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

Edson Luiz França Senne  
FEG/UNESP - Universidade Estadual Paulista - Brazil

Luiz Antonio Nogueira Lorena  
LAC/INPE - Instituto Nacional de Pesquisas Espaciais - Brazil

Marcos Antonio Pereira  
LAC/INPE - Instituto Nacional de Pesquisas Espaciais – Brazil

Abstract

The Lagrangean/surrogate relaxation was explored recently as a faster computational alternative to traditional Lagrangean heuristics. We combine Lagrangean/surrogate relaxation and traditional column generation approaches to accelerate and stabilize primal and dual bounds, through an improved reduced cost selection. The Lagrangean/surrogate multiplier modifies the reduced cost criterion, resulting in the selection of more productive columns for the p-median problem, which deals with the localization of p facilities (medians) on a network in order to minimize the sum of all the distances from each demand point to its nearest facility. Computational tests running p-median instances taken from the literature are presented.

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

Correspondences to: Luiz Antonio Nogueira Lorena, LAC/INPE - Caixa Postal 515, 12.201-970 São José dos Campos, SP – Brazil

Acknowledgments: The authors acknowledge Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (proc. 350034/91-5 and 302408/88-6), for partial financial support.
1. INTRODUCTION

This paper describes some relationships between the Lagrangean/surrogate relaxation and the column generation process for linear programming problems. The Lagrangean/surrogate relaxation uses the local information of a surrogate constraint relaxed in Lagrangean way, to accelerate subgradient like methods. A local search is conducted at some initial iteration of subgradient methods, adjusting the step sizes. The reduction of computational times can be substantial for large-scale problems [19, 22].

Column generation is a powerful tool for solving large scale linear programming problems that arise when the columns of 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, 25, 26, 27].

In classical implementations of column generation methods, the algorithm iterates between a restricted master problem and a column generation sub-problem. Solving the master problem yields a dual solution, which is used to update the cost coefficients for the sub-problem that will produce the new incoming columns, depending on pricing tests.

The equivalence between Dantzig-Wolfe decomposition, column generation and Lagrangean relaxation optimization is well known. Solving a linear programming by Dantzig-Wolfe decomposition is equivalent to 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. In this paper we propose the utilization of the Lagrangean/surrogate relaxation to accelerate the column generation step, generating new productive sets of columns.
Other attempts to stabilize the dual appeared before, like the Boxstep method [17], where the optimization in the dual space is explicitly restricted to a bounded region with the current dual solution as the central point. The Bundle methods [20] define a trust region combined with penalties to prevent significant changes between consecutive dual solutions. The Analytic Center Cutting Plane method [7] takes the current analytic center of the dual function in the next iteration, instead of considering the optimal dual solution, avoiding the dual solutions to change too dramatically. Neame [20] describes these and other recent alternative methods to stabilize dual solutions (du Merle et al. [8]). See also Lubbecke and Desrosiers [16] for selected topics in column generation.

The search for $p$-median nodes on a network is a classical location problem. The objective is to locate $p$ facilities (medians) such that 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/surrogate relaxation and subgradient optimization in a primal-dual viewpoint revealed to be a good solution approach to the problem [22].

The use of column generation to solve $p$-median problems was not sufficiently explored. Initial researches appear in [11] and [24]. The authors report convergence problems, even for small instances, when the number of medians is small compared to the number of candidate nodes 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 combined approach of Lagrangean/surrogate relaxation and column generation technique to the $p$-median problem. The paper is organized as follows. Section 2 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 4 presents the algorithms and the next section shows some 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:

\[
\begin{align*}
(Pmed): \quad v(Pmed) &= \text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j=1}^{n} x_{ij} = 1, \text{ for } i \in N \quad (1) \\
& \sum_{j=1}^{n} x_{jj} = p \quad (2) \\
& x_{ij} \leq x_{jj}, \text{ for } i, j \in N \quad (3) \\
& x_{ij} \in \{0,1\}, \text{ for } i, j \in N \quad (4)
\end{align*}
\]

where:

- $[d_{ij}]_{n \times n}$ is a symmetric cost (distance) matrix, with $d_{ii} = 0$, for $i \in N$;
- $p$ is the number of facilities (medians) to be located;
- $n$ is the number of nodes in the network, $N = \{1, \ldots, n\}$;
- $[x_{ij}]_{n \times n}$ is the allocation matrix, with $x_{ij} = 1$ if node $i$ is assigned to median $j$, and $x_{ij} = 0$, otherwise; $x_{jj} = 1$ if node $j$ is a median and $x_{jj} = 0$, otherwise.

Constraints (1) and (3) ensure that each node $i$ is allocated to only one node $j$, which must be a median. Constraint (2) determines that exact $p$ nodes are to be selected for the localization of the medians and (4) gives the integer conditions. Any feasible solution for $(Pmed)$ partitions the set $N$ into $p$ disjoint subsets, defining clusters containing each one median and the nodes allocated to it.

The model $(Pmed)$ is a classical formulation explored in other papers [22]. Garfinkel et al. [11] and Swain [24] applied the Dantzig-Wolfe decomposition to $(Pmed)$ obtaining the following set partition problem with cardinality constraint:
\[
v(\text{SP-Pmed}) = \text{Min} \sum_{k=1}^{m} c_k y_k
\]

subject to
\[
\sum_{k=1}^{m} A_k y_k = 1
\]
\[
\sum_{k=1}^{m} y_k = p
\]
\[y_k \in \{0, 1\}\]

where
\[S = \{S_1, S_2, \ldots, S_m\}\] is the set of all subsets of \(N\),
\[M = \{1, 2, \ldots, m\}\],
\[A_k = [a_i]_{n \times 1}, \text{ for } k \in M; \text{ with } a_i = 1 \text{ if } i \in S_k, \text{ and } a_i = 0 \text{ otherwise, and}
\]
\[c_k = \text{Min}_{i \in S_k} \left( \sum_{j \in S_k} d_{ij} \right), \text{ for } k \in M.
\]

For each subset \(S_k\), the median node is decided when the cost \(c_k\) is calculated. So, the columns of \((\text{SP-Pmed})\) consider implicitly the constraints set (3) in \((\text{Pmed})\). 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 [18].

The cardinality of \(M\) can be huge, so a partial set of columns \(K \subset M\) is considered instead. In this case, problem \((\text{SP-Pmed})\) is also known as the \textit{restricted master problem} in the column generation context [1].

The search for exact solutions of \((\text{SP-Pmed})\) is not the objective of this paper. We intend to derive relaxation relationships, so the problem to be solved by column generation is the following linear programming relaxation of the corresponding set covering formulation for \((\text{Pmed})\):
Problem (SC-Pmed) is relaxed version of (SP-Pmed), so \( v(\text{SC-Pmed}) \leq v(\text{SP-Pmed}) \). 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 \( \mu_i, i = 1, \ldots, n \), and \( \rho \) are used to generate new columns \( \alpha_j = [\alpha_{ij}]_i \) as solutions of the following sub-problem:

\[
(\text{SubPmed}): \quad v(\text{SubPmed}) = \min_{j \in N} \left[ \min_{\alpha_{ij} \in [0,1]} \sum_{i=1}^{n} (d_{ij} - \mu_i) \alpha_{ij} \right].
\]

Problem (SubPmed) is easily solved, considering each \( j \in N \) as a median node, and setting \( \alpha_{ij} = 1 \), if \( d_{ij} - \mu_i \leq 0 \) and \( \alpha_{ij} = 0 \), if \( d_{ij} - \mu_i > 0 \). The new sets \( S_j \) are defined as \{\( i \) | \( \alpha_{ij} = 1 \) on (SubPmed)\}.

The reduced cost is \( rc = v(\text{SubPmed}) - \rho \) and \( rc < 0 \) is the condition for incoming columns. If \( j^* \) denotes the node index reaching the overall minimum for \( v(\text{SubPmed}) \), then the column \( \left[ \frac{\alpha_{j^*}}{1} \right] \) is added to (SC-Pmed) if \( v(\text{SubPmed}) < \rho \). But it is well known (see reference [1]) that every column \( \left[ \frac{\alpha_j}{1} \right] \), for \( j = 1, \ldots, n \), satisfying:
can be added to the pool of columns, possibly accelerating the column generation process.

3. THE LAGRANGEAN/SURROGATE INTEGRATION TO COLUMN GENERATION

The equivalencies of the Dantzig-Wolfe decomposition, column generation and Lagrangean relaxation optimization are well known. Solving a linear programming by Dantzig-Wolfe decomposition is equivalent as solving the corresponding Lagrangean problem by Kelley's cutting plane method [15]. The Lagrangean relaxation corresponding to the Dantzig-Wolfe decomposition for problem \((P_{med})\) presented in Section 2 is:

\[
(L_{\pi,\lambda}P_{med}): \quad v(L_{\pi,\lambda}P_{med}) = \min \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} - \pi_i) x_{ij} + \lambda \left( \sum_{j=1}^{n} x_{jj} - p \right) + \sum_{i=1}^{n} \pi_i
\]

subject to (3) and (4)

where \(\pi \in R^n\) and \(\lambda \in R\) are the Lagrangean multipliers of constraints (1) and (2), respectively.

Solving \((L_{\pi,\lambda}P_{med})\) generates new cutting planes on the Kelley's method. If \(\mu \in R^n\) and \(\rho \in R\) are dual variables associated to constraints (7) and (8) of \((SC-P_{med})\), respectively, this is equivalent to find the column \(j^*\) solving sub-problem \((SubP_{med})\). The column \(\frac{\alpha_{j^*}}{1}\) is added to \((SC-P_{med})\) if \(r \rho < 0\), as well as all the corresponding columns \(\frac{\alpha_j}{1}\) satisfying expression (9), can be added to the pool of columns.
As the number of medians is not implicitly considered in \((SubPmed)\), we decide to use a relaxation only of constraints (1), in the Lagrangean sense with multipliers \(\pi \in \mathbb{R}^n\), 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 [22]. A general description of the Lagrangean/surrogate relaxation appeared in [19]. We summarize the relaxation in this section showing how to combine it with the column generation approach.

For given \(t \in \mathbb{R}\) and \(\pi \in \mathbb{R}^n\), the Lagrangean/surrogate relaxation of problem \((Pmed)\) is formulated as:

\[
(LS_{\pi,t}Pmed): \quad \nu(LS_{\pi,t}Pmed) = \text{Min} \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} - t\pi_i)x_{ij} + t\sum_{i=1}^{n} \pi_i
\]

subject to \((2) - (4)\)

Problem \((LS_{\pi,t}Pmed)\) is solved considering implicitly constraint (2) and decomposing for index \(j\), obtaining the following \(n\) problems:

\[
\text{Min} \sum_{i=1}^{n} (d_{ij} - t\pi_i)x_{ij}
\]

subject to \((3)\) and \((4)\).

Each problem is easily solved letting \(\beta_j = \sum_{i=1}^{n} \{\min\{0, d_{ij} - t\pi_i\}\}\), and defining \(J\) as the index set of the \(p\) smallest \(\beta_j\) (here constraint (2) is considered implicitly). Then, a solution \(x_{ij}^\pi\) to problem \((LS_{\pi,t}Pmed)\) is:

\[
x_{ij}^\pi = \begin{cases} 
1, & \text{if } j \in J \\
0, & \text{otherwise}
\end{cases}
\]
and for all \( i \neq j \),

\[
x_{ij}^* = \begin{cases} 
1, & \text{if } j \in J \text{ and } d_{ij} - t\pi_i < 0 \\
0, & \text{otherwise}
\end{cases}
\]

The solution value is calculated as

\[
v(LS_{\pi,Pmed}) = \sum_{i=1}^{n} \beta_j x_{jj}^* + t \sum_{i=1}^{n} \pi_i.
\]

Note that \( x_{jj}^* \) is always candidate to be 1, since \((d_{ij} - t\pi_i) = -t\pi_i \leq 0\), and this allows one or more \( x_{ij} \)'s to be 1 if the corresponding \((d_{ij} - t\pi_i)\) are negative.

The usual Lagrangean relaxation is obtained for \( t = 1 \). For a fixed multiplier \( \pi \), the best value for \( t \) can be obtained as optimal solution of the local Lagrangean dual problem:

\[
(D_{\pi,0}): \quad v(D_{\pi,t}) = \max_{t \in R} \{v(LS_{\pi,t}, Pmed)\}.
\]

It is well known that the function \( l: R \to R, (t, v(LS_{\pi,t}, Pmed)) \) is concave and piecewise linear. Figure 1 shows a typical situation for the Lagrangean/surrogate bounds.

![Figure 1 – Lagrangean/surrogate bounds](image)
An exact solution to \((D_{\pi,t})\) may be obtained by a search over different values of \(t\) \[22\]. 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 \(a = b = \text{undefined}\).
2. Repeat the steps 3 and 4 while \(a\) or \(b\) are undefined.
3. Solve \((L_{\pi_i,Pmed})\) obtaining \(x^\pi\) and calculate the slope of the Lagrangean/surrogate function as 
   \[
   \text{slope}^\pi = \sum_{i=1}^{n} \pi_i \left( 1 - \sum_{j=1}^{n} x_{ij}^\pi \right).
   \]
4. If \(\text{slope}^\pi > 0\), set \(a = t\). Otherwise, set \(b = t\).
   If \(b\) remains undefined set \(t = t + s\).
   If \(a\) remains undefined set \(t = t - s\).
5. Find the improved value of \(t\) by a dichotomous search on \((a, b)\).

We assumed \(t_0 = 0.1\) and \(s = 0.1\) in the computational tests.

The Lagrangean/surrogate problem is integrated to the column generation process transferring the multipliers \(\mu_i\), for \(i = 1, ..., n\), of problem \((SC-Pmed)\) to the Lagrangean dual problem \(\max_{\pi} \nu(L_{\pi,Pmed})\). The median with smallest contribution on \(\nu(D_{\pi,\pi})\) (and allocated non-medians) results to be the node index selected to produce the incoming column on the new sub-problem:

\((\text{Sub}_\pi Pmed)\): \[
\nu(\text{Sub}_\pi Pmed) = \min_{j \in N} \left[ \min_{\alpha, e(0,1)} \sum_{i=1}^{n} (d_{ij} - \eta \mu_i) \alpha_{ij} \right].
\]
Let $j$ be the node index reaching the overall minimum on $v(Sub_{Pmed})$. The new sets $S_j$ are $\{i \mid \alpha_{ij} = 1 \text{ on } (Sub_{Pmed})\}$ and the column $\begin{bmatrix} \alpha_j \\ 1 \end{bmatrix}$ is added to $(SC-Pmed)$ if

$$\sum_{i=1}^{n} (d_{ij} - \mu_j)\alpha_{ij} < \rho,$$

as well as all the corresponding columns $\begin{bmatrix} \alpha_j \\ 1 \end{bmatrix}$ satisfying expression (9) can be added to the pool of columns. Note that the columns generated can be different from the ones generated using $(Sub_{Pmed})$, but they are incoming columns only if they satisfy the usual reduced cost tests.

The following two conjectures were confirmed in computational tests (presented in section 5):

- 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 converges;
- The Lagrangean/surrogate multiplier is always situated in the interval $0 \leq t \leq 1$.

Comparing sub-problems $(Sub_{Pmed})$ and $(Sub_{Pmed})$ it is easy to see that, for $0 \leq t \leq 1$, if $d_{ij} - \mu_i > 0$ then $d_{ij} - t\mu_i > 0$ and in the column $\begin{bmatrix} \alpha_j \\ 1 \end{bmatrix}$ the corresponding $\alpha_{ij} = 0$ was not modified using multiplier $t$. If $d_{ij} - \mu_i \leq 0$ then $d_{ij} - t\mu_i \leq 0$ or $d_{ij} - t\mu_i > 0$ and in the column $\begin{bmatrix} \alpha_j \\ 1 \end{bmatrix}$ some $\alpha_{ij} = 1$ can be flipped to $\alpha_{ij} = 0$. A direct consequence is that for the same multipliers $\mu_i$, the column cost $c_k = \min_{\alpha_{ij} \in S_j} \left( \sum_{j \in S_j} d_{kj} \right)$ 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.
4. ALGORITHM IMPLEMENTATION

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

**CG(t)**

(i) Set an initial pool of columns to \((SC\text{-}Pmed)\);
(ii) Solve \((SC\text{-}Pmed)\) using CPLEX [14] and return the dual prices \(\mu_i (i = 1, ..., n)\) and \(\rho\);
(iii) Solve approximately a local Lagrangean/surrogate dual \(\max_{t \geq 0} v(LS_{\mu,t}Pmed)\), returning the corresponding columns of \((Sub_{1Pmed})\);
(iv) Append the columns \(\begin{bmatrix} \alpha_j \\ -1 \end{bmatrix} \) satisfying expression (9) to \((SC\text{-}Pmed)\);
(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 not executed, and the usual Lagrangean bound \((LS_{\mu,1Pmed})\) implicitly solves problem \((Sub_{1Pmed})\). In any case the bounds \(v(SC\text{-}Pmed)\) and \(v(LS_{\mu,1Pmed})\) 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.

\(n_cols = 0;\)
While \((n_cols < Num\_Cols)\) do
  Let \(M = \{n_1, ..., n_p\} \subseteq N\) be a randomly generated set of nodes.
For each $k = 1, \ldots, p$ do

$$S_k = \{ n_i \} \cup \{ i \in N - M \mid d_{in} = \text{Min}_{j \in M} (d_{ij}) \}$$

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

For $j = 1, \ldots, n$ do

Set $\alpha_j = 1$ if $j \in S_k$

$\alpha_j = 0$ otherwise

End_for

Include column $\begin{bmatrix} \frac{\alpha_j}{1} \end{bmatrix}$ in the initial pool of columns.

End_For;

$\text{ncols} = \text{ncols} + p$;

End_While;

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

**RC**

Let

- $\text{mean}_rc$ be the average of the reduced costs for the initial pool of columns (after IC application) of (SC-Pmed);
- $\text{tot}_cols$ 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) ($i = 1, \ldots, \text{tot}_cols$);
- $rc\_factor$ be a parameter to control the strength of the test.

For $i = 1, \ldots, \text{tot}_cols$ do

Delete column $i$ from the current (SC-Pmed) if $rc_i > rc\_factor \times \text{mean}_rc$.

End_For;
5. COMPUTATIONAL TESTS

The algorithm presented in the previous section was implemented in C and executed on a Sun Ultra 30 workstation. The initial set of instances used for the tests were drawn from OR-Library [2]. The results are reported in the following tables (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 [22], 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 required to setup the problem.

Table 1 reports the results for CG(t) and CG(1) (in parentheses) obtained for rec_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;
- primal gap = 100 \times \frac{|v(SC-Pmed) − optimal|}{optimal}, or the percentual deviation from optimal to the best primal solution value v(SC-Pmed) found by CPLEX;
- dual gap = 100 \times \frac{(optimal − v(LS_{z,Pmed}))}{optimal}, or the percentual deviation from optimal to the best relaxation value v(LS_{z,Pmed}) found;
- the total computational time (in seconds).
Table 1. Computational results for instances from OR-Library

<table>
<thead>
<tr>
<th>n</th>
<th>p</th>
<th>optimal solution</th>
<th>iter</th>
<th>columns generated</th>
<th>columns used</th>
<th>primal gap</th>
<th>dual gap</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>5</td>
<td>5819</td>
<td>184</td>
<td>5458 (5969)</td>
<td>3861 (3775)</td>
<td>– (–)</td>
<td>– (–)</td>
<td>36.35 (36.31)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>194</td>
<td>16929 (23630)</td>
<td>11763 (12533)</td>
<td>– (–)</td>
<td>– (–)</td>
<td>902.77 (1625.63)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>200</td>
<td>24375 (24483)</td>
<td>20584 (18701)</td>
<td>– (–)</td>
<td>– (–)</td>
<td>3861 (3775)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>244</td>
<td>39299 (48431)</td>
<td>38173 (42704)</td>
<td>0.246 (–)</td>
<td>1.796 (–)</td>
<td>17889.12 (23337.79)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td>33342 (55200)</td>
<td>26638 (36864)</td>
<td>– (0.108)</td>
<td>– (0.215)</td>
<td>10749.91 (13214.36)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>731</td>
<td>12040 (40166)</td>
<td>8016 (30381)</td>
<td>– (–)</td>
<td>– (0.118)</td>
<td>831.22 (1057.43)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>30</td>
<td>198 (1000)</td>
<td>12040 (40166)</td>
<td>0.686 (0.832)</td>
<td>1.662 (1.022)</td>
<td>52807.93 (83877.77)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4374</td>
<td>60624 (85762)</td>
<td>53181 (64266)</td>
<td>– (–)</td>
<td>– (–)</td>
<td>36829.25 (41202.98)</td>
</tr>
<tr>
<td>400</td>
<td>5</td>
<td>1000</td>
<td>8162</td>
<td>675 (627)</td>
<td>41156 (66680)</td>
<td>26561 (26070)</td>
<td>– (–)</td>
<td>1055.20 (1078.27)</td>
</tr>
<tr>
<td>400</td>
<td>10</td>
<td>6999</td>
<td>4809</td>
<td>195 (191)</td>
<td>18160 (24213)</td>
<td>13130 (13101)</td>
<td>– (–)</td>
<td>1055.20 (1078.27)</td>
</tr>
</tbody>
</table>

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 smaller number of columns. Figure 2 shows that the typical behaviors of the Lagrangean bound $v(LS_{\pi,Pmed})$ and the Lagrangean/surrogate bound $v(LS_{\pi,Pmed})$ are conserved using column generation. The figure shows the values obtained at each iteration of $CG(t)$ and $CG(1)$ for a problem instance with $n = 900$ and $p = 300$. 
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 [22]), 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 [22]. The results presented in Table 2 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 primal gap = 100 \( \times \) (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

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p$</th>
<th>optimal solution</th>
<th>primal gap</th>
<th>dual gap</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>LS</td>
<td>CG</td>
<td>LS</td>
</tr>
<tr>
<td>100</td>
<td>33</td>
<td>1355</td>
<td>–</td>
<td>–</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(–)</td>
<td>(–)</td>
<td>(0.35)</td>
</tr>
<tr>
<td>200</td>
<td>67</td>
<td>1255</td>
<td>–</td>
<td>–</td>
<td>4.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(–)</td>
<td>0.667</td>
<td>4.55</td>
</tr>
<tr>
<td>300</td>
<td>100</td>
<td>1729</td>
<td>0.116</td>
<td>–</td>
<td>16.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(–)</td>
<td>0.058</td>
<td>(4.90)</td>
</tr>
<tr>
<td>400</td>
<td>133</td>
<td>1789</td>
<td>0.112</td>
<td>–</td>
<td>51.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(–)</td>
<td>0.950</td>
<td>(6.04)</td>
</tr>
<tr>
<td>500</td>
<td>167</td>
<td>1828</td>
<td>0.055</td>
<td>–</td>
<td>127.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0.036)</td>
<td>0.310</td>
<td>(12.91)</td>
</tr>
<tr>
<td>600</td>
<td>200</td>
<td>1989</td>
<td>0.302</td>
<td>–</td>
<td>257.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0.101)</td>
<td>0.285</td>
<td>(17.59)</td>
</tr>
<tr>
<td>700</td>
<td>233</td>
<td>1847</td>
<td>0.081</td>
<td>–</td>
<td>482.97</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0.325)</td>
<td>0.379</td>
<td>(21.41)</td>
</tr>
<tr>
<td>800</td>
<td>267</td>
<td>2026</td>
<td>0.518</td>
<td>–</td>
<td>1374.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0.222)</td>
<td>0.346</td>
<td>(27.95)</td>
</tr>
<tr>
<td>900</td>
<td>300</td>
<td>2106</td>
<td>0.047</td>
<td>0.518</td>
<td>3058.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(0.607)</td>
<td>(0.443)</td>
<td>(49.99)</td>
</tr>
</tbody>
</table>

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 [22]).

The results from Table 1 show that CG(t) is able to generate fewer and higher quality columns than CG(I). 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 $rc\_factor$

<table>
<thead>
<tr>
<th>$rc_factor$</th>
<th>iter</th>
<th>columns generated</th>
<th>columns used</th>
<th>primal gap</th>
<th>dual gap</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>403(487)</td>
<td>18493(47634)</td>
<td>7543(7364)</td>
<td>–</td>
<td>–</td>
<td>619.63 (971.59)</td>
</tr>
<tr>
<td>0.4</td>
<td>414(1000)</td>
<td>20395(167247)</td>
<td>6627(3270)</td>
<td>–</td>
<td>–</td>
<td>613.79 (1370.99)</td>
</tr>
<tr>
<td>0.3</td>
<td>400(1000)</td>
<td>23521(186267)</td>
<td>3886(421)</td>
<td>–0.276</td>
<td>2.010</td>
<td>532.27 (905.67)</td>
</tr>
</tbody>
</table>

The results from Table 3 and Figure 3 shows 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 in section 3, the computational tests revealed that the Lagrangean/surrogate multiplier $t$ is always situated on the interval $[0,1]$ , and as the iteration number increases it goes to 1. 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 3 – (SC-Pmed) values at each iteration
The computational tests proceeded now considering a large-scale instance. The Pcb3038 instance in the TSPLIB, compiled by Reinelt [21], was considered for the tests. The results are given in Table 4, Table 5 and Table 6. In these tables, primal gap and dual gap are calculated as following:

\[ \text{primal gap} = 100 \times \frac{|v(\text{SC-Pmed}) - \text{best known solution}|}{\text{best known solution}} \]
\[ \text{dual gap} = 100 \times \frac{\text{best known solution} - v(\text{LS}_{\text{Pmed}})}{\text{best known solution}} \]

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

<table>
<thead>
<tr>
<th>p</th>
<th>best known solution</th>
<th>iter (columns generated)</th>
<th>columns used</th>
<th>primal gap (0.043)</th>
<th>dual gap (0.043)</th>
<th>total time (35132.76)</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>187723.46</td>
<td>42 (48)</td>
<td>58339 (65007)</td>
<td>44599 (44081)</td>
<td>0.043</td>
<td>0.044</td>
</tr>
<tr>
<td>350</td>
<td>170973.34</td>
<td>47 (37)</td>
<td>58758 (65545)</td>
<td>45576 (43956)</td>
<td>0.044</td>
<td>0.045</td>
</tr>
<tr>
<td>400</td>
<td>157030.46</td>
<td>33 (35)</td>
<td>50807 (60287)</td>
<td>37318 (39563)</td>
<td>0.008</td>
<td>0.008</td>
</tr>
<tr>
<td>450</td>
<td>145422.94</td>
<td>32 (30)</td>
<td>45338 (52515)</td>
<td>32637 (33544)</td>
<td>0.052</td>
<td>0.053</td>
</tr>
<tr>
<td>500</td>
<td>135467.85</td>
<td>22 (21)</td>
<td>31778 (36386)</td>
<td>22854 (22839)</td>
<td>0.036</td>
<td>0.036</td>
</tr>
</tbody>
</table>
Table 5. Computational results for Pcb3038 instances ($rc_{factor} = 0.5$)

<table>
<thead>
<tr>
<th>$p$</th>
<th>best known solution</th>
<th>iter</th>
<th>columns generated</th>
<th>columns used</th>
<th>primal gap</th>
<th>dual gap</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>187723.46</td>
<td>79 (67)</td>
<td>96796 (11597)</td>
<td>40053 (39448)</td>
<td>0.043 (0.043)</td>
<td>0.044 (0.043)</td>
<td>19371.01 (36029.23)</td>
</tr>
<tr>
<td>350</td>
<td>170973.34</td>
<td>65 (53)</td>
<td>86113 (90651)</td>
<td>29179 (31664)</td>
<td>0.044 (0.044)</td>
<td>0.045 (0.044)</td>
<td>7077.99 (12905.94)</td>
</tr>
<tr>
<td>400</td>
<td>157030.46</td>
<td>53 (49)</td>
<td>77174 (94716)</td>
<td>22857 (30101)</td>
<td>0.008 (0.008)</td>
<td>0.008 (0.008)</td>
<td>2872.48 (5682.90)</td>
</tr>
<tr>
<td>450</td>
<td>145422.94</td>
<td>40 (41)</td>
<td>55870 (80631)</td>
<td>18662 (23767)</td>
<td>0.052 (0.052)</td>
<td>0.052 (0.053)</td>
<td>1288.56 (2568.56)</td>
</tr>
<tr>
<td>500</td>
<td>135467.85</td>
<td>34 (53)</td>
<td>45092 (79338)</td>
<td>16750 (22956)</td>
<td>0.036 (0.036)</td>
<td>0.036 (0.044)</td>
<td>716.76 (1425.33)</td>
</tr>
</tbody>
</table>

Table 6. Computational results for Pcb3038 instances ($rc_{factor} = 0.2$)

<table>
<thead>
<tr>
<th>$p$</th>
<th>best known solution</th>
<th>iter</th>
<th>columns generated</th>
<th>columns used</th>
<th>primal gap</th>
<th>dual gap</th>
<th>total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>187723.46</td>
<td>617 (854)</td>
<td>958984 (1655221)</td>
<td>28718 (93535)</td>
<td>0.043 (0.043)</td>
<td>0.044 (0.043)</td>
<td>36333.01 (117707.31)</td>
</tr>
<tr>
<td>350</td>
<td>170973.34</td>
<td>393 (719)</td>
<td>576789 (1232357)</td>
<td>24475 (74005)</td>
<td>0.044 (0.044)</td>
<td>0.044 (0.044)</td>
<td>10823.10 (49874.03)</td>
</tr>
<tr>
<td>400</td>
<td>157030.46</td>
<td>235 (586)</td>
<td>330475 (1232440)</td>
<td>15973 (54724)</td>
<td>0.008 (0.008)</td>
<td>0.008 (0.008)</td>
<td>4529.20 (39883.02)</td>
</tr>
<tr>
<td>450</td>
<td>145422.94</td>
<td>155 (363)</td>
<td>176348 (843026)</td>
<td>13489 (20517)</td>
<td>0.052 (0.052)</td>
<td>0.052 (0.052)</td>
<td>2356.97 (12990.88)</td>
</tr>
<tr>
<td>500</td>
<td>135467.85</td>
<td>121 (210)</td>
<td>119884 (420737)</td>
<td>12997 (24254)</td>
<td>0.035 (0.036)</td>
<td>0.035 (0.036)</td>
<td>1682.15 (4340.33)</td>
</tr>
</tbody>
</table>

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, 23].

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.
REFERENCES


