Stabilizing column generation using Lagrangean/surrogate relaxation: an application to p-median location problems

Edson L. F. Senne
elfsenne@feg.unesp.br
FEG/UNESP - Universidade Estadual Paulista
Faculdade de Engenharia - Departamento de Matemática
12500-000 Guaratinguetá, SP - Brazil

Luiz A. N. Lorena
lorena@lac.inpe.br
LAC/INPE – Brazilian Space Research Institute
Caixa Postal 515
12.201-970 São José dos Campos, SP - Brazil

Abstract

The Lagrangean/surrogate relaxation was explored recently as a faster computational alternative to traditional Lagrangean heuristics. We combine the Lagrangean/surrogate and the traditional column generation approaches to accelerate and stabilize primal and dual bounds obtained using the reduced cost selection. The Lagrangean/surrogate multiplier modifies the reduced cost criterion, providing the selection of new productive columns. The p-median problem is the problem of locating p facilities (medians) on a network such as the sum of all the distances from each demand point to its nearest facility is minimized. Computational tests running p-median instances taken from the literature are presented.

Keywords: Location, Large-Scale Optimization, Mathematical Programming.

1 Introduction

This work describes some relationships between the Lagrangean/surrogate relaxation and the column generation process for linear programming problems. The Lagrangean/surrogate relaxation applies the local surrogate information of constraints relaxed in Lagrangean relaxation, to accelerate subgradient like methods. A local search is conducted at some initial iteration of subgradient methods, correcting wrong step sizes. The gain in computational times can be substantial for large-scale problems [18, 21].

Column generation is a powerful tool for solving large scale linear programming problems. Such linear programming problems may arise when the columns in the problem are not known in advance and a complete enumeration of all columns is not an option, or the problem is rewritten using Dantzig-Wolfe decomposition (the columns correspond to all extreme points of a certain constraint set) [3]. Column generation is a natural choice in several applications, such as the well-known cutting-stock problem, vehicle routing and crew scheduling [4, 5, 6, 12, 13, 23, 24, 25].

In a classical fashion of column generation, the algorithm iterates between a column generation sub-problem and a restricted master problem. Solving the master problem yields a certain dual solution, which is used in the sub-problem to determine whether there is any column that might be an incoming column.

The equivalence between Dantzig-Wolfe decomposition, column generation and Lagrangean relaxation optimization is well known. Solving a linear programming by Dantzig-Wolfe decomposition is the same as solving the Lagrangean dual by Kelley's cutting plane method [15]. However, in many cases a straightforward application of column generation may result in slow convergence. The Lagrangean/surrogate relaxation is showed in this paper to be an acceleration process to the column generation, generating new productive sets of columns at each algorithm iteration.

Other attempts to stabilize the dual appeared before, like the Boxstep method [16], where the optimization in the dual space is explicitly restricted to a bounded region with the current dual point as the central point. The Bundle methods [19] define a trust region

combined with penalty to prevent the next dual solution of changing too much. The Analytic center cutting plane method [7], where the goal is to avoid the current dual solution to change too dramatically, takes the analytic center of a region in the dual function instead of having the optimal dual point as next iteration. Neame [19] describes these and other recent alternative methods to stabilize the dual (du Merle et al. [8]).

The search for p-median nodes on a network is a classical location problem. The objective is to locate p facilities (medians) such as the sum of the distances from each demand point to its nearest facility is minimized. The problem is well known to be NP-hard and several heuristics have been developed for p-median problems. The combined use of Lagrangean relaxation and subgradient optimization in a primal-dual viewpoint was found to be a good solution approach to the problem [21].

The use of column generation to solve p-median problems was not sufficiently explored. The initial attempts appear to be the ones of [11] and [22]. The authors report convergence problems, even for small instances, when the number of medians is small compared to the number of candidate points in the network. This observation was also confirmed later by Galvão [10]. The solution of large-scale instances using a stabilized approach is reported by du Merle et al.[8].

We explain in this paper the column generation approach to the p-median problem, combined with the Lagrangean/surrogate relaxation. The paper is organized as follows. Section two presents p-median formulations and the traditional column generation process. The next section summarizes the Lagrangean/surrogate application to the problem and how it can be used in conjunction with the column generation process. Section four presents the algorithms and the next section gives computational results evidencing the benefits of the new approach.

2 P-median formulations and column generation

The p-median problem considered in this paper can be formulated as the following binary integer programming problem:

(Pmed): $$v(Pmed) = Min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1; \ j \in N \quad (1)$$

$$\sum_{i=1}^{n} x_{ii} = p \quad (2)$$

$$x_{ij} \leq x_{ii}; \ i, j \in N \quad (3)$$

$$x_{ij} \in \{0,1\}; \ i, j \in N \quad (4)$$

where:

$[d_{ij}]_{nxn}$ is a symmetric cost (distance) matrix, with $d_{ii} = 0$, $\forall i$;

p is the number of facilities (medians) to be located;

n is the number of nodes in the network, N = {1, ..., n};

$[x_{ij}]_{nxn}$ is the allocation matrix, with $x_{ij} = 1$ if median i is allocated to node j, and $x_{ij} = 0$, otherwise; $x_{ii} = 1$ if node i is a median and $x_{ii} = 0$, otherwise.

Constraints (1) and (3) ensure that each node j is allocated to only one node i, which must be a median. Constraint (2) determines the exact number, p, of medians to be located, and (4) gives the integer conditions.

(Pmed) is a classical formulation explored in other papers [21]. Garfinkel et al. [11] and Swain [22] applied the Dantzig-Wolfe decomposition to *(Pmed)* obtaining the following problem (set partition with a cardinality constraint):

(SP-Pmed): $v(SP\text{-}Pmed) = Min \sum_{k=1}^{m} c_k x_k$

subject to

$$\sum_{k=1}^{m} A_k x_k = 1 \qquad (5)$$

$$\sum_{k=1}^{m} x_k = p \qquad (6)$$

$$x_k \in \{0,1\},$$

where

$S = \{S_1, S_2, ..., S_m\}$, is the set of all subsets of N,

$A = [a_{ik}]_{nxm}$, is a matrix with $a_{ik} = 1$ if $i \in S_k$, and $a_{ik} = 0$ otherwise; and

$$c_k = \underset{i \in S_k}{Min} \left(\sum_{j \in S_k} d_{ij} \right).$$

For each subset S_k, the open median is decided when the cost c_k is calculated. So, the columns of *(SP-Pmed)* consider implicitly the constraints set (3) in *(Pmed)*. Constraints (1) and (2) are conserved and respectively updated to (5) and (6), according the Dantzig-Wolfe decomposition process. The same formulation is found in Minoux [17].

The number of subsets in *S* can be huge, and a partial set of columns is considered instead. In this case, problem *(SP-Pmed)* is also known as the restricted master problem in the column generation context [1].

The search for exact solutions of *(SP-Pmed)* is not an objective of this paper. We are dealing with relaxation relationships, respectively, the *Lagrangean/surrogate* (to be defined in next section) and a relaxed master problem solved by the column generation process. So, the problem to be solved by column generation is the linear programming set covering relaxation of *(SP-Pmed)*:

(SC-Pmed): $v(SC\text{-}Pmed) = Min \sum_{k=1}^{m} c_k x_k$

subject to

$$\sum_{k=1}^{m} A_k x_k \geq 1 \qquad (7)$$

$$\sum_{k=1}^{m} x_k = p$$

$$x_k \in [0, 1].$$

Observe that $d_{ij} \geq 0$, $\forall i,j$ and (5) can be replaced with (7) in the linear model. The main advantage is that problem *(SC-Pmed)* is easier to be solved than *(SP-Pmed).*

After defining an initial pool of columns, problem *(SC-Pmed)* is solved and the final dual costs p_i, i = 1,...,n and α are used to generate new columns by solving the following sub-problem:

(Sub-Pmed): $$v(Sub\text{-}Pmed) = \underset{j \in N}{Min}\left[\underset{y_{ij} \in \{0,1\}}{Min} \sum_{i=1}^{n} (d_{ij} - p_i) y_{ij} \right].$$

Each vertex in N is selected to be a median and problem *(Sub-Pmed)* is easily solved setting (for each j = 1,...,n) $y_{ij} = 1$, *if* $d_{ij} - p_i \leq 0$ and $y_{ij} = 0$, *if* $d_{ij} - p_i > 0$. Let j^* be the vertex index reaching the overall minimum on *v(Sub-Pmed)*. The new sets S_j are $\{i \mid y_{ij} =$ 1 on *(Sub-Pmed)*$\}$ and the column $\begin{bmatrix} y_{j^*} \\ 1 \end{bmatrix}$ is added to *(SC-Pmed)* if $v(Sub\text{-}Pmed) < |\alpha|$.

The *reduced cost* is $rc = v(Sub\text{-}Pmed) - \alpha$ and $rc < 0$ is the condition for incoming columns, but it is well known (see reference [1]) that, for j = 1,...,n, all the corresponding columns $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ satisfying:

$$\left[\underset{y_{ij} \in \{0,1\}}{Min} \sum_{i=1}^{n} (d_{ij} - p_i) y_{ij} \right] < |\alpha|, \qquad (8)$$

can be added to the pool of columns, possibly accelerating the column generation process.

3 The Lagrangean/surrogate integration to column generation

It is well known the equivalencies of the Dantzig-Wolfe decomposition, column generation and Lagrangean relaxation optimization. Solving a linear programming by Dantzig-Wolfe decomposition is the same as solving the Lagrangean by Kelley's cutting plane method [15]. The Lagrangean relaxation corresponding to the Dantzig-Wolfe decomposition presented in section 2 is:

$$\textbf{(LPmed}^{p,a}\textbf{):}\quad v(LPmed^{p,a}) = Min \sum_{i=1}^{n}\sum_{j=1}^{n}(d_{ij}-p_i)x_{ij} + a\left(\sum_{i=1}^{n}x_{ii} - p\right) + \sum_{i=1}^{n}p_i$$

subject to (3) and (4),

where $\pi \in R_+^n$ and α are the Lagrangean multipliers of constraints (1) and (2).

Solving *(LPmed^{p,a})* generates new cutting planes on the Kelley´s method that corresponds to new columns in *(SC-Pmed)*. Since $d_{ii} = 0$, $i = 1,\ldots,n$, the number of open medians in *(LPmed^{p,a})* depends on the number of coefficients $(a - p_i) < 0$. Defined this number, it can be identified the median given the smallest contribution to *v(LPmed^{p,a})* (and their allocated non-medians). If the $\pi \in R_+^n$ and α are dual variables coming from *(SC-Pmed)* this result to be equivalent to find the column j^* using sub-problem *(Sub-Pmed)*. The column $\begin{bmatrix} y_{j^*} \\ 1 \end{bmatrix}$ is added to *(SC-Pmed)* if $rc < 0$, as well as all the corresponding columns $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ satisfying expression (8) can be added to the pool of columns.

While the number of medians is not implicitly considered in *(Sub-Pmed)* we decide to use a relaxation only of constraints (1) (multipliers $\pi \in R_+^n$) for the planned combination of

Lagrangean/surrogate relaxation and the column generation process. The Lagrangean/surrogate relaxation for the p-median problem was presented in the work of Senne and Lorena [21]. A general description of the Lagrangean/surrogate relaxation appeared in the work of Narciso and Lorena [18]. We summarize the relaxation in this section showing how to combine it with the column generation approach.

For a given t ≥ 0 and $\pi \in R_+^n$, the *Lagrangean/surrogate* relaxation of problem *(Pmed)* is given by:

$$\textbf{(LS}_t\textbf{Pmed}^p\textbf{):} \qquad v(LS_t Pmed^p) = Min \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} - t\pi_i) x_{ij} + t \sum_{i=1}^{n} \pi_i$$

subject to (2), (3) and (4).

Problem $(LS_t Pmed^p)$ is solved considering implicitly constraint (2) and decomposing for index j, obtaining the following n problems:

$$Min \sum_{i=1}^{n} (d_{ij} - t\pi_i) x_{ij} \text{ subject to (3) and (4).}$$

Each problem is easily solved letting $b_j = \sum_{i=1}^{n} \left[Min\left(0, d_{ij} - t\pi_i\right) \right]$, and choosing I as the index set of the p smallest b_j (here constraint (2) is considered implicitly). Then, a solution x_{ij}^p to problem $(LS_t Pmed^p)$ is:

$$x_{jj}^p = \begin{cases} 1, \ if \ \ j \in I \\ 0, \ otherwise \end{cases}$$

and for all i ≠ j,

$$x_{ij}^p = \begin{cases} 1, \ if \ \ j \in I \ \ and \ \ d_{ij} - t\pi_i < 0 \\ 0, \ otherwise \end{cases}$$

The Lagrangean/surrogate solution is given by $v(LS_t Pmed^{\,p}) = \sum_{j=1}^{n} b_j x_{jj}^p + t\sum_{i=1}^{n} p_i$.

Note that x_{jj}^p is always candidate to be 1, since $(d_{jj} - tp_i) = -tp_i \leq 0$, and this allows one or more x_{ij}´s to be 1 if the corresponding $(d_{ij} - tp_i)$ are negative.

If t is set to 1 it results on the usual Lagrangean relaxation. For a fixed multiplier π, the best value for t can be found solving approximately a local Lagrangean dual $v(D_t^p) = \underset{t \geq 0}{Max}\ v(LS_t Pmed^p)$. It is well known that the function l: $R^+ \rightarrow R$, (t, $v(LS_t Pmed^p)$) is concave and piecewise linear. Figure 1 shows a typical situation for the Lagrangean/surrogate bounds.

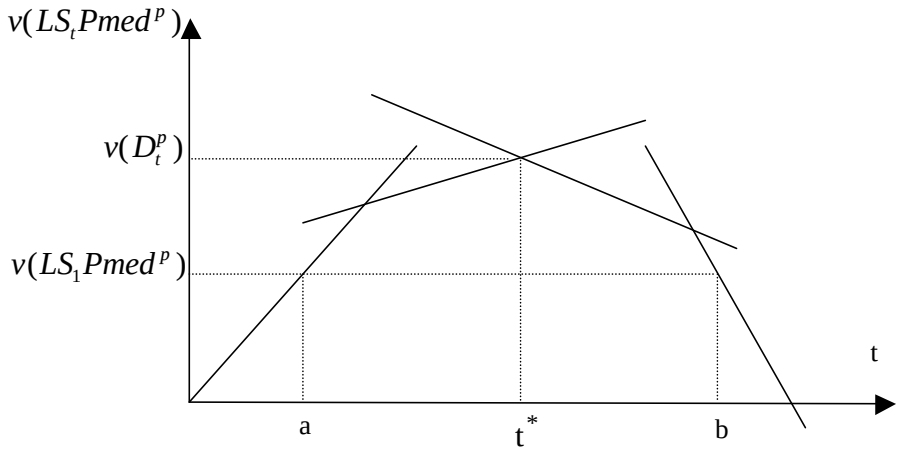


Figure 1 – Lagrangean/surrogate bounds

An exact solution to (D_t^p) may be obtained by a search over different values of t [21]. For the purpose of this paper, however, it is enough that $t \in (a, b)$ in order to obtain an improved bound in relation to the usual Lagrangean relaxation. So, a convenient value for t can be found by the following search procedure:

Let t_0 be the initial value for t, and s be the step size.

1. Set *a* = *b* = *undefined*.
2. Repeat the steps 3 and 4 while *a or b are undefined*.
   3. Solve $(LS_t Pmed^p)$ obtaining x^p and calculate the slope of the Lagrangean/surrogate function as $s^p = \sum_{i=1}^{n} p_i \left(1 - \sum_{i=1}^{n} x_{ij}^p\right)$.
   4. Set *a* = *t*, if ($s^p < 0$). Otherwise, set *b* = *t*. If *b* remains *undefined* set *t* = *t* + *s*. Otherwise, set *t* = *t* – *s*, if *a* remains undefined.
5. Find the improved value of *t* by a *dichotomous search* on *(a, b)*.

We have used t_0 = *0.1* and *s* = *0.1* in the computational tests.

The Lagrangean/surrogate problem is integrated to the column generation process transferring the multipliers p_i (i = 1,...,n) of problem *(SC-Pmed)* to the problem $\underset{t\geq 0}{Max}\, v(LS_t Pmed^p)$. The median with smallest contribution on $v[\underset{t\geq 0}{Max}\, v(LS_t Pmed^p)]$ (and allocated non-medians) results to be the one selected to produce the incoming column on the new sub-problem:

$$\textbf{(Sub}_t\textbf{Pmed):} \qquad v(Sub_t Pmed) = \underset{j\in N}{Min}\left[\underset{y_{ij}\in\{0,1\}}{Min}\sum_{i=1}^{n}\left(d_{ij} - t.p_i\right)y_{ij}\right].$$

Let j^{**} be the vertex index reaching the overall minimum on $v(Sub_t Pmed)$. The new sets S_j are $\{i \mid y_{ij} = 1$ on $(Sub_t Pmed)\}$ and the column $\begin{bmatrix} y_{j^{**}} \\ 1 \end{bmatrix}$ is added to *(SC-Pmed)* if $\left[\sum_{i=1}^{n}\left(d_{ij} - \pi_{j^{**}}\right)y_{ij^{**}}\right] < |\alpha|$, as well as the corresponding columns $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ satisfying expression (8) can be added to the pool of columns. Note that the columns generated can be different from the ones generated using *(SubPmed)*, but they are incoming columns only if satisfy the usual reduced cost tests.

Rewriting *($Sub_t Pmed$)*,

$$v(Sub_t Pmed) = \min_{j \in N} \left[\min_{y_{ij} \in \{0,1\}} \sum_{i=1}^{n} \left(d_{ij} - t.p_i \right) y_{ij} \right]$$

$$= \min_{j \in N} \left\{ \min_{y_{ij} \in \{0,1\}} \left[\sum_{i=1}^{n} \left(d_{ij} - p_i \right) y_{ij} + (1-t) p_i \sum_{i=1}^{n} y_{ij} \right] \right\},$$

and the multiplier (1 – t) can be seen as a dual variable corresponding to the following additional constraint in the master problem *(SC-Pmed)*:

$$\sum_{i=1}^{n} \sum_{k=1}^{m} p_i A_k x_k \geq \sum_{i=1}^{n} p_i \qquad (9)$$

Constraint (9) is obtained taking the dual variables π_i (i = 1,…,n) in the current master problem. The new (SC-Pmed) is:

(SC-Pmed^p): $v(SC\text{-}Pmed^p) = \min \sum_{k=1}^{m} c_k x_k$

subject to

$$\sum_{k=1}^{m} A_k x_k \geq 1$$

$$\sum_{k=1}^{m} x_k = p$$

$$\sum_{i=1}^{n} \sum_{k=1}^{m} p_i A_k x_k \geq \sum_{i=1}^{n} p_i$$

$$x_k \in [0, 1].$$

Constraint (9) is a surrogate constraint derived of constraints (5) in *(SC-Pmed)* and is considered only implicitly by the dual variable (1 – t). It follows by linear programming duality that (1 – t) ≥ 0 and as defined before, that t ≥ 0. Therefore the multiplier t is always situated in the interval [0,1].

It appears that the implicit consideration of (9) could be beneficial to the column generation process. Some columns can be anticipated in the process. We conjecture in the following that these new identified columns can be more productive for the column generation process than the ones generated by *(SubPmed)*.

The joint application of the Dantzig-Wolfe and Kelley´s methods give an indication that the Lagrangean/surrogate multiplier t must converge to 1 as the primal/dual process converge.

Comparing the sub-problems *($Sub_t Pmed$)* and *($SubPmed$)* it is easy to see that for $0 \leq t \leq 1$, if $d_{ij} - p_i > 0$ then $d_{ij} - t p_i > 0$ and in the column $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ the corresponding $y_{ij} = 0$ was not modified using multiplier t. If $d_{ij} - p_i \leq 0$ then $d_{ij} - t p_i \leq 0$ or $d_{ij} - t p_i > 0$ and in the column $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ some $y_{ij} = 1$ can be flipped to $y_{ij} = 0$. A direct consequence is that for the same multipliers p_i (i = 1,...,n), the column cost $c_k = \underset{i \in S_k}{Min}\left(\sum_{j \in S_k} d_{ij} \right)$ calculated for problem *(SC-Pmed)* can be smaller using the Lagrangean/surrogate approach. This effect is best shown on computational tests of section 5 and results on faster convergence, even when multiple columns are added to the pool at each iteration of the process.

4 The algorithms

The column generation algorithms proposed in this paper can be stated as:

CG (t)

(i) Set an initial pool of columns to *(SC-Pmed)*;

(ii) Solve *(SC-Pmed)* using the CPLEX [14] and return the duals prices p_i (i = 1,...,n) and α;

(iii) Solve approximately a local Lagrangean/surrogate dual $\underset{t \geq 0}{Max}\, v(LS_t Pmed^p)$, returning the corresponding columns of *($Sub_t Pmed$)*;

(iv) Append to *(SC-Pmed)* the columns $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ satisfying expression (8);

(v) If no columns are found in step (iv) then stop;

(vi) Perform tests to remove columns and return to step (ii).

Steps (i) and (vi) will be described below. Making $t = 1$, *CG (1)* gives the traditional column generation process. In this case, the search for t in the step (iii) is no more necessary, and the usual Lagrangean bound $(LS_1 Pmed^p)$ implicitly solves the $(Sub_1 Pmed)$ problem. In any case the bounds *v(SC-Pmed)* and $v(LS_t Pmed^p)$ are calculated at each iteration.

The following algorithm is used in step (i):

IC

Let

Num_Cols be the maximum number of columns for the initial pool of columns.

ncols = 0;
While (ncols < *Num_Cols*) do
Let $M = \{ n_1, n_2, ..., n_p \} \subset N$ be a randomly generated set of nodes.
For each k = 1, 2, ..., p do
$S_k = \{ n_k \} \cup \{ i \in N - M \mid d_{in_k} = \underset{j \in M}{Min}(d_{ij}) \}$

$$c_k = \underset{i \in S_k}{Min} \left(\sum_{j \in S_k} d_{ij} \right)$$

For j = 1,…,n do
Set $y_j = 1$ if $j \in S_k$
$y_j = 0$, otherwise
End_for
Include column $\begin{bmatrix} y_j \\ 1 \end{bmatrix}$ in the initial pool of columns.
End_For;
ncols = ncols + p;
End_While;

The algorithm used in step (vi) is the following:

RC

Let

mean_rc	be the average of the reduced costs for the initial pool of columns (after *IC* application) of *(SC-Pmed)*;
tot_cols	be the total number of columns in the current *(SC-Pmed)*;
rc_i	be the reduced cost of the columns in the current *(SC-Pmed)* (i = 1,..., *tot_cols*);
rc_factor	be a parameter to control the strength of the test.

For i = 1, ..., *tot_cols* do
    Delete column i from the current *(SC-Pmed)* if $rc_i > rc\_factor * mean\_rc$.
End_For;

5 Computational tests

The approach discussed above was programmed in *C* and run on a *Sun Ultra 30* workstation. The initial set of instances used for the tests is drawn from *OR-Library* [2]. The results are reported in the tables below (note that the symbol "–" in these tables means "null gap") and the algorithms are summarized as:

- *CG(t)* – described in section 4, and uses the column generation process combined with the Lagrangean/surrogate relaxation;
- *CG(1)* – described in section 4, and uses the traditional column generation process and also gives the Lagrangean relaxation bound;
- *LS* – described in the paper [21], and uses the Lagrangean/surrogate relaxation embedded on a dual optimized by a subgradient method.

In these tables, all the computer times do not include the time needed to setup the problem.

Table 1 reports the results for *CG(t)* and *CG(1)* (in parentheses) obtained for *rc_factor* = 1.0 and maximum number of iterations = 1000. *Table 1* contains:

- the number of nodes in the network and the number of medians to be located;
- the optimal integer solution for the instance (available in OR-Library);
- *iter* = the number of iterations;
- the total number of columns generated;
- the number of columns effectively used in the process;
- *gap_primal* = 100 * | (*v(SC-Pmed)* – optimal) | / optimal, that is, the percentage deviation from optimal to the best primal solution value *v(SC-Pmed)* found by CPLEX ;
- *gap_dual* = 100 * (optimal – $v(LS_t Pmed^p)$) / optimal, that is, the percentage deviation from optimal to the best relaxation value $v(LS_t Pmed^p)$ found;
- the total computational time (in seconds).

Table 1. Computational results for instances from OR-Library

n	p	*optimal solution*	*iter*	*columns generated*	*columns used*	*gap_primal*	*gap_dual*	*total time*
100	5	5819	184 (155)	5458 (5969)	3861 (3775)	– (–)	– (–)	36.35 (36.31)
200	5	7824	399 (381)	16929 (23630)	11763 (12533)	– (–)	– (–)	902.77 (1625.63)
200	10	5631	936 (757)	24375 (24483)	20584 (18701)	– (–)	– (–)	996.00 (864.83)
300	5	7696	1000 (919)	39299 (48431)	38173 (42704)	0.246 (–)	1.796 (–)	17889.12 (23337.79)
300	10	6634	731 (1000)	33342 (55200)	26638 (36864)	– (0.108)	– (0.215)	10749.91 (13214.36)
300	30	4374	198 (1000)	12040 (40166)	8016 (30381)	– (–)	– (0.118)	831.22 (1057.43)
400	5	8162	1000 (1000)	60624 (85762)	53181 (64266)	0.686 (0.832)	1.662 (1.022)	52807.93 (83877.77)
400	10	6999	675 (627)	41156 (66680)	26561 (26070)	– (–)	– (–)	36829.25 (41202.98)
400	40	4809	195 (191)	18160 (24213)	13130 (13101)	– (–)	– (–)	1055.20 (1078.27)

The combined use of Lagrangean/surrogate and column generation can be very interesting, especially for large-scale problems. Algorithm *CG(t)* is faster and found the same results of *CG(1)* generating a small number of columns. *Figure 2* shows that the

typical behaviors of $v(LS_1Pmed^p)$ (called here the Lagrangean bound) and $v(LS_tPmed^p)$ (called here the Lagrangean/surrogate bound) are conserved using column generation. The figure shows the values obtained at each iteration of *CG(t)* and *CG(1)* for a problem with n = 900 and p = 300.

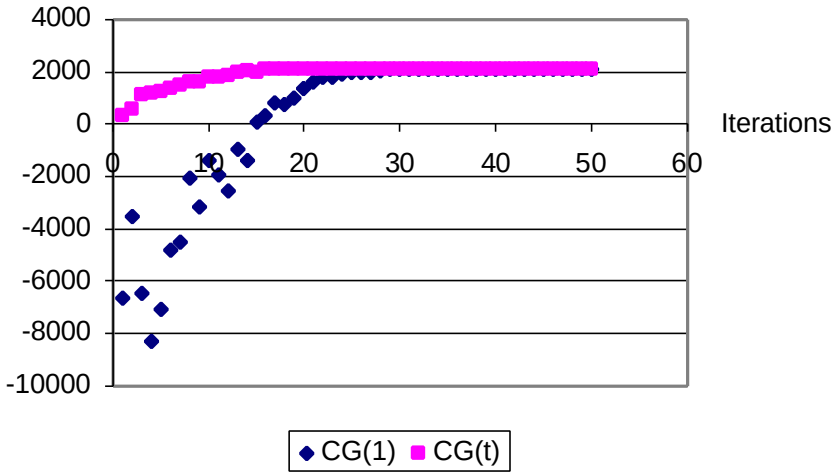


Figure 2. Typical computational behavior of the dual bounds $v(LS_1Pmed^p)$ and $v(LS_tPmed^p)$

The results of *Table 1* also show that for a given number of nodes, the smaller the number of medians in the instance, the harder is the problem to be solved using the column generation approaches *CG(t)* or *CG(1)*. The opposite occurs for Lagrangean and Lagrangean/surrogate approaches combined with subgradient search methods (algorithm *LS* in [21]), i.e., the instances for which the number of medians is about 1/3 of the number the nodes are the more difficult ones to solve.

Table 2 shows the results obtained for the set of the most time consuming instances (for *LS* approaches) from OR-Library in order to compare the *CG* approaches discussed here and the Lagrangean/surrogate approach presented in [21]. The results presented in *Table2* were obtained for *rc_factor* = 1.0 and the maximum number of iterations = 50. The columns *CG* show the results for *CG(t)* and *CG(1)* (in parentheses). For the *LS* algorithm, the gap_primal = 100 * (feasible solution – optimal)/ optimal, where the feasible solution is obtained after a local search procedure performed on the clusters identified by medians.

Table 2. Comparison of LS and CG approaches

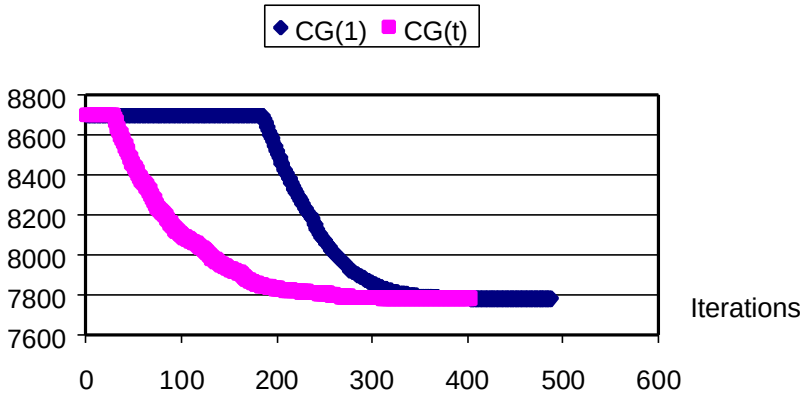
n	p	*optimal solution*	*gap_primal*		*gap_dual*		*total time*	
			LS	*CG*	*LS*	*CG*	*LS*	*CG*
100	33	1355	–	– (–)	–	– (–)	0.58	0.37 (0.35)
200	67	1255	–	– (–)	–	– (0.667)	4.00	1.29 (1.89)
300	100	1729	–	0.116 (–)	–	0.058 (–)	16.78	4.55 (4.90)
400	133	1789	–	0.112 (–)	–	0.950 (0.783)	51.80	6.21 (6.04)
500	167	1828	–	0.055 (0.036)	–	0.310 (0.210)	127.60	11.00 (12.91)
600	200	1989	–	0.302 (0.101)	–	0.285 (0.235)	257.02	15.81 (17.59)
700	233	1847	–	0.081 (0.325)	–	0.379 (0.785)	482.97	21.50 (21.41)
800	267	2026	–	0.518 (0.222)	–	0.346 (0.271)	1374.74	26.14 (27.95)
900	300	2106	0.047	0.518 (0.607)	0.004	0.827 (0.443)	3058.65	33.37 (49.99)

The instances in *Table 2* seem to be easy for *CG* approaches and hard for *LS* approach. For these instances the computational tests have confirmed the superiority of the combined use of Lagrangean/surrogate and column generation compared to the Lagrangean/surrogate embedded in a subgradient search method (note that the *LS* approach was already shown to be faster than Lagrangean heuristics in [21]).

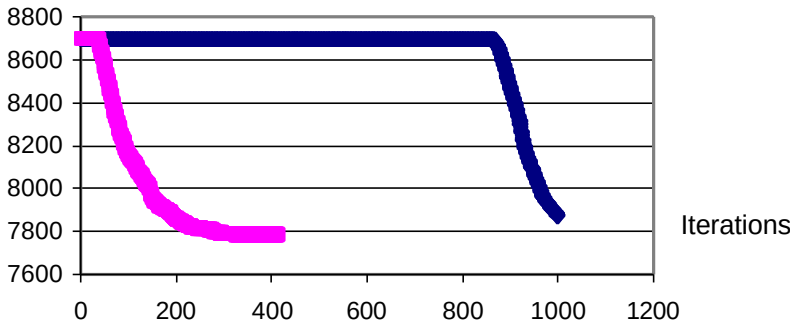
The results from *Table 1* show that *CG(t)* is able to generate fewer and higher quality columns than *CG(1)*. This becomes evident when the number of useful columns is limited by decreasing *rc_factor*, as reported by *Table 3* and shown by *Figure 3*, for the instance with n = 200 and p = 5.

Table 3 – Limiting useful columns by rc_factor

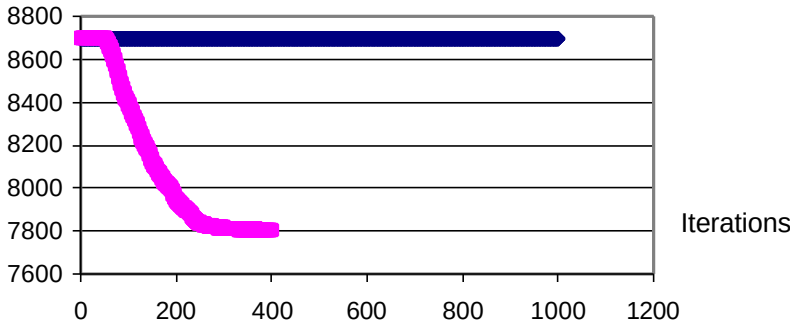
rc_factor	*iter*	*columns generated*	*columns used*	*gap_primal*	*gap_dual*	*total time*
0.5	403 (487)	18493 (47634)	7543 (7364)	– (–)	– (–)	619.63 (971.59)
0.4	414 (1000)	20395 (167247)	6627 (3270)	– (0.631)	– (4.635)	613.79 (1370.99)
0.3	400 (1000)	23521 (186267)	3886 (421)	–0.276 (11.171)	2.010 (65.181)	532.27 (905.67)



(a) results for rc_fctor = 0.5



(b) results for rc_fctor = 0.4



(c) results for rc_fctor = 0.3

Figure 3 – (SC-Pmed) values at each iteration

The results from *Table 3* and *Figure 3* show that a column generation procedure which includes a Lagrangean/surrogate algorithm *CG(t)* is able to produce high quality approximate solutions even if only a few number of columns is used. The traditional approach *CG(1)* keeps on several iterations with no improvement on the optimal value of the master problem, or it can stay unchanged all the time (see *Figure 3* for *rc_factor* = 0.3).

As commented before the Lagrangean/surrogate multiplier *t* is situated on the interval [0, 1], and as the iteration number increases, the usefulness of the surrogate inequality (9) disappears. *Figure 4* shows the values of this multiplier at each iteration of the *CG(t)* for the problem n = 300 and p = 5 presented in *Table 1*.

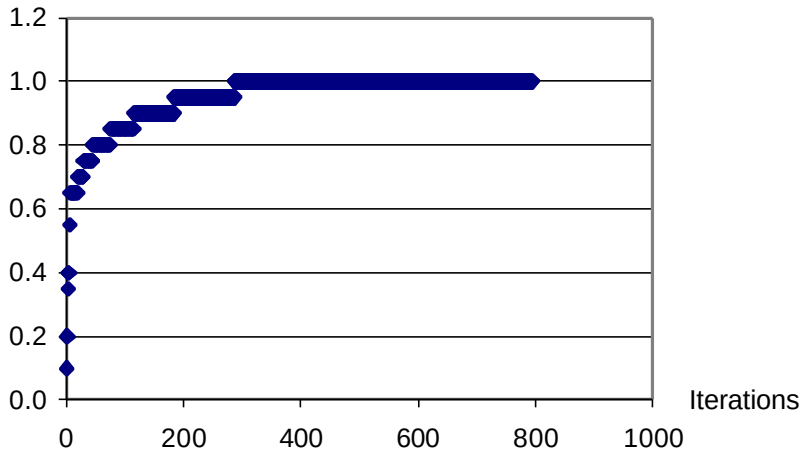


Figure 4. The increasing of the Lagrangean/surrogate multiplier

The computational tests proceeded now considering a large-scale instance. The *Pcb3038* instance in the *TSPLIB*, compiled by Reinelt [20], was considered for the tests. The results are given in *Table 4*, *Table 5* and *Table 6*. In these tables *gap_primal* and *gap_dual* are calculated as following:

- *gap_primal* = 100 * | (*v(SC-Pmed)* – best known solution) | / best known solution
- *gap_dual* = 100 * (best known solution – $v(LS_t Pmed^p)$) / best known solution

Table 4. Computational results for Pcb3038 instances (rc_factor = 1.0)

p	*best known solution*	*iter*	*columns generated*	*columns used*	*gap_primal*	*gap_dual*	*total time*
300	187723.46	42 (48)	58339 (65007)	44599 (44081)	0.043 (0.043)	0.044 (0.043)	22235.02 (35132.76)
350	170973.34	47 (37)	58758 (65545)	45576 (43956)	0.044 (0.044)	0.045 (0.045)	10505.93 (20457.59)
400	157030.46	33 (35)	50807 (60287)	37318 (39563)	0.008 (0.008)	0.008 (0.008)	4686.27 (8962.82)
450	145422.94	32 (30)	45338 (52515)	32637 (33544)	0.052 (0.052)	0.053 (0.052)	1915.84 (3241.71)
500	135467.85	22 (21)	31778 (36386)	22854 (22839)	0.036 (0.035)	0.036 (0.036)	597.86 (787.46)

Table 5. Computational results for Pcb3038 instances (rc_factor = 0.5)

p	*best known solution*	*iter*	*columns generated*	*columns used*	*gap_primal*	*gap_dual*	*total time*
300	187723.46	79 (67)	96798 (111597)	40053 (39448)	0.043 (0.043)	0.044 (0.043)	19371.01 (36029.23)
350	170973.34	65 (53)	86113 (90651)	29179 (31664)	0.044 (0.044)	0.045 (0.044)	7077.99 (12905.94)
400	157030.46	53 (49)	77174 (94716)	22857 (30101)	0.008 (0.008)	0.008 (0.008)	2872.48 (5682.90)
450	145422.94	40 (41)	55870 (80631)	18662 (23767)	0.052 (0.052)	0.052 (0.053)	1288.56 (2568.56)
500	135467.85	34 (53)	45092 (79338)	16750 (22956)	0.036 (0.036)	0.036 (0.044)	716.78 (1425.33)

Table 6. Computational results for Pcb3038 instances (rc_factor = 0.2)

p	*best known solution*	*iter*	*columns generated*	*columns used*	*gap_primal*	*gap_dual*	*total time*
300	187723.46	617 (834)	958984 (1655221)	28718 (93535)	0.043 (0.043)	0.044 (0.043)	36333.01 (117707.31)
350	170973.34	393 (719)	576789 (1232357)	24475 (74005)	0.044 (0.044)	0.044 (0.044)	10823.10 (49874.03)
400	157030.46	235 (586)	330475 (1232440)	15973 (54724)	0.008 (0.008)	0.008 (0.008)	4529.20 (39883.02)
450	145422.94	155 (363)	176348 (843026)	13489 (20517)	0.052 (0.052)	0.052 (0.052)	2356.97 (12990.88)
500	135467.85	121 (210)	119884 (420737)	12997 (24254)	0.035 (0.036)	0.035 (0.036)	1682.15 (4340.33)

The results from tables *4*, *5* and *6* confirm that *CG(t)* is really able to generate better quality columns than *CG(1)*. Evidently, if more columns are deleted by *RC* algorithm,

more iterations are necessary to reach the same results, which highlights the superiority of *CG(t)* as compared to *CG(1)*. The *rc_factor* can be viewed as a trade-off parameter to decide about available time and storage conditions.

Based on the computational tests we can draw the following overall conclusions:

- It appears that instances with small number of medians are hard to column generation approaches and easy for Lagrangean/surrogate and subgradient methods. But instances with large number of medians are easy to column generation and hard to Lagrangean/surrogate and subgradient methods. It seems that they are companion methods in this sense.
- Algorithm *CG(t)* can be used as a substitute of *CG(1)*, specially on hard instances and when the limit of generated columns is an important factor.

6 Comments and conclusion

The column generation has been recognized as a useful tool for modeling and solving large-scale linear programming problems. Despite that, the column generation application may have some computational problems, when the sub-problem generates too many columns not improving the master problem bound.

The combined use of Lagrangean/surrogate relaxation and column generation shows some improvement to the traditional column generation process. Depending on the instance both methods, the column generation and the Lagrangean/surrogate embedded with subgradient like methods, can be improved.

Algorithm *CG(t)* also calculates lower bounds, the Lagrangean/surrogate bound, that can be used, in similar way to other bounds [9], to stop the process at a convenient iterations limit. It also can be useful to branch-and-price methods [1, 25].

The *CG(t)* application to p-median problems is an alternative to Lagrangean heuristics, especially on hard instances. Some instances remains hard to column generation and more research need to be addressed to this topic.

Acknowledgments: The authors acknowledge Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP (proc. 99/06954-7) and Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (processes 302408/88-6 and 300837/89-5) for partial financial research support. The first author also acknowledges Fundação para o Desenvolvimento da UNESP – FUNDUNESP (proc. 127/2001-DFP) for partial financial support. We also acknowledge the very useful suggestions and comments of two anonymous referees.

References

1. Barnhart, C.; Johnson, E.L.; Nemhauser, G.L.; Savelsbergh, M.W.P. and Vance, P.H. *Branch-and-Price: Column Generation for Solving Huge Integer Programs*, Operations Research 46 (1998) 316-329.
2. Beasley, J.E. *Or-library: Distributing test problems by electronic mail*. Journal Operational Research Society, 41:1069-1072, 1990.
3. Dantzig, G.B. and Wolfe, P. *Decomposition principle for linear programs.* Operations Research, 8: 101-111, 1960.
4. Day, P.R. and Ryan, D.M. *Flight Attendant Rostering for Short-Haul Airline Operations*, Operations Research 45 (1997) 649-661.
5. Desrochers, M. and Soumis, F. *A Column Generation Approach to the Urban Transit Crew Scheduling Problem*, Transportation Science 23 (1989) 1-13.
6. Desrochers, M.; Desrosiers, J. and Solomon, M. *A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows*, Operations Research 40 (1992) 342-354.
7. du Merle, O.; Goffin, J.L. and Vial, J.P. *On Improvements to the Analytic Centre Cutting Plane Method* Computational Optimization and Applications 11 (1998) 37-52.
8. du Merle, O.; Villeneuve, D.; Desrosiers, J. and Hansen, P. *Stabilized column generation.* Discrete Mathematics, 194: 229-237, 1999.
9. Farley, A.A. *A note on bounding a class of linear programming problems, including cutting stock problems.* Operations Research, 38: 992-993, 1990.
10. Galvão, R.D. *A Note on Garfinkel, Neebe and Rao's LP Decomposition for the p-Median Problem.* Transportation Science, 15 (3): 175-182, 1981.
11. Garfinkel, R.S.; Neebe, W. and Rao, M.R. *An Algorithm for the M-median Location Problem.* Transportation Science 8: 217-236, 1974.

12. Gilmore, P.C. and Gomory, R.E. *A linear programming approach to the cutting stock problem.* Operations Research, 9: 849-859, 1961.

13. Gilmore, P.C. and Gomory, R.E. *A linear programming approach to the cutting stock problem - part ii.* Operations Research, 11: 863-888, 1963.

14. ILOG Inc., Cplex Division. CPLEX 6.5, 1999.

15. Kelley, J.E. *The Cutting Plane Method for Solving Convex Programs*, Journal of the SIAM 8 (1960) 703-712.

16. Marsten, R.M.; Hogan, W. and Blankenship, J. *The Boxstep method for large-scale optimization*. Operations Research, 23: 389-405, 1975.

17. Minoux, M. *A Class of Combinatorial Problems with Polynomially Solvable Large Scale Set Covering/Set Partitioning Relaxations*. RAIRO, 21 (2): 105–136, 1987.

18. Narciso, M.G.; Lorena, L.A.N. *Lagrangean/surrogate Relaxation for Generalized Assignment Problems.* European Journal of Operational Research , 114(1), 165-177, 1999.

19. Neame, P.J. *Nonsmooth Dual Methods in Integer Programming* Phd Thesis - Department of Mathematics and Statistics, The University of Melbourne March 1999.

20. Reinelt, G. *The traveling salesman problem: computational solutions for TSP applications*. Lecture Notes in Computer Science 840, Springer Verlag, Berlin, 1994.

21. Senne, E.L.F.; Lorena, L.A.N. *Lagrangean/Surrogate Heuristics for p-Median Problems*. In Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, M. Laguna and J.L. Gonzalez-Velarde (eds.) Kluwer Academic Publishers, pp. 115-130, 2000.

22. Swain, R.W. *A Parametric Decomposition Approach for the Solution of Uncapacitated Location Problems.* Management Science, 21: 955-961, 1974.

23. Valério de Carvalho, J.M. *Exact Solution of Bin-Packing Problems Using Column Generation and Branch-and-Bound*, Universidade do Minho, Departamento Produção e Sistemas Working Paper, 1996.

24. Vance, P. *Crew scheduling, cutting stock and column generation: solving huge integer programs.* PhD thesis, Georgia Institute of Technology, 1993.

25. Vance, P.H.; Barnhart, C.; Johnson, E.L. and Nemhauser, G.L. *Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound*, Computational Optimization and Applications 3 (1994) 111-130.