as given in Barahona and Chudak (2005).

Both facilities 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 Euclidian distances separating pairs of points in the plane.

# Computational experiments

Such test problems have interesting known properties, 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;
- (iii) the value of the LP relaxation is about 0.998 of the optimal value.

# Computational experiments

Three types of instances based on different magnitudes of fixed cost are considered:

- (i) *Type I*,  $f_i = \sqrt{n}/10, \forall i \in I$
- (ii) *Type II*,  $f_i = \sqrt{n}/100, \forall i \in I$
- (iii) *Type III*,  $f_i = \sqrt{n}/1000, \forall i \in I$

In order to avoid numerical problems, all data entries are made to be integer by rounding them to four significant digits.

# Computational experiments

Table 1: Testing CPLEX.

$f_i$	$n$	$z_{mip^*}$	$t_{mipopt}$	$z_{VNDS}$	$t_{VNDS}$
$\sqrt{n}/10$	100	209805.00	1.26	209805.00	0.09
	200	361531.00	7.19	361531.00	0.75
	300	511252.00	26.97	511428.00	1.55
	400	660444.00	67.03	660444.00	6.72
	500	799791.00	141.55	802841.00	24.40
	600	926829.00	241.20	926829.00	31.59
	700	1058487.00	720.65	1058487.00	46.20
$\sqrt{n}/100$	100	70769.00	0.84	70769.00	0.16
	200	138674.00	5.05	138674.00	0.89
	300	203000.00	24.13	203000.00	2.70
	400	267729.00	34.32	267729.00	4.80
	500	328235.00	62.64	328235.00	9.95
	600	388733.00	205.90	388734.00	10.97
	700	447089.00	117.75	447089.00	16.55
	800	503200.00	200.73	503200.00	24.47
	900	557946.00	443.44	557953.00	30.43
	1000	611110.00	372.52	611110.00	52.14
	1100	665303.00	1209.21	665303.00	57.48

# Computational experiments

Table 1: Testing CPLEX.

$f_i$	$n$	$z_{mip^*}$	$t_{mipopt}$	$z_{VNDS}$	$t_{VNDS}$
$\sqrt{n}/1000$	100	9959.00	0.87	9959.00	0.19
	200	27806.00	5.10	27806.00	0.23
	300	49687.00	15.72	49687.00	0.20
	400	74711.00	31.77	74711.00	0.26
	500	99794.00	56.00	99794.00	0.30
	600	124479.00	83.39	124479.00	0.45
	700	150446.00	117.75	150446.00	0.59
	800	175042.00	167.02	175042.00	0.83
	900	199145.00	233.64	199145.00	1.05
	1000	223206.00	274.58	223206.00	1.30
	1100	246267.00	584.98	246267.00	1.56



# Computational experiments

Larger instances of size up to 15000 x 15000 were generated using the procedure of Barahona and Chudak (2005) already described.

When  $n \leq 7000$ , a 1800 MHz PC Pentium IV is used, while for  $n > 7000$ , a SUN Enterprise 10000 (with 400 MHz and 64 Gb of RAM) which is slower but has sufficient memory to handle the larger problems.

# Computational experiments

Table 2: Main results: Type I instances.

$n$	$p$	<i>Objective values</i>				<i>Time (sec.)</i>				<i>Time total</i>		<i>% gap</i>	
		B&B	Sliding	RVNS	VNDS	B&B	Sliding	RVNS	VNDS	<i>Best</i>	<i>All</i>	B&B	<i>Sliding</i>
000	15	1431038	1431013.5	1524403	1431038	34.7	25.1	0.1	15.9	41.1	69.1	0.00	0.0017
100	16	1555369	1555027.5	1666104	1555369	471.0	35.3	0.1	21.8	57.2	165.9	0.00	0.0017
300	18	1789843	1789021.3	1906447	1789843	75.4	38.2	0.1	14.2	52.5	98.5	0.00	0.0459
400	18	1904195	1903679.5	2077133	1905380	57.4	45.8	0.2	39.5	85.4	115.6	0.06	0.0892
500	18		2023878.0	2141113	2024911		205.8	0.2	94.0	299.9	387.7		0.0510
000	20		2581964.9	2790620	2590631		842.5	0.4	98.0	940.6	1505.7		0.3350
500	21		3101007.7	3422644	3106197		1923.0	0.6	56.0	1979.6	2504.3		0.1673
000	23		3602388.1	3904718	3606160		3073.1	0.3	153.3	3226.7	4078.5		0.1047
500	23		4099448.7	4504614	4116586		7943.7	0.3	113.9	8057.9	10982.2		0.4183
000	25		4581458.1	4879807	4599619		15927.2	0.5	245.1	16172.8	18342.0		0.3964
500	26		5047959.0	5476685	5077153		31329.1	0.9	183.5	31513.5	34624.9		0.5783
000	29		5517681.8	6099060	5548718		52734.9	1.2	118.4	52854.5	55923.6		0.5623

# Computational experiments

Table 3: Main results: Type II instances.

$n$	$p$	<i>Objective values</i>				<i>Time (sec.)</i>				<i>Time total</i>		<i>% gap</i>	
		B&B	Sliding	RVNS	VNDS	B&B	Sliding	RVNS	VNDS	<i>Best</i>	<i>All</i>	B&B	<i>Sliding</i>
500	62	328235	328235.0	355279	328235	1.0	1.0	1.2	24.4	26.6	51.2	0.0000	0.0000
000	77	611110	611110.0	694078	611110	5.9	4.9	0.2	11.6	16.7	96.7	0.0000	0.0000
500	85	872278	872216.0	983339	872434	124.4	9.8	0.3	171.6	181.7	345.1	0.0179	0.0250
000	92	1122577	1122498.6	1248881	1123159	495.8	24.2	0.6	138.3	163.1	440.3	0.0518	0.0580
500	104		1366092.2	1495801	1366643		105.5	1.2	485.7	592.4	965.7		0.0400
000	107	1595895	1595895.0	1748782	1595896	75.5	63.3	1.4	315.6	380.3	905.8	0.0001	0.0000
500	111		1819686.8	1999865	1820639		357.4	1.9	1106.8	1466.1	2274.7		0.0520
000	113		2042313.4	2296471	2043218		500.2	0.9	2366.6	2867.7	4030.2		0.0440
500	119		2255880.7	2526011	2256254		502.4	1.5	1767.6	2271.5	3668.4		0.0160
000	124		2466883.6	2768239	2467480		1054.5	2.3	1360.2	2417.0	4095.1		0.0240

# Computational experiments

Table 4: Main results: Type III instances.

$n$	$p$	<i>Objective values</i>				<i>Time (sec.)</i>				<i>Time total</i>		<i>% gap</i>	
		B&B	Sliding	RVNS	VNDS	B&B	Sliding	RVNS	VNDS	<i>Best</i>	<i>All</i>	B&B	<i>Sliding</i>
500	347	99794	99794.0	107984	99794	0.1	0.1	0.2	0.3	0.6	1.5	0.0000	0.0000
1000	391	223206	223206.0	247790	223206	1.1	1.1	0.7	1.3	3.1	7.4	0.0000	0.0000
1500	410	332750	332744.0	382516	332764	8.6	2.6	1.0	3.0	6.6	17.3	0.0042	0.0060
2000	453	438574	438568.5	496630	438578	68.0	4.8	2.7	30.7	38.2	110.6	0.0009	0.0023
2500	498	542203	542182.5	614880	542267	86.1	7.3	3.2	34.8	45.3	146.9	0.0118	0.0157
3000	519	642321	642309.0	736939	642321	78.3	8.8	4.2	152.1	165.1	243.5	0.0000	0.0018
3500	542	741097	741057.3	848933	741126	555.9	13.3	5.6	87.1	106.0	268.6	0.0039	0.0092
4000	570	839922	839909.5	972883	839942	205.3	21.1	5.8	162.2	189.1	393.6	0.0024	0.0039
4500	582	932428	932361.5	1069821	932597	5156.1	23.4	7.3	113.7	144.4	277.1	0.0181	0.0253
5000	600	1028249	1028235.0	1217007	1028255	1266.3	32.2	6.9	239.6	278.7	613.9	0.0006	0.0019
6000	637	1211889	1211861.0	1384204	1211932	4030.7	47.6	14.3	761.4	823.3	1036.4	0.0035	0.0058
7000	668	1392127	1392099.0	1593475	1392232	28547.7	87.8	14.9	715.5	818.88	1293.1	0.0075	0.0096
8000	701		1569292.0	1791174	1569767		117.1	22.0	1043.8	1182.9	1874.3		0.0303
9000	724		1742075.5	2022581	1742388		154.8	20.2	1528.0	1703.0	2182.9		0.0180
10000	752		1915065.1	2163915	1915562		172.0	36.3	1530.4	1738.7	2740.1		0.0259
11000	768		2079253.9	2391519	2079616		267.0	33.4	1144.2	1444.6	2686.6		0.0175
12000	789		2245167.0	2551364	2245526		316.6	40.4	3113.3	3470.3	4504.6		0.0215
13000	802		2411167.2	2702463	2411335		375.4	80.9	6266.9	6723.2	7716.4		0.0109
14000	827		2570329.3	2913820	2570792		478.7	74.1	3919.3	4472.1	7935.4		0.0180
15000	844		2733373.3	3134626	2733979		557.7	61.3	5103.4	5722.4	8500.1		0.0221

# Computational experiments

From the summary results in Tables 2, 3 and 4, the following observations are made:

- 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  $15000 \times 15000$ , the largest gap obtained is of the order of 0.03%. For Type II, the maximum gap is 0.06% for problem sizes up to  $5000 \times 5000$ , 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  $3000 \times 3000$ . Meanwhile the computation time for VNDS is very reasonable considering the problem sizes investigated. For example problems up to  $3000 \times 3000$  take only a few minutes; the largest one ( $15000 \times 15000$ ) ran for 1.8 hr. It is also interesting to note that Type III instances, the easiest for our VNDS, were the hardest for Barahona and Chudak's V&RRWC.

# Computational experiments

The sliding simplex method is capable of solving the dual exactly for the large-scale problems investigated. This is quite impressive considering that the largest problem solved has  $n + mn = 225,015,000$  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 scale SPLP's is achieved. Our largest problem solved ( $7000 \times 7000$ ) set a new record (soon to be beaten, as shown below).

# Computational experiments

A series of instances of type IV, in which different fixed costs are drawn randomly from a uniform distribution on the interval  $[\sqrt{n}/1000, \sqrt{n}/10]$ , were generated.



# Computational experiments

Table 5: Main results: Type IV instances.

$n$	$p$	<i>Objective values</i>				<i>Time (sec.)</i>				<i>Time total</i>		<i>% gap</i>	
		B&B	Sliding	RVNS	VNDS	B&B	Sliding	RVNS	VNDS	<i>Best</i>	<i>All</i>	B&B	<i>Sliding</i>
500	46	368408	368408	643882	368408	2.3	2.2	0.3	20.1	22.6	24.2	0.0000	0.0000
1000	77	578740	578740	1130397	578740	5.2	5.1	0.6	20.1	25.8	26.8	0.0000	0.0000
1500	109	740870	740870	1058235	740870	30.9	30.7	1.6	527.1	559.5	594.3	0.0000	0.0000
2000	128	915155	915155	2098798	915155	65.8	65.4	1.6	344.4	411.4	423.9	0.0000	0.0000
2500	144	1071962	1071962	1547638	1071962	109.8	109.5	3.4	134.9	247.8	262.7	0.0000	0.0000
3000	178	1193847	1193847	1864527	1193847	107.9	107.7	3.6	1191.9	1303.2	1309.7	0.0000	0.0000
3500	198	1334569	1334569	2056067	1334620	150.5	150.4	4.9	1045.5	1200.7	1318.4	0.0038	0.0038
4000	206	1438336	1438304	2218219	1438336	250.1	249.9	6.3	932.6	1188.8	1221.3	0.0000	0.0022
4500	231	1541880	1541880	2593945	1542339	298.2	297.9	6.6	1064.6	1369.1	1371.3	0.0298	0.0298
5000	246	1693821	1693782	2616424	1695554	394.3	369.7	8.2	1490.7	1868.6	1977.8	0.1023	0.1046
5500	255	1813342	1813324	2941636	1813342	468.1	462.6	8.0	2824.8	3295.4	3495.0	0.0000	0.0010
6000	281	1917477	1917477	3108396	1917477	811.1	810.9	8.7	2617.9	3437.5	3575.3	0.0000	0.0000
6500	306	2016429	2016284	3519048	2016678	812.2	551.2	8.6	4390.0	4949.7	5388.0	0.0123	0.0196
7000	305	2154829	2152592	3960288	2154829	1130.8	747.3	8.1	4627.3	5382.7	5816.5	0.0000	0.1039
8000	331	2320081	2320081	5652921	2320944	1262.0	1260.9	20.0	6895.4	8176.3	8685.3	0.0372	0.0372
9000	362	2529390	2529390	6554965	2529575	1364.1	1362.9	20.0	3510.4	4893.4	5219.5	0.0073	0.0073
10000	384	2737359	2737350	7528204	2737359	4224.6	1592.3	20.0	5262.0	6874.3	9998.4	0.0000	0.0003
11000	411	2934323	2934323	8149233	2934428	1896.5	1893.2	20.0	5812.1	7725.4	7785.4	0.0036	0.0036
12000	430	3110714	3110714	8068905	3110714	2571.3	2567.2	20.1	4930.3	7517.6	7737.5	0.0000	0.0000
13000	458	3300155	3300155	8975232	3300155	3126.1	3122.8	20.2	9530.5	12673.4	13612.0	0.0000	0.0000
14000	473	3461208	3461208	9091698	3461208	4764.8	4759.1	20.2	8168.8	12948.1	13120.9	0.0000	0.0000
15000	490	3645572	3645340	9787616	3645990	19217.6	5429.8	20.2	4898.0	10348.1	26237.3	0.0115	0.0178



# Computational experiments

The results show that:

- instances with different fixed costs are easier to solve problems of the same size and uniform distribution of users location, than instances 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 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 12000, 13000 and 14000 users; the branch-and-bound algorithm has less work to do than with instances of type II. II and especially I, although some computing time is required even when there is no duality gap to obtain an integer solution.

# Computational experiments

*p-median*

BIRCH Instances - TYPE I

		VNDS		S.Simplex		Branch and Bound	
<i>n</i>	<i>p</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>
10000	100	12428.46	883.97	12428.46	2966.35		
15000	100	18639.28	1687.63	18639.26	16897.61		
20000	100	24840.28	3012.72	24840.27	43021.83		
9600	64	11934.81	942.03	11934.81	4413.52		
12800	64	15863.79	1879.12	15863.79	8915.88		
16000	64	20004.55	2042.28	20004.55	19767.00		
19200	64	24018.33	4880.60	24018.33	37936.05		
10000	25	12455.71	410.96	12455.71	25518.55		
12500	25	15597.85	1353.64	15597.12	47536.92		
15000	25	18949.26	1681.99	18949.25	91527.82		
17500	25	21937.40	1524.05	21937.40	123991.89		
20000	25	25096.82	2262.23	25096.82	217205.60		

# Computational experiments

BIRCH Instances - TYPE II

		VNDS		S.Simplex		Branch and Bound	
<i>n</i>	<i>p</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>
10000	100	9629.56	3483.89	9624.77	6951.23	9625.60	46033.25
15000	100	15904.77	6067.38	15895.93	31038.01	15895.93	31038.01
20000	100	19982.67	19982.68	19974.57	62182.20	19979.56	159836.62
9600	64	8230.09	2175.64	8224.04	10707.92		
12800	64	10216.72	3259.07	10210.36	32809.85		
16000	64	13340.47	3753.25	13335.34	83919.72		
19200	64	15207.56	3039.95	15207.11	234902.23		
10000	25	7203.39	508.35	7203.39	131440.47		
12500	25	8605.31	580.87	8576.09	94049.62		
15000	25	9513.64	2329.05	9513.64	510352.37		
17500	25	12535.68	2287.66	12535.37	1068651.50		
20000	25	13052.81	3538.60	13022.17	454044.84		

BIRCH Instances - TYPE III

		VNDS		S.Simplex		Branch and Bound	
<i>n</i>	<i>p</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>	<i>f - obj</i>	<i>time</i>
25000	25	17312.86	650.29				
50000	25	35877.32	700.60				
75000	25	50393.09	851.13				
100000	25	72874.34	1252.25				

# Computational experiments

BIRCH Instances - TYPE I					
		Refine-VNS		RVNS	
$n$	$p$	$f - obj$	$time$	$f - obj$	$time$
10000	100	12652.39	60.34	12836.12	59.88
15000	100	18969.18	60.72	19415.91	59.12
20000	100	25461.72	59.90	26666.65	59.56
9600	64	12040.19	51.58	12058.62	59.73
12800	64	16034.26	46.03	16054.46	59.74
16000	64	20316.82	46.45	20514.65	58.23
19200	64	24343.16	60.30	24900.55	59.56
10000	25	12499.06	40.97	12489.28	58.92
12500	25	15615.30	45.26	15683.09	59.14
15000	25	19078.12	47.59	19038.40	59.26
17500	25	22035.82	49.05	22020.87	59.23
20000	25	25211.80	20.14	25213.84	59.43

# Computational experiments

BIRCH Instances - TYPE II					
		Refine-VNS		RVNS	
$n$	$p$	$f - obj$	$time$	$f - obj$	$time$
10000	100	9874.63	47.14	9858.32	59.35
15000	100	16370.46	58.08	16450.29	58.92
20000	100	20657.58	57.61	21985.56	59.17
9600	64	8379.8	52.84	8304.14	59.23
12800	64	10367.73	43.77	10377.09	59.26
16000	64	13530.79	36.89	13530.79	59.29
19200	64	15568.52	45.57	16023.56	59.37
10000	25	7248.78	46.02	7210.77	58.23
12500	25	8643.58	44	8618.98	58.14
15000	25	9618.78	41.15	9570.92	59.04
17500	25	12742.73	25.5	12587.68	59.12
20000	25	13176.01	32.7	13213.35	58.15

BIRCH Instances - TYPE III					
		Refine-VNS		RVNS	
$n$	$p$	$f - obj$	$time$	$f - obj$	$time$
25000	25	17290.04	16.72	17335.81	59.24
50000	25	35924.01	23.09	36086.01	58.97
75000	25	50385.23	30.19	50460.37	58.67
100000	25	71702.27	28.74	73335.6	59.03

# Concluding remarks

- Exploiting both the primal and the dual (or the complementary slackness conditions) reduces drastically the size of both the primal and the dual.
- This leads to solve exactly, or with a small error and a guarantee of quality, much larger SPLP and PMP problems than done before.
- Many generalizations appear to be possible.