Volume 40
Issue 12
December 2013
ISSN 0305-0548
ELSEVIER
computers & operations research
Including a special section on
GRASP with path relinking: Developments and applications
Guest Editors:
Rafael Marti, Mauricio G. C. Resende and Celso C. Ribeiro
Editor: Stefan Nickel
Available online at www.sciencedirect.com
ScienceDirect

Computers & Operations Research 40 (2013) 3147–3158

Contents lists available at ScienceDirect

Computers & Operations Research

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

A hybrid column generation with GRASP and path relinking for the network load balancing problem

Dorabella Santos [a,*], Amaro de Sousa [b], Filipe Alvelos [c]

[a] Instituto de Telecomunicações, 3810-193 Aveiro, Portugal
[b] Instituto de Telecomunicações/DETI, Universidade de Aveiro, 3810-193 Aveiro, Portugal
[c] Centro Algoritmi/Dep. Produção e Sistemas, Universidade do Minho, 4710-057 Braga, Portugal

ARTICLE INFO

Available online 18 May 2013

Keywords:
GRASP with path relinking
Column generation
Hybrid meta-heuristics
Network load balancing

ABSTRACT

In this paper, a hybrid meta-heuristic is proposed which combines the GRASP with path relinking method and Column Generation. The key idea of this method is to run a GRASP with path relinking search on a restricted search space, defined by Column Generation, instead of running the search on the complete search space of the problem. Moreover, column generation is used not only to compute the initial restricted search space but also to modify it during the whole algorithm. The proposed heuristic is used to solve the network load balancing problem: given a capacitated telecommunications network with single path routing and an estimated traffic demand matrix, the network load balancing problem is the determination of a routing path for each traffic commodity such that the network load balancing is optimized, *i.e.*, the worst link load is minimized, among all such solutions, the second worst link load is minimized, and continuing in this way until all link loads are minimized. The computational results presented in this paper show that, for the network load balancing problem, the proposed heuristic is effective in obtaining better quality solutions in shorter running times.

1. Introduction

The Greedy Randomized Adaptive Search Procedure (GRASP) is a meta-heuristic first introduced in [1] for the set covering problem. It is a multi-start local search method where, at each start, a solution is randomly generated (with some greediness) and local search is applied to it to find a local optimum solution. During the multiple starts, the best local optimum solution is saved as the incumbent solution. path relinking (PR), originally proposed as an intensification method applied to tabu search in [2], is a method that tries to find better solutions by the combination of two initial solutions. The common combination of GRASP with PR is to run PR at the end of each GRASP iteration between its local optimum solution and one solution randomly selected from a list of previously found elite solutions (for a good survey on the many applications where GRASP with path relinking has been applied, please see [3]).

Usually, GRASP with PR (GRASP+PR) has been proposed to be applied on the complete search space of the optimization problems. The motivation is that constraining the search space might leave out of the search for all (or, at least, some) optimal solutions. Nevertheless, there is a computational burden on this approach. The GRASP search takes more computation time per iteration (since the neighbor set is usually large) and might take a significant running time in the analysis of bad neighbor solutions (although, PR alleviates this burden since it is applied to pairs of already good quality solutions). In this paper, we take a different approach. The key idea is to apply GRASP+PR on a restricted search space, instead of the complete search space of the problem. Constraining the search to a restricted space aims to make the search more efficient (since the search space is reduced) but this can only be effective if the considered restricted space keeps the best solutions. To manage the restricted search space, we use Column Generation.

Assume that the optimization problem is defined in such a way that a solution of a problem is a combination of solutions of smaller sub-problems (SPs). Column generation (CG) is a decomposition approach, driven by linear programming, which enables each SP to be tackled by any optimization algorithm. Although the first proposals have more than 50 years [4,5], CG is still an active field of research because of two main reasons. First, the availability of computational tools has turned the implementation of CG based algorithms more robust, exposing their advantages over other approaches where no decomposition or less computationally demanding decomposition methods (such as sub-gradient methods) are used. Second, CG has been successfully applied on a wide range of problems, such as the ones mentioned in [6,7].

* Corresponding author.
E-mail addresses: dorabella@av.it.pt (D. Santos), asou@ua.pt (A. de Sousa), falvelos@dps.uminho.pt (F. Alvelos).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.05.006

In our approach, the initial restricted search space is composed by the columns generated by CG solving the linear programming (LP) relaxation of the original problem. Then, during the GRASP +PR search, the restricted search space is modified by including new columns and/or excluding existing columns. The new columns are generated by CG solving a perturbed problem which is defined based on the current incumbent solution (defined as the best solution found so far) and on the LP value of the problem (determined at the beginning when the first search space is computed). In general, a perturbed problem is defined by adding some constraints to the original problem in order to either force and/or prevent some properties exhibited by the current incumbent solution. This approach may be seen as a particular case of the general framework for combining CG and meta-heuristics entitled SearchCol (meta-heuristic search by Column Generation) [8] and we apply this approach to the network load balancing problem.

Consider a capacitated telecommunications network supporting a set of commodity demands, with an estimated set of demand commodities, where each commodity must be supported by the network through a single routing path. The balancing of the network traffic load is an important traffic engineering objective that maximizes the robustness of the network to unpredictable traffic growth.

The most straightforward way of defining the network load balancing objective is to minimize the worst link load. If the worst link load is a, with $0 \le a \le 1$, then all commodity demands can uniformly grow up to $(1-a)/a$ before the network becomes saturated and, therefore, the lower the value of a is, the more robust the network becomes to unpredictable demand growth. In [9], appropriate mathematical formulations were presented for this objective function, showing that the case where demand bifurcation is allowed is easily solved through linear programming but the single path routing case is NP-hard.

It might happen that there are different solutions with the same minimum worst link load a (in the general case, there are). Among such solutions, if the second worst link load is b, with $0 \le b \le a$, then all commodity demands not routed through the link with the worst link load can uniformly grow up to $(1-b)/b$ before the network becomes saturated. Therefore, value b must also be minimized provided that the minimum worst link load a is maintained. Generalizing this idea to all other link loads, we reach a complete definition of the network load balancing objective: to minimize the worst link load; among all such solutions, to minimize the second worst link load; and continuing in this way until all link loads are minimized. This load balancing objective is related with the concept of lexicographical minimization (as will be seen in Section 3) and of min–max fairness and general methods have been addressed in previous works, as in [10,11], to deal with it. The lexicographical minimization concept has also been recently applied as a network routing hop minimization objective, in [12], aiming, in that case, to optimize the delay suffered in the network by all commodities.

An alternative approach, proposed in [13], to define a load balancing optimization function (and further exploited in [14]) is to minimize the sum of all link costs where the cost of a link is an increasing piecewise linear function of its load. The main merit of such an approach is that the optimization problem can be defined by a single integer linear programming model, although the resulting models are hard to solve through exact methods (in fact, meta-heuristics are proposed in [13,14] to solve the resulting load balancing problems). Moreover, such an approach is an approximation of the network load balancing objective. Consider an example of a network with 5 links with two possible routing solutions: S_1 with link load values 0.7, 0.4, 0.4, 0.4 and 0.3 (sorted in a non-increasing order), and S_2 with link load values 0.5, 0.5, 0.5, 0.5 and 0.4 (also sorted in a non-increasing order). According to [13], S_1 is better than S_2 since it has a cost value of 3.57 while S_2 has a higher cost value of 3.87. Nevertheless, S_2 is better since its worst link load is 0.5 (it can cope with a demand growth of 100% of all commodities) while the worst link load of S_1 is 0.7 (it can only cope only with a growth of 42.8% of all commodities).

The network load balancing optimization problem considered in this paper was also recently addressed in [15,16]. In [15], CG is used to define the restricted search space but this space is kept constant while the search is conducted. In that work, GRASP+PR is also used as the search algorithm but PR is applied only between the local minimum solution of each GRASP iteration and the incumbent solution (*i.e.*, no PR elite list is used). That approach was compared to the equivalent GRASP+PR search applied to the complete search space of the problem and the computational results have shown that the restricted search space enabled much better results. In [16], an extended version of [15] is proposed where the GRASP+PR is periodically interrupted (periodicity time is an input parameter) and new columns are inserted in the search space using CG. That work has shown that the additional columns either improve the efficiency of the algorithm or, at least, show similar efficiency in the cases where the previous approach is already very efficient. The present work further extends the algorithm presented in [16] by (i) using an elite list where PR is applied with all its solutions (and not only with the incumbent solution), (ii) adopting a dynamic criteria to decide when the GRASP+PR search is interrupted to run CG, (iii) letting the search space be modified also by the exclusion of columns, and (iv) using different perturbed problem definitions. The computational results will show that this approach is much more efficient since, now, better results are always obtained even for the cases where the initial approach [15] is already very efficient.

This paper is organized as follows. In Section 2, we present our general hybrid meta-heuristic for solving any optimization problem provided that its linear relaxation can be solved through CG. In Section 3, we define the network load balancing problem and describe how it can be solved through mathematical programming. In Section 4, we explain how the general hybrid meta-heuristic is realized to solve the network loading problem. In Section 5, we describe a set of problem instances and present the computational results together with their analysis. Finally, in Section 6, we summarize the main conclusions and identify the topics for further research.

2. Hybrid column generation with GRASP and path relinking

The key idea of our approach is to run a GRASP+PR search on a restricted search space defined by CG instead of running GRASP +PR on the complete search space of the problem. Moreover, CG is used not only to compute the initial search space but also to modify it during the whole algorithm. Running the search on a restricted space aims to make the search more efficient provided that the restricted space still contains good quality solutions.

At the beginning, the search space is composed by the columns generated by CG when solving the LP relaxation of the original problem. Note that, besides defining the initial search space, this initial CG step also determines the optimal LP value of the original problem which is used afterwards. During the GRASP+PR search, the restricted search space is modified by including new columns and/or excluding existing ones.

The inclusion of new columns is done as follows. First, a perturbed problem is defined which is based on the current incumbent solution of GRASP+PR and on the LP value of the original problem (obtained in the initial CG step). This operation might generate new columns that are included in the restricted search space. The perturbed problem is highly dependent on the targeted optimization problem. In general, a perturbed problem is defined by adding some constraints to the original problem in order to either force and/or prevent some properties exhibited by the current incumbent solution. The aim is that the additional columns generated by CG, while solving the perturbed problem, when added to the search space, might produce a new restricted search space with better quality solutions.

The exclusion of existing columns is done as follows. During the search, we count the number of times each column is included on the different local optimum solutions found by GRASP+PR. When the restricted search space is modified, the columns not present in any local optimum solution are excluded from the restricted search space. The motivation is straightforward: if these columns are not present in any previously computed local optimum solution, they probably do not produce good quality solutions and the overall quality of the restricted search space is improved without them.

We start by presenting in Algorithm 1 the pseudo-code of the method when no columns are excluded. In this pseudo-code, P is the original problem, LB is the LP lower bound of P, P' is a perturbed problem, Ω is a set of columns defining a restricted search space, Φ is a set of new columns generated by a CG step and I is the incumbent solution.

Algorithm 1. Pseudo-code of the method without exclusion of columns.

```
1: $(\Phi, LB) \leftarrow ColumnGeneration(P, \{\})$
2: $\Omega \leftarrow \Phi$
3: while $runtime < MaxTime$ do
4:   $I \leftarrow GRASPwithPR(\Omega, Criteria, runtime)$
5:   $P' \leftarrow Perturbation(I, LB)$
6:   $\Phi \leftarrow ColumnGeneration(P', \Omega)$
7:   $\Omega \leftarrow \Omega \cup \Phi$
8: end
```

The algorithm starts by solving the LP relaxation of the original problem P, in step 1, with procedure *ColumnGeneration*. This procedure takes as input problem P and an empty set of columns and returns the set of generated columns Φ and the LP value of its solution LB. Then, in step 2, the algorithm sets the restricted search space Ω as the set of columns Φ.

Afterwards, the algorithm runs the while cycle (steps 3–8) until a maximum running time is reached (given by parameter *MaxTime*). Inside each cycle, the algorithm runs the *GRASPwithPR* procedure on the restricted search space defined by Ω until the *Criteria* is met (step 4) and returns its incumbent solution I. The *GRASPwithPR* procedure also takes the current *runtime* as an input parameter, which is used by *Criteria* to stop the procedure if the maximum running time, given by *MaxTime*, is reached. Then, the algorithm defines a perturbed problem P', in step 5, with procedure *Perturbation* based on the current incumbent I and the LB value (previously computed in step 1). The LP relaxation of the perturbed problem P' is solved in step 6, where the procedure *ColumnGeneration* takes as input the set of columns Ω and returns the set Φ of additional generated columns. Finally, the algorithm includes in step 7 the additional columns (set Φ) into the restricted search space Ω before repeating the cycle.

Note that in Algorithm 1, we have also used the set of columns Ω defining the restricted search space as the set of columns to be provided as input to CG. To define the method when columns are excluded, we need to define two additional sets of columns. We use Θ to define the whole set of columns generated by the different CG procedures and E to define the set of columns to be excluded. With this additional notation, Algorithm 2 presents the pseudo-code of the generic procedure with exclusion of columns.

Algorithm 2. Pseudo-code of the method with exclusion of columns.

```
1: $(\Phi, LB) \leftarrow ColumnGeneration(P, \{\})$
2: $\Omega \leftarrow \Phi$
3: $\Theta \leftarrow \Phi$
4: while $runtime < MaxTime$ do
5:   $(I, E) \leftarrow GRASPwithPR(\Omega, Criteria, runtime)$
6:   $P' \leftarrow Perturbation(I, LB)$
7:   $\Phi \leftarrow ColumnGeneration(P', \Theta)$
8:   $\Theta \leftarrow \Theta \cup \Phi$
9:   $\Omega \leftarrow (\Omega \cup \Phi) - E$
10: end
```

Compared with Algorithm 1, Algorithm 2 has two additional steps (steps 3 and 8) which always maintain the set Θ with all the generated columns and this is the set used as input to all CG procedures (step 7). Moreover, procedure *GRASPwithPR* also returns, in step 5, the set E of columns that were not present in any local optimal solution. Finally, the restricted search space Ω is modified, in step 9, not only by including the new columns of set Φ (as in step 7 of Algorithm 1) but also by excluding the columns of set E.

Note that the inclusion of new columns during the search enlarges the restricted search space and, therefore, can be seen as a diversification technique making the search proceed on a larger space. On the other hand, the exclusion of columns can be seen as an intensification technique since the restricted search space is reduced to the columns already used in the local optimum solutions found on the previous search.

This method is a generic procedure that can be applied to solve any optimization problem P, provided that the linear relaxation of P can be solved through CG. To define a heuristic algorithm based on this method to a particular problem, some of its components are problem dependent, namely, the *Perturbation* procedure (which defines how a perturbed problem P' is set based on an incumbent I and on the LP relaxation value of P), the *Criteria* definition (which states when the GRASP+PR search is interrupted to enable the modification of the restricted search space) and the GRASP+PR problem dependent parameters. In the next section, we define the load balancing optimization problem and then, in Section 4, we explain how these components are defined for this particular optimization problem.

3. Network load balancing optimization

Consider a telecommunications network modeled on a graph $G(N,A)$ where N is the set of network nodes and A is the set of network links connecting nodes. The link between nodes $i \in N$ and $j \in N$ is denoted by $\{i,j\}$ and each link $\{i,j\} \in A$ has a given capacity $c_{\{ij\}}$. Consider a set of commodities K, where each commodity $k \in K$ is to be routed through a single path on the network and is characterized by its origin node $o_k \in N$, its destination node $d_k \in N$ and its demand $b_k > 0$.

Let P_k be the set of paths available on graph G between the end nodes of $k \in K$ and let $\delta^{pk}_{\{ij\}}$ be a binary parameter that is 1 if link $\{i,j\} \in A$ is in the path $p \in P_k$. To model the optimization problem, we consider the following decision variables: the binary variables ϕ^p_k which are 1 if path $p \in P_k$ is chosen as the routing path of commodity $k \in K$; and the real variables $\mu_{\{ij\}}$ accounting for the load of link $\{i,j\} \in A$. The following set of constraints defines the set of feasible solutions

$$\sum_{p \in P_k} \phi_k^p = 1 \quad \forall k \in K \tag{1}$$

$$\sum_{k \in K} \sum_{p \in P_k} b_k \delta_{\{ij\}}^{pk} \phi_k^p = c_{\{ij\}} \mu_{\{ij\}} \quad \forall \{i,j\} \in A \tag{2}$$

$$\phi_k^p \in \{0,1\}\ ,\ \mu_{\{ij\}} \in [0,1] \tag{3}$$

Constraints (1) guarantee that exactly one path of P_k is chosen for each $k \in K$, constraints (2) account for the load of each link, and constraints (3) are the variables domain constraints.

The load balancing optimization problem uses the concept of lexicographical optimization. Given two vectors $a=(a_1, \ldots, a_m)$ and $b=(b_1, \ldots, b_m)$, vector a is said to be lexicographically smaller than vector b if either $a_1 < b_1$ or if there exists an index $l \in \{1, \ldots, m-1\}$ such that $a_i = b_i$ for all $i \le l$ and $a_{l+1} < b_{l+1}$. Now consider the vector of link loads $\mu = (\mu_{\{ij\}}: \{i,j\} \in A)$ and let $[\mu]$ be the vector obtained from μ by rearranging its elements in a non-increasing order. The load balancing optimization problem can be defined in a non-linear manner as

lexmin $[\mu]$

Subject to

$$(1)\text{–}(3) \tag{4}$$

where **lexmin** denotes the lexicographical minimization of $[\mu]$, *i.e.*, finding a vector $[\mu*]$ which is lexicographically minimal among all possible vectors $[\mu]$.

It is known that the solution of the load balancing optimization problem can be obtained by solving a sequence of mixed integer linear programming models. One such method is the conditional means approach (for details, please see [17,18]). First, we consider the vector θ of the accumulated elements of $[\mu]$

$$\theta = (\theta_l = \textstyle\sum_{t=1}^{l} [\mu]_l : l = 1, \ldots, |A|) \tag{5}$$

where $[\mu]_l$ is the lth element of $[\mu]$ and θ_l is the lth element of θ. Note that the load balancing optimization problem (4) is equivalent (in the sense that the optimal solution set is the same) to

lexmin θ

Subject to

$$(1)\text{–}(3) \tag{6}$$

Now, considering the additional variables r_t for $t=1, \ldots, |A|$, and $d_{\{ij\}}^t$ for $t=1, \ldots, |A|$ and $\{i,j\} \in A$, the conditional means approach states that the optimal value θ_l^* of the lth element of θ can be obtained by solving the following mixed integer linear programming model:

$$\theta_l^* = \min\left(l r_l + \sum_{\{i,j\} \in A} d_{\{ij\}}^l \right) \tag{7}$$

Subject to

(1)–(3)

$$\mu_{\{ij\}} \le r_t + d_{\{ij\}}^t \quad \forall t \in \{1, 2, \ldots, l\}, \forall \{i,j\} \in A \tag{8}$$

$$t r_t + \sum_{\{i,j\} \in A} d_{\{ij\}}^t \le \theta_t^* \quad \forall t \in \{1, 2, \ldots, l-1\} \tag{9}$$

$$r_t \ge 0,\ d_{\{ij\}}^t \ge 0 \tag{10}$$

Note that the set of constraints (9) ensure that, when solving the optimal value θ_l^* of the lth element of θ, the optimal values θ_t^* of all previous elements $t \in \{1,2,\ldots,l-1\}$ are guaranteed. The solution value θ_1^* of the first model gives the worst link load $[\mu]_1$. For the other link loads, the lth worst link value $[\mu]_l$ is given by $\theta_l^* - \theta_{l-1}^*$.

The use of CG to solve the LP relaxation of this sequence of models is straightforward since constraints (1) define a sub-problem (SP) for each commodity $k \in K$ whose solution is a path. For this reason, CG is usually named Path Generation when applied to network routing problems and the columns generated by CG for the SP associated to commodity k represent paths between its end nodes o_k and d_k in graph G. In this case, each SP is a shortest path problem, which can be easily solved through a shortest path algorithm (for a good survey on the many network routing problems solved through CG, please see [19]).

4. Heuristic algorithm

In the network load balancing problem, the restricted search space Ω is defined by the sets of paths P_k that are possible routing paths for each commodity $k \in K$. Therefore, a solution is defined by selecting a path $p \in P_k$ for each $k \in K$ and its value array $[\mu]$ is computed by rearranging in a non-increasing order the load values $\mu_{\{ij\}}$ given by constraints (2). The following subsections address separately how the problem dependent components of the generic method described in Section 2 were defined to reach a heuristic algorithm tailored for the network load balancing problem.

4.1. *Definition of perturbed problems*

Note that, as described in Section 2, a perturbed problem P' is defined based on an incumbent solution I and on the LP relaxation value of P. Consider the value of I given by $[\mu]$. In the load balancing problem, its LP relaxation is not defined by a single scalar value but, instead, by an array of link loads sorted in a non-increasing order. Let us denote this array by $[\eta]$ where $[\eta]_l$ is its lth element.

By definition, $[\eta]$ is a lower bound for the optimal solution of P in the lexicographical sense, *i.e.*, $[\eta]$ is lexicographical smaller than (or equal to) $[\mu]$ for any incumbent I. If they are equal, it means that I is an optimal solution but this is never the case. So, in general, there is an index δ for which $[\eta]_i = [\mu]_i$ for $i < \delta$ and $[\eta]_\delta < [\mu]_\delta$. In this work, we propose the use of two perturbed problem, denoted by P_1 and P_2.

The motivation of P_1 is to generate a perturbed problem aiming to find additional columns that might lower the value $[\mu]_\delta$ since it exhibits a gap to its lower bound $[\mu]_\delta$. To define P_1, we take the incumbent solution I and we sort the links $\{i,j\} \in A$ in a non-increasing order of their load values $\mu_{\{ij\}}$ on I. Then, we compute the set S containing all links $\{i,j\}$ of order δ and higher, whose load values $\mu_{\{ij\}}$ are higher than $[\eta]_\delta$ (remember that δ is first index for which $[\eta]_\delta < [\mu]_\delta$). Finally, we define P_1 by adding to P the following constrains: for each path $p \in P_k$ that is in the incumbent solution I and includes one link $\{i,j\}$ belonging to S, we add a constraint forbidding commodity $k \in K$ to use paths that include link $\{i,j\}$. The motivation of perturbed problem P_1 is to generate, through CG applied to P_1, additional paths that do not use the links forcing the value $[\mu]_\delta$ on the current incumbent solution.

In the case of perturbed problem P_2, instead of trying to lower the value $[\mu]_\delta$ of the current incumbent solution (the positive gap between $[\mu]_\delta$ and $[\eta]_\delta$ does not mean that $[\mu]_\delta$ is not optimal), we assume that $[\mu]_\delta$ is its optimal value and the motivation is to generate additional columns that might lower the values of $[\mu]_i$ for $i > \delta$. In this case, the perturbed problem P_2 consists on solving the LP relaxation of the original problem P assuming that its optimal values are the values of the incumbent $[\mu]_i$ for $i \le \delta$. To do so, we compute the values θ_l^* for $l \le \delta$ using the relations given by (5) and we run the conditional means approach (described in Section 4) starting on the model correspondent to the $(\delta+1)$th value $\theta_{\delta+1}^*$.

4.2. *Definition of criteria*

Recall from Section 2 that we have to define the *Criteria* that determine when each GRASP+PR search run is interrupted to enable the modification of the restricted search space.

The simplest approach for the definition of the *Criteria* is to interrupt the search when there is an improvement on the incumbent solution *I*. This is not a good approach, though, because of two reasons. First, note that the restricted search space can be modified by including new paths. Since the quality of the new paths is usually related with the quality of the incumbent solution (see the discussion on the previous subsection 4.1), then, too short GRASP+PR search runs might not have enough time to improve significantly the incumbent solution causing the inclusion of undesirable paths. Second, the restricted search space can also be modified by excluding existing paths which is based on the local minimum solutions found by the search. Therefore (and like in the previous case), too short GRASP+PR search runs might compute a very small number of local minimum solutions and may result in excluding too many existing paths.

In our implementation, we have considered to let the search run for a number of GRASP+PR iterations, given by a parameter *Iter*, after any improvement of the incumbent solution. In our problem instances, the best performance was obtained when *Iter* is around 30 and this is the value set in all runs of the computational results presented in this paper.

Moreover, since the incumbent value is an array of non-increasing link load values $[\mu]$, and since the first values are more significant to the objective function (they correspond to the worst load values), we have additionally considered that the incumbent solution improves only if its link load array $[\mu]$ improves on its most significant values, *i.e.*, on at least one of the values $[\mu]_i$ with i up to a parameter *Iload*. In our computational results, we have assumed $Iload=8$ for two reasons. First, the link load values of the best solutions below the 8th worst load value are usually much lower than the worst link load value. Second, the number of commodities not routed through the links with the 8 worst load values are, in general, a small percentage of the total number of commodities.

Finally, we have also combined this criterion with the maximum runtime of the overall method given by parameter *MaxTime* (see Section 2).

In summary, one of the *Criteria*, named *C*1, used in our implementation is to run the GRASP+PR search until

(*Condition* 1 **and** *Condition* 2) **or** *Condition* 3 **is true**

where the three conditions are defined by

Condition 1: the incumbent solution has improved in one of the worst *Iload* link loads.
Condition 2: the search is *Iter* iterations without improving any of the worst *Iload* link loads of the incumbent solution.
Condition 3: the overall *runtime* reached *MaxTime*.

For reasons that will be clear when discussing the computational results, we have also used another *Criteria*, named *C*2. In this case, we have a parameter *Index* and the search stops if the value $[\mu]_{Index}$ of the incumbent improves. When using *C*2, each GRASP+PR search is interrupted when

Condition 3 **or** *Condition* 4 **is true**

where the new *Condition 4* is defined by

Condition 4: the incumbent solution has improved its link load value $[\mu]_{Index}$.

4.3. *Definition of GRASP with path relinking*

GRASP is a multi-start local search method first introduced in [1]. At each start, an initial solution is randomly generated (with some greediness) and local search is applied to it to find a local optimum solution. During the multiple starts, the best local optimum solution is saved as the incumbent solution. At the end, the incumbent solution is the solution of the search. To define the GRASP implementation, one needs to define how the initial solutions are generated and how the neighbors of a solution are computed on the local search.

In the network load balancing problem, we compute an initial solution by sorting randomly the set of commodities $k \in K$ and selecting for each commodity (by the previous order) the path $p \in P_k$ that, together with the previous selected paths, gives the best link load array $[\mu]$ in the lexicographical sense. In each local search step, we use a best improvement strategy, *i.e.*, we compute all neighbor solutions and move to the one that exhibits the best improvement in the objective function. The set of neighbors of a given solution is the set of all solutions which are different from the current one on a single path. Therefore, the total number of neighbor solutions is given by

$$\sum_{k \in K} \left(\left| P_k \right| - 1 \right)$$

Note that, in the generation of the initial solutions, the original GRASP strategy [1] states that each path should be randomly selected among a list of candidate paths that provide the lowest incremental objective function penalty. Some preliminary tests have shown that in the network load balancing problem, the original strategy does not provide additional efficiency and the randomness provided by the different commodity random orders is enough to diversify the search over the search space.

Path relinking (PR), originally proposed as an intensification method applied to tabu search in [2], is a method that tries to find better solutions by the combination of two initial solutions. In the network load balancing problem, we select one initial solution as the starting solution and the other as the target solution and compute the commodities $k \in K$ whose paths are different between the starting solution and the target solution. Then, PR is the iterative process where, at each iteration, we select a commodity to switch the path of the starting solution by the one of the target solution. The selected commodity is the one that gives the best link load array $[\mu]$ in the lexicographical sense. The number of iterations is equal to the number of commodities whose paths are different between the starting and the target solutions. During the process, we save the best solution as the result of PR.

We have used an additional feature in our implementation of PR. At the end, if the best solution is different from the initial solutions, we apply local search to find a local minimum solution. Our computational results show that this additional feature improves the efficiency of the resulting heuristic algorithms.

The common combination of GRASP with PR is to run PR at the end of each GRASP iteration between the local minimum solution given by the local search and one solution randomly selected from a list of previously found elite solutions. The list of elite solutions starts empty and is updated with the best solutions up to a maximum number (given by the maximum elite list size, which is a parameter of the algorithm) and provided that they are not too similar.

Before defining the best GRASP+PR settings for our problem, we have conducted some preliminary efficiency tests which showed that:

(i) The use of an elite list does not provide significant improvements when PR is applied only to one randomly selected elite solution; in this case, it is enough to apply PR between the GRASP local minimum solution and the current incumbent solution (which is equivalent to consider that the elite list size is 1).

(ii) There is a significant improvement if PR is applied between the GRASP local minimum solution and each solution from the set of elite solutions only if the list size is not too large and the elite solutions are not too similar.

(iii) Forward and backward PR, *i.e.*, PR applied both from the GRASP local minimum solution to an elite solution and in the opposite direction, is the best strategy in terms of heuristic efficiency.

Given these preliminary results, we have considered two parameters: *MaxList* is the maximum size of the elite list and *MinDiff* is the minimum percentage of commodities whose paths must be different between two solutions in order to let both solutions be in the elite list.

Then, our final implementation of GRASP+PR is as follows. At any time, the solutions included in the elite list are sorted by their link load arrays $[\mu]$ from the best to the worst in the lexicographical sense. At the end of each GRASP iteration:

1. We apply both forward and backward PR between the local minimum solution and each solution from the set of elite solutions. These PR operations generate a set of new solutions (one for each currently included elite solution).
2. We sort these new solutions by their link load arrays $[\mu]$ from the best to the worst in the lexicographical sense.
3. According to the previous order, for each new solution, we do the following operations:
   - 3.1 if there is a better solution in the current sorted elite list and the new solution has a percentage number of different paths lower than *MinDiff*, the new solution is discarded;
   - 3.2 otherwise, we insert the new solution in the appropriate position of the current sorted elite list and we eliminate all solutions that are worse than the inserted one (in the lexicographical sense) and that have a percentage number of different paths lower than *MinDiff*.
4. We update the elite list with the best *MaxList* solutions.
5. We update the incumbent solution with the first solution of the elite list.

Note that in the overall method, as described in Section 2, the GRASP+PR is run several times. Although not detailed in the algorithms description of Section 2, we also use the elite list as a memory mechanism between different GRASP+PR runs by initializing the elite list of each run with the last elite list of the previous run. This can be safely done since the solutions contained in the elite list are always local minimum solutions and, therefore, cannot contain paths that are excluded from the restricted search space of the next run.

5. Computational results

This section is divided in the following subsections. Section 5.1 presents the test instances used in the computational results. Section 5.2 presents the preliminary analysis of the test instances together with the methodology used for efficiency comparison among the different algorithm options. Section 5.3 analyzes the efficiency improvement obtained by GRASP+PR when an elite list is used. Section 5.4 presents the computational results together with their analysis for the different alternative designs of our proposed heuristic method. Finally, Section 5.5 presents additional computational tests showing that the modification of the restricted search space, through CG, during the search is effective in obtaining better solutions in shorter running times.

5.1. Test instances

All algorithms were developed in C++ programming language and have used the CPLEX 12.1 software package to solve the restricted master problem of CG. In order to test the different algorithms of the proposed heuristic, we have defined a set of 24 test instances based on the well-known network topology of the NSF network with 26 nodes and 42 links.

In all test instances, we have considered that most of the links have 1 Gbps ($=10^6$ Kbps) of capacity but some of them have a capacity of 10 Gbps ($=10^7$ Kbps). Fig. 1 highlights (with lines in bold) the higher capacity links for the first 12 test instances while Fig. 2 highlights the higher capacity links for the second 12 test instances.

In all test instances, we have randomly generated a set of demand commodities with the aim of emulating different possible real scenarios. For each test instance, we have used the following methodology. We start by selecting a node set S with $|S|$ nodes (when $|S|=26$, the set S includes all network nodes and when $|S|<26$, the set S is randomly generated with the same probability of all nodes being included). Then, we randomly select a subset $W\subset S$ with $|W|$ nodes. We assign a parameter of 3 to the nodes in W and a parameter of 1 to the nodes in $S\backslash W$. Finally, for each pair of nodes $\{i,j\}$ in S ($i\in S$ and $j\in S\backslash\{i\}$), we assume that there is a commodity k between the two nodes (*i.e.*, $o_k=i$ and $d_k=j$) and we generate an integer value b_k, uniformly distributed between a minimum value v_m and a maximum value v_M, and multiply it by the parameters of i and j to obtain the demand value (in hundreds of Kbps) of commodity k.

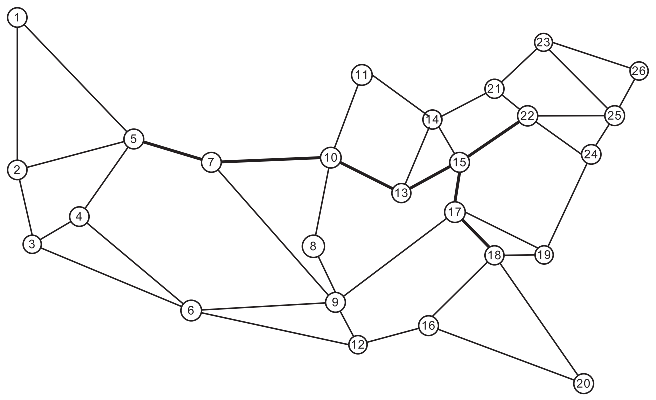

Fig. 1. Network topology for the first 12 test instances.

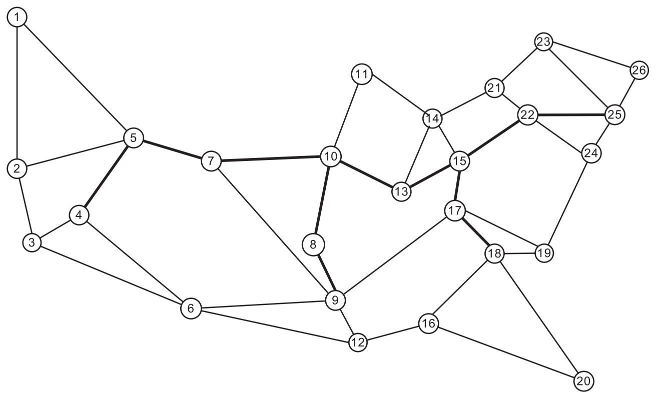

Fig. 2. Network topology for the second 12 test instances.

Table 1
Set of parameters used to generate each test instance.

Instance	v_m	v_M	|S|	|W|
NSF′00	1	300	26	0
NSF′01	1	300	26	0
NSF′02	1	600	26	0
NSF′03	201	400	26	0
NSF′04	1	600	20	0
NSF′05	1	600	20	0
NSF′06	201	400	20	0
NSF′07	201	400	20	0
NSF′08	1	300	26	6
NSF′09	1	300	26	6
NSF′10	1	300	20	6
NSF′11	1	300	20	6
NSF″00	1	400	26	0
NSF″01	1	400	26	0
NSF″02	1	600	26	0
NSF″03	201	400	26	0
NSF″04	1	800	20	0
NSF″05	1	800	20	0
NSF″06	301	500	20	0
NSF″07	301	500	20	0
NSF″08	1	300	26	6
NSF″09	1	300	26	6
NSF″10	1	300	20	6
NSF″11	1	300	20	6

Table 2
Characterization of each test instance.

Instance	*G/B*	*H/L*	No. of paths
NSF′00	*B*	*H*	1245
NSF′01	*B*	*H*	1043
NSF′02	*B*	*H*	1221
NSF′03	*G*	*H*	1240
NSF′04	*B*	*L*	792
NSF′05	*G*	*L*	750
NSF′06	*B*	*L*	899
NSF′07	*G*	*L*	772
NSF′08	*G*	*H*	1318
NSF′09	*B*	*H*	1245
NSF′10	*B*	*L*	743
NSF′11	*B*	*L*	681
NSF″00	*G*	*H*	904
NSF″01	*B*	*H*	938
NSF″02	*B*	*H*	1068
NSF″03	*G*	*H*	1129
NSF″04	*B*	*L*	633
NSF″05	*G*	*L*	598
NSF″06	*G*	*L*	641
NSF″07	*G*	*L*	657
NSF″08	*G*	*H*	1128
NSF″09	*B*	*H*	1071
NSF″10	*B*	*L*	688
NSF″11	*B*	*L*	659

Note that when we consider $|S| < 26$, we are assuming that some network nodes act as transit nodes and, therefore, there are no commodities going to/coming from them. Moreover, when we consider $|W| > 0$, we are assuming that these nodes serve a larger number of clients and, therefore, the corresponding commodity demands are higher, on average, which is accounted for with the multiplication of the random values by 3 (the parameter value assigned to these nodes).

The set of parameters used on each test instance is presented in Table 1 (NSF′ instances are based on the network of Fig. 1 and NSF″ instances are based on the network of Fig. 2).

5.2. Preliminary analysis and methodology

Before presenting and discussing the computational results of the different algorithms in the next subsections, we start by making a first characterization of the different test instances.

First, we have computed the number of paths generated by CG solving the LP relaxation of the network load balancing problem, which defines the initial restricted search space. The column "No. of paths" of Table 2 shows the number of paths generated for each test instance. We also classify each test instance in terms of complexity in High (H), when $|S|=26$ with a total number of $26 \times 25/2 = 325$ commodities, and Low (L), when $|S|=20$ with a total number of $20 \times 19/2 = 190$ commodities (the column "*H/L*" of Table 2 shows the classification of each test instance concerning the instance complexity). An expected relationship which is easily observed in Table 2 is that the number of paths generated by CG is much lower for Low test instances than for High test instances. Nevertheless, dividing for each test instance the number of paths by the number of commodities, the number of paths per commodity (which is around 4, on average) is not too large and this value is close to the average for all test instances.

Then, we have solved through mathematical programming (using CPLEX 12.1) the integer problem with the sets P_k given by the initial restricted search space, *i.e.*, by the set of paths generated by CG solving the LP relaxation of the integer problem; this is the Integer Restricted Master Problem (IRMP). This is still a hard problem to be solved. Therefore, we run each test instance limiting the conditional means algorithm (see Section 3) to solve up to the 8th worst link load and to a maximum runtime of 6 h. Then, with such solutions, we have classified each instance in Good (G) if at least the 3 worst link loads of the IRMP optimal solution are equal to their LP relaxation values or Bad (B) otherwise. Note that when a test instance is classified as Good, it means that its initial restricted search space contains solutions that are optimal at least for the 3 worst link loads. On the other hand, when a test instance is classified as Bad, it means that we do not know if the initial restricted search space contains good quality solutions. Therefore, the strategy of modifying the restricted search space during the GRASP+PR search might be potentially more efficient on the second type of test instances. Column "*G/B*" of Table 2 shows this classification for each test instance (a curious fact observed in all test instances is that the number of worst link loads equal to their LP relaxation values was never 1 or 2; it was always either zero or higher than 2).

Since the proposed method is a stochastic process, it gives different solutions on different runs. In order to generate useful data for comparison analysis in the follows subsections, we have adopted the following methodology. Whenever we want to compare two algorithms, we run both algorithms 10 times for each test instance and compare the pair of solutions of each run. Then, we sum the number of times the solution of the second algorithm is better (in the lexicographical sense) than the solution of the first algorithm. Finally, we average these numbers over the set of test instances of interest. In this way, a result higher than 50% shows that the second algorithm is better than the first algorithm while a result lower than 50% shows that the second algorithm is worse than the first algorithm.

5.3. GRASP+PR computational results

The aim of this subsection is to present computational results showing that the use on an elite list in PR effectively improves the GRASP+PR search efficiency. In the computational results of this subsection, we have maintained the search space constant during the whole run. We have run this algorithm with two different PR settings. In the first setting, we apply PR with parameter *MaxList*$=1$ (in this case, the parameter *MinDiff* is meaningless), which corresponds to the case when PR is applied between the local minimum solution and the incumbent solution (*i.e.*, no elite list is used). In the second setting, we apply PR with parameters *MaxList*=20 and *MinDiff*=30% (see Section 4.3 for the meaning of these parameters). In all cases, we have run the algorithms with a *MaxTime* of 60 s.

Table 3 gives the average results for different instance sets: "Global" means that the results are averaged over all test instances and the other lines mean that the results are averaged over all test instances with the referred classification.

These results show that the use of an elite list improves the efficiency of the algorithm since, globally, 58.1% of the runs produce better results. Note that the use of the elite list improves the performance of all types of instance sets, although the performance improvement is higher in the Bad test instances, when compared with the Good test instances, and in the test instances involving a higher number of commodities (High test instances).

During all runs, we have also computed the average occupation (*i.e.*, average number of solutions) kept on the elite list in the second setting. Although we have used a maximum elite list size of 20, the average occupation of the elite list was between 4 and 5 in all cases, which shows that the *MaxList*=20 did not limit the number of solutions accepted in the elite list. This algorithm behavior is also observed in the runs using the restricted search space modifications (analyzed in the next subsections). Since we are requiring that the solutions included in the elite list are quite different (*MinDiff*=30%), most of the solutions inserted into the elite list eliminate other worse solutions from the list because they are too close to the inserted ones. Therefore, the average occupation of the list is not too high and the GRASP+PR search is not penalized (it does not take too much time applying PR to too many solutions from the elite list).

We have run the algorithm with the values of *MaxList*=5, 10 and 15 (maintaining the value of *MinDiff*=30%) and the algorithm showed a similar average performance with *MaxList* ≥10 and a worse average performance with *MaxList*=5. So, the algorithm performance is not sensitive to this parameter provided that it is large enough. We have also run the algorithm with the values of *MinDiff*=10%, 20% and 40% (maintaining the value of *MaxList*=20). The value of *MinDiff*=30% provided the best performance showing that the algorithm is very sensitive to this parameter. For smaller values of *MinDiff*, the average occupation of the elite list becomes larger penalizing the performance of the algorithm due to the time penalty of applying PR to too many (and too similar) elite solutions. On the other end, for larger values of *MinDiff*, the average occupation of the elite list becomes too small making the algorithm performance similar to the one with no elite list (first setting of Table 3).

5.4. Heuristic computational results

The aim of this subsection is to assess the efficiency of the proposed heuristics when each of the two perturbed problems, P_1 and P_2 (described in Section 4) is used and in the two possible alternatives: either with or without exclusion of columns. Since the efficiency performance was always better when using the elite list (as illustrated in the previous subsection), in this subsection we present the results of all algorithms running with the PR parameter settings of *MaxList*=20 and *MinDiff*=30%.

Like in the previous subsection, all algorithms were run with a *MaxTime* of 60 s. Nevertheless, in the algorithm variants with search space modification, we account for the *runtime* the time spent only in the GRASP+PR search steps (we ignore the time spent in the CG steps), *i.e.*, the maximum time given to the sum of all GRASP+PR steps is 60 s. In this subsection, the aim is to assess the impact on the search efficiency of modifying the restricted search space without taking into account the time penalty required to run the additional CG steps.

First, let us address the efficiency performance of the algorithms with the perturbed problem P_1 (see Section 4.1) both without exclusion of paths (Algorithm 1 of Section 2) and with exclusion of paths (Algorithm 2 of Section 2). In both cases, we have used *Criteria* $C1$. Table 4 shows for each instance set, the average number of times that the restricted search space was modified, together with the total number of paths added and excluded by all modifications averaged by ten runs. Note that the global number of added paths does not represent a dramatic increase on the total number of paths of the restricted search space (when comparing these values with the values of the initial restricted search space presented in Table 2). Moreover, the number of excluded paths (in the case with exclusion of paths) is in the same order of magnitude of the number of added paths which means that in this case, the total number of paths of the different restricted search spaces does not vary significantly.

Now, Table 5 shows the efficiency performance of both algorithms when compared with running GRASP+PR with no restricted search space modifications. These results show a clear improvement of the efficiency of the algorithms using P_1 and a slightly better performance when exclusion of columns is used (63.2% against 62.3%). Note that although the algorithms efficiency improves for all instance sets, the algorithm performs better in the Good test instances, when compared with the Bad test instances, and in the test instances involving a lower number of commodities (Low test instances).

Table 4
Characterization of algorithms with perturbed problem P_1.

Instance sets	P_1 without exclusion of paths		P_1 with exclusion of paths		
	Modifications	Added paths	Modifications	Added paths	Excluded paths
Global	7.2	171.3	7.1	185.2	160.5
B	6.3	171.3	6.4	183.7	158.2
G	8.6	171.1	8.0	187.4	163.7
H	6.7	255.3	6.2	272.5	200.7
L	7.8	87.3	7.9	98.0	120.2

Table 3
GRASP+PR efficiency when the restricted search space is not modified.

Instance sets	PR with incumbent (%)	PR with elite list (%)
Global	41.9	58.1
B	38.8	61.2
G	47.0	53.0
H	39.0	61.0
L	45.8	54.2

Table 5
Efficiency performance of algorithms with perturbed problem P_1.

Instance sets	P_1 without exclusion of paths (%)	P_1 with exclusion of paths (%)
Global	62.3	63.2
B	61.1	61.3
G	63.6	65.2
H	59.1	61.1
L	65.5	65.3

Let us now address the efficiency performance of the algorithms with the perturbed problem P_2, once again both without exclusion of columns (Algorithm 1) and with exclusion of columns (Algorithm 2). Like in the previous P_1 cases, we have used *Criteria* $C1$ in both cases. Table 6 shows for each instance set, the average number times that the restricted search space was modified, together with the total number of paths added and excluded by all modifications averaged by ten runs. When compared with the P_1 case, the number of modifications is similar but there is a huge difference in terms of the number of added and excluded paths: the algorithms with P_2 add much less paths and exclude much more paths. This means that in this case, the size of the restricted search spaces is almost constant throughout the algorithms execution, when there is no exclusion of paths, and is significantly reduced throughout the algorithms execution, when there is exclusion of paths.

Now, Table 7 shows the efficiency performance of both algorithms when compared with running GRASP+PR with no restricted search space modifications. These results show no average improvement (50%) of the efficiency of the algorithms without exclusion of paths and a small global improvement with exclusion of paths (54.5%). However, with exclusion of paths, the algorithm based on P_2 performs better (56.2%) in the Bad test instances, when compared with the Good test instances (52.1%). This is particularly relevant due to the following two reasons. First, this result is obtained by an algorithm that reduces significantly the restricted search space throughout its execution (as observed before). Second, this algorithm performs better in the cases that the algorithms based on P_1 perform worse.

Table 6
Characterization of algorithms with perturbed problem P_2.

Instance sets	P_2 without exclusion of paths		P_2 with exclusion of paths		
	Modifications	Added paths	Modifications	Added paths	Excluded paths
Global	6.4	10.8	7.4	11.5	391.2
B	5.2	9.9	6.1	10.4	379.6
G	8.0	12.0	9.2	13.2	407.5
H	5.6	12.5	6.9	13.7	480.9
L	7.1	9.0	7.9	9.4	301.6

Table 7
Efficiency performance of algorithms with perturbed problem P_2.

Instance sets	P_2 without exclusion of paths (%)	P_2 with exclusion of paths (%)
Global	50.0	54.5
B	47.2	56.2
G	55.1	52.1
H	48.2	45.7
L	52.8	63.3

Up to now, and analyzing all results together, we can conclude that the algorithm based on P_1 with exclusion of paths is the best algorithm. Nevertheless, remember from Section 4.1 that when a perturbed problem is generated based on an incumbent I with value given by $[\mu]$ and on a lower bound given by $[\eta]$, there is always an index δ for which $[\eta]_i = [\mu]_i$ for $i < \delta$ and $[\eta]_\delta < [\mu]_\delta$. Remember also that P_1 aims to generate additional paths to lower the value of $[\mu]_\delta$ (since it exhibits a gap to its lower bound $[\mu]_\delta$) and P_2 assumes that $[\mu]_\delta$ is optimal and aims to generate additional paths to lower the values of $[\mu]_i$ for $i > \delta$.

So, a potentially more efficient approach is to combine P_1 and P_2 in the same algorithm where P_1 is used while $[\mu]_\delta$ is improving; P_2 is used one time if $[\mu]_\delta$ does not improve; and, then, no perturbed problem is used if $[\mu]_\delta$ never improves again. Since P_1 has better performance on Good test instances and P_2 has better performance on Bad test instances, the aim is to obtain a heuristic algorithm that combines the good features of both perturbed problems.

The resulting algorithm is as follows (see Algorithms 1 and 2 descriptions in Section 2):

```
Step 0: the first GRASPwithPR procedure runs with Criteria = C1;
        we set State = TRUE;
Step 1: if State is TRUE, the Perturbation procedure uses
        P_1 to generate the perturbed problem and we set
        Criteria = C1 for the next GRASPwithRP procedure; if
        State is FALSE, the Perturbation procedure uses P_2 to
        generate the perturbed model and we set Criteria = C2
        with Index = δ for the next GRASPwithRP procedure;
Step 2: if [μ]_δ has improved in the GRASPWithPath procedure,
        we set State = TRUE; otherwise, we set State = FALSE;
        we go to Step 1.
```

Table 8 shows for each instance set, both the characteristics and the efficiency performance of these algorithms when compared with running GRASP+PR with no restricted search space modifications. The algorithms that combine P_1 and P_2: (i) show a significant decrease in the number of times the restricted search space is modified; (ii) have a number of added paths similar to the case when P_1 is used, and (iii) have a number of excluded paths significantly lower than the case when P_1 is used.

Concerning the efficiency performance, these algorithms are far better than the best of the previous ones. Globally, the algorithm with exclusion of paths exhibits an efficiency performance of 66.2% against the previous best value of 63.2% of the algorithm based on P_1 with exclusion of paths. Note that this algorithm performs better in the Bad test instances, when compared with the Good test instances, and in the test instances involving a higher number of commodities (High test instances).

5.5. *Final computational results*

In the runs on the previous subsection, we have also computed the runtime spent by the algorithms on each CG step (although not accounted for in the *runtime* to limit the running time to the

Table 8
Characterization and efficiency performance of the algorithms combining perturbed problems P_1 and P_2.

Instance Sets	$P_{1/2}$ without exclusion of paths			$P_{1/2}$ with exclusion of paths			
	Modific.	Added paths	Improv. (%)	Modific.	Added paths	Excluded paths	Improv. (%)
Global	4.9	175.2	61.1	5.0	176.8	94.5	66.2
B	4.4	168.2	61.3	4.4	170.2	90.3	68.2
G	5.6	184.9	60.9	5.8	186.0	100.3	63.1
H	4.8	252.0	65.2	4.8	251.6	121.1	68.4
L	5.2	98.4	56.8	5.1	102.0	67.9	64.0

MaxTime considered). We have observed that each CG step (which is run on each restricted search space modification) takes an average time between 3 and 4 s for the L(ow) complexity instances and between 6 and 7 s for the H(igh) complexity instances. Even in the best algorithm where the average number of modifications was 5.0 (see Table 8), these times represent a running time penalty (not accounted for in the previous subsection) which is significant in the considered *MaxTime* of 60 s.

In this section, the aim is to determine how effective the modification of the restricted search space is during the search when compared with maintaining the restricted search space constant, taking into account the time penalty of the additional CG steps and giving more time to the algorithms.

We have first run GRASP+PR applied to the restricted search space given by initial CG and maintaining this search space constant during the whole run. Then, we have run the best heuristic algorithm of the previous subsection, which is combining the perturbed problems P_1 and P_2 and using the exclusion of columns. For both algorithms, we have now considered a *MaxTime* of 300 s and for the second algorithm the CG step running times were also included in the *runtime* computation. Table 9 shows for each instance set, both the characteristics and the efficiency performance of the best heuristic algorithm when compared with running GRASP+PR with no restricted search space modifications.

Comparing these results with the ones of Table 8, we can conclude that the algorithm efficiency improves even further with larger running times for all types of test instances. Globally, the algorithm efficiency has improved from 66.2% (see Table 8) to 70.9% when we run the algorithms for 300 s.

Note that, as pointed out in subsection 5.2, when a test instance is classified as Bad, it means that we do not know if the initial restricted search space contains good quality solutions. The proposed algorithm has its best efficiency performance in the B(ad) test instances (75.0%) showing that the strategy of modifying the restricted search space during the GRASP+PR search makes the algorithm more efficient on this type of instances. Moreover, the proposed heuristic algorithm also exhibits better performance in H(igh) test instances, which shows that the strategy of modifying the restricted search space during the GRASP+PR search is more efficient for larger search spaces making it a valid method for solving large scale problem instances.

In the next tables (Table 10 for the NSF′ test instances and Table 11 for the NSF″ test instances), we present for each test instance the 8 worst link load values of the LP lower bound of each test instance ("LP" lines) and the 8 worst link load values of the best out of ten runs of each algorithm (A is the algorithm without restricted search space modification and B is the algorithm with restricted search space modification). For each algorithm, we present also in these tables the time instant when the best

Table 9
Characterization and efficiency performance of the algorithm combining P_1 and P_2 with exclusion of columns.

Instance sets	No. of modifications	Added paths	Excluded paths	Improvement (%)
Global	6.6	196.1	117.3	70.9
B	5.4	185.7	108.8	75.0
G	8.4	210.7	129.2	64.8
H	6.9	274.1	148.8	73.4
L	6.4	118.1	85.8	68.4

Table 10
LP values and best results for the NSF′ test instances.

Instance	Case	1st (%)	2nd (%)	3rd (%)	4th (%)	5th (%)	6th (%)	7th (%)	8th (%)	Time
NSF′00	LP	31.71	31.71	31.71	31.71	31.71	31.70	31.70	31.70	–
	A	**31.75**	31.75	31.74	31.73	31.72	31.68	31.67	31.67	137.8
	B	**31.74**	31.72	31.70	31.70	31.70	31.69	31.69	31.69	26.0
NSF′01	LP	32.91	32.91	32.91	32.91	32.87	32.87	32.87	32.86	–
	A	**32.95**	32.95	32.93	32.93	32.92	32.92	32.92	32.88	208.4
	B	**32.92**	32.91	32.91	32.90	32.88	32.88	32.87	32.87	68.1
NSF′02	LP	68.20	68.20	68.20	68.19	68.19	68.19	68.19	67.19	–
	A	**68.26**	68.23	68.21	68.21	68.20	68.17	68.11	67.23	4.0
	B	**68.21**	68.21	68.21	68.19	68.19	68.19	68.16	67.27	20.0
NSF′03	LP	70.02	70.02	70.02	64.23	64.23	64.23	64.22	64.22	–
	A	70.02	70.02	70.02	**64.65**	64.62	64.62	64.56	64.54	145.4
	B	70.02	70.02	70.02	**64.63**	64.47	64.46	64.36	64.30	20.0
NSF′04	LP	37.93	37.93	37.93	37.93	37.93	37.93	37.88	37.51	–
	A	**37.95**	37.95	37.93	37.93	37.91	37.91	37.90	37.51	73.7
	B	**37.94**	37.93	37.93	37.93	37.93	37.92	37.89	37.52	110.5
NSF′05	LP	47.36	47.35	47.35	43.59	43.59	43.04	43.03	43.03	–
	A	47.36	47.35	47.35	**43.61**	43.57	43.26	43.23	43.17	223.1
	B	47.36	47.35	47.35	**43.60**	43.58	43.21	43.18	43.09	63.4
NSF′06	LP	39.23	39.23	39.22	39.22	38.24	38.24	38.24	38.05	–
	A	39.37	39.37	39.36	**39.23**	38.94	38.15	38.07	38.01	286.0
	B	39.37	39.37	39.36	**39.09**	39.07	38.58	38.44	38.41	181.7
NSF′07	LP	49.50	49.50	49.50	44.31	44.31	40.34	40.34	38.38	–
	A	49.50	49.50	49.50	44.31	44.31	40.34	40.34	**39.15**	139.9
	B	49.50	49.50	49.50	44.31	44.31	40.34	40.34	**38.47**	99.2
NSF′08	LP	84.26	84.26	84.25	64.24	64.23	63.46	63.46	63.41	–
	A	84.26	84.26	84.25	**68.94**	68.92	68.80	68.78	67.40	24.1
	B	84.26	84.26	84.25	**64.63**	64.51	64.47	64.44	64.24	49.4
NSF′09	LP	92.33	92.32	92.32	92.32	89.13	89.13	65.13	65.12	–
	A	103.41	103.41	103.38	95.82	95.82	**91.07**	91.07	91.03	233.4
	B	103.41	103.41	103.38	95.82	95.82	**85.86**	85.86	85.85	36.6
NSF′10	LP	78.89	78.88	55.42	55.42	55.42	44.34	44.34	40.54	–
	A	78.90	78.87	**55.43**	55.42	55.41	44.35	44.34	41.58	229.5
	B	78.90	78.87	**55.42**	55.42	55.42	44.34	44.34	43.71	161.3
NSF′11	LP	66.47	66.46	64.19	64.19	64.19	64.19	64.18	62.76	–
	A	66.48	66.45	**64.59**	64.59	63.99	63.93	63.84	63.57	285.2
	B	66.48	66.45	**64.23**	64.23	64.23	64.20	64.05	62.82	136.7

solutions were found. The tables highlight (in bold) the first link value that makes the solution of one algorithm better (in the lexicographical sense) than the solution of the other.

Note that, when comparing the solution values of both algorithm alternatives with the LP lower bound values in Tables 10 and 11, in general, both solutions are of good quality since they are close to the lower bound values. The only exception is NSF′09 test instance whose worst link load lower bound is 92.33% (see Table 10) while both solutions exhibit 3 link loads above 100% (so far, we are still not able to compute a feasible solution to this test instance and, therefore, we do not know if such solution exists).

Comparing the solution values between the two algorithm alternatives, these results show that, except for the NSF″07 test instance, the best solution found by our algorithm is always better than the one found without restricted search space modifications in all other 23 test instances (in the NSF″07 case, the solutions are equal in the 8 worst link load values). When the first algorithm provides already a solution with load values matching the link load lower bounds, our algorithm can still improve some load values on the higher order links. When the first algorithm provides solutions with some gap to the link load lower bound values, our algorithm can usually (but not always) improve the worst link load values.

Concerning the running time to compute the best solutions, our algorithm exhibits an average time of 87.6 s (out of 300 s) among all test instances while the algorithm without restricted search space modifications exhibits an average time of 175.9 s (out of 300 s), which is roughly twice the first value.

In conclusion, the proposed heuristic method is capable of significantly improving the performance of GRASP with path relinking by efficiently managing the restricted search space with column generation based on the combination of both proposed perturbed problems P_1 and P_2 and with exclusion of columns. The computational results show that the improved performance is both in terms of the quality of the solutions and in terms of the average time to compute them.

6. Conclusions

In this paper, we have proposed a general heuristic method which combines the traditional GRASP with path relinking method with Column Generation. This method can be used to solve any problem provided that its LP relaxation can be solved through Column Generation. The key idea of the method is to run a GRASP with path relinking search on a restricted search space defined by column generation instead of running on the complete search space of the problem. Moreover, column generation is used not only to compute the initial restricted search space but also to modify it during the whole algorithm.

We have applied this general heuristic method to derive a heuristic algorithm to solve the network load balancing problem. Its application was based on two different proposed perturbed problems. We first assessed the merits of each perturbed problem in generating additional columns that could make the search more efficient. Finally, we proposed the combination of both perturbed problems in a single heuristic algorithm that has successfully improved the efficiency of the GRASP with path relinking search in test instances of varying characteristics.

Note that, for large problem instances with huge search spaces, it is crucial to constraint the search space in order to make the

Table 11
LP values and best results for the NSF″ test instances.

Instance	Case	1st (%)	2nd (%)	3rd (%)	4th (%)	5th (%)	6th (%)	7th (%)	8th (%)	Time
NSF″00	LP	37.04	37.04	37.04	37.04	37.04	37.04	36.76	36.76	–
	A	**37.06**	37.06	37.04	37.03	37.03	37.02	36.79	36.77	134.0
	B	**37.05**	37.04	37.04	37.04	37.04	37.03	36.84	36.77	23.5
NSF″01	LP	38.33	38.33	38.33	38.32	38.29	38.29	38.29	38.28	–
	A	**38.35**	38.34	38.34	38.32	38.30	38.30	38.28	38.23	220.9
	B	**38.34**	38.33	38.32	38.32	38.32	38.32	38.29	38.22	76.4
NSF″02	LP	53.47	53.47	53.47	53.47	53.47	53.47	48.92	48.92	–
	A	**53.49**	53.48	53.47	53.47	53.46	53.45	53.17	51.22	88.8
	B	**53.48**	53.48	53.47	53.47	53.47	53.45	48.94	48.94	71.3
NSF″03	LP	54.11	54.11	54.11	54.11	54.11	54.11	53.12	53.12	–
	A	**54.13**	54.13	54.12	54.11	54.09	54.08	53.17	53.15	201.9
	B	**54.12**	54.11	54.11	54.11	54.11	54.10	53.31	53.12	149.3
NSF″04	LP	55.65	55.65	55.65	55.65	45.78	40.01	40.00	38.26	–
	A	**55.66**	55.65	55.65	55.64	55.29	46.50	45.44	45.39	275.3
	B	**55.65**	55.65	55.65	55.65	45.78	44.60	44.49	44.48	285.1
NSF″05	LP	52.42	52.42	52.41	42.41	42.40	42.41	42.40	42.40	–
	A	52.42	52.42	52.41	**42.49**	42.41	42.41	42.39	42.39	18.8
	B	52.42	52.42	52.41	**42.48**	42.41	42.41	42.38	42.33	90.7
NSF″06	LP	52.90	52.90	52.90	52.89	48.02	48.02	48.02	45.58	–
	A	52.90	52.90	52.90	52.89	**48.03**	48.02	48.01	45.58	153.7
	B	52.90	52.90	52.90	52.89	**48.02**	48.02	48.02	45.58	13.7
NSF″07	LP	50.18	50.18	50.18	50.17	49.65	49.64	49.65	49.64	–
	A	50.18	50.18	50.18	50.17	49.65	49.65	49.64	49.64	268.4
	B	50.18	50.18	50.18	50.17	49.65	49.65	49.64	49.64	56.2
NSF″08	LP	67.77	67.77	67.76	67.76	66.84	66.83	66.83	57.24	–
	A	67.77	67.77	67.76	67.76	66.96	**66.90**	66.63	63.65	226.3
	B	67.77	67.77	67.76	67.76	66.96	**66.81**	66.72	63.61	16.7
NSF″09	LP	76.37	76.37	76.37	76.36	66.33	66.33	66.33	58.96	–
	A	76.38	76.37	**76.37**	76.36	76.05	74.73	74.72	74.71	295.9
	B	76.38	76.37	**76.36**	76.36	70.02	69.94	69.92	69.91	241.0
NSF″10	LP	74.18	74.17	70.50	70.50	66.93	66.93	66.92	66.48	–
	A	74.19	74.16	70.50	70.50	66.93	**66.93**	66.92	66.72	261.8
	B	74.19	74.16	70.50	70.50	66.93	**66.92**	66.92	66.51	95.3
NSF″11	LP	43.82	43.82	43.81	43.81	43.81	43.81	38.10	38.10	–
	A	**43.84**	43.83	43.82	43.80	43.80	43.79	43.31	41.18	85.6
	B	**43.83**	43.82	43.81	43.81	43.81	43.80	41.83	41.83	11.3

search efficient. Running the search on a restricted space aims to improve the search efficiency but the efficiency of the search depends on the overall quality of the solutions that belong to the restricted search space. In the case of the network load balancing problem, we have shown that column generation provides an efficient method to manage the search space keeping it reasonably small and improving the quality of its solutions. In general, this is a promising method in problems whose LP relaxation can be solved through column generation and with good LP bounds (*i.e.*, the LP bounds given by the associated decomposition is close to the optimal solution value). This is the case of the network load balancing problem, as was already known from [15].

As a final remark, we have previously dealt with the network load balancing optimization as a traffic engineering objective in routing cases such as multiple spanning trees based routing networks [20] and single path routing with path protection [21]. A future line of research is to evaluate how the general heuristic method proposed here for the simplest case of single path routing can be generalized to these more complex traffic engineering variants.

Acknowledgments

The authors wish to thank the reviewers' comments which let them improve significantly the final version of this paper. This work has been conducted under the project PTDC/EIA-EIA/100645/2008 "SearchCol: Meta-heuristic Search by Column generation" (funded by FCT). Dorabella Santos was funded by Portuguese FCT under post-doc grant SFRH/BPD/41581/2007.

References

[1] Feo T, Resende M. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 1989;8:67–71.
[2] Glover F. Tabu search and adaptive memory programming—advances, applications and challenges. In: Barr RS, Helgason RV, Kennington JL, editors. Interfaces in computer science and operations research. Kluwer; 1996. p. 1–75.
[3] Resende M, Ribeiro C. GRASP with path relinking: recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M, editors. Metaheuristics: progress as real problem solvers. Springer; 2005. p. 29–63.
[4] Ford L, Fulkerson D. A suggested computation for maximal multicommodity network flows. Management Science 1958;5:97–101.
[5] Dantzig G, Wolfe P. Decomposition principle for linear programs. Operations Research 1960;8:101–11.
[6] Desaulniers G, Desrosiers J, Solomon M, editors. Column Generation, New York: Springer; 2005.
[7] Lübbecke M, Desrosiers J. Selected topics in column generation. Operations Research 2005;53:1007–23.
[8] Alvelos F, de Sousa A, Santos D. SearchCol: metaheuristic search by column generation. In: Blesa M, Blum C, Raidl G, Roli A, Sampels M, editors, Hybrid metaheuristics. Lecture notes in computer science, vol. 6373; 2010. p. 190–205.
[9] Wang Y, Wang Z. Explicit routing algorithms for internet traffic engineering. In: Proceedings of 8th international Conference on computer communications and networks (ICCCN); 1999. p. 582–588.
[10] Georgiadis L, Georgatsos P, Floros K, Sartzetakis S. Lexicographically optimal balanced networks. IEEE/ACM Transactions on Networking 2002;10(6):818–29.
[11] Nace D, Pióro M. Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial. IEEE Surveys and Tutorials 2008;10(4):5–17.
[12] Gouveia L, Patrício P, de Sousa AF. Lexicographical minimization of routing hops in telecommunication networks. In: Julia Pahl, Torsten Reiners, Stefan Voß, editors, Network optimization. Lecture notes in computer science, vol. 6701; 2011. p. 216–229.
[13] Fortz B, Thorup M. Internet traffic engineering by optimizing OSPF weights. In: Proceedings of 19th IEEE conference on computer communications (INFOCOM); 2000. pp. 519–528.
[14] Fortz B, Thorup M. Optimizing OSPF/IS-IS weights in a changing world. IEEE Journal on Selected Areas in Communications 2002;20(4):756–67.
[15] Santos D, de Sousa A, Alvelos F, Pióro M, Link load balancing optimization of telecommunication networks: A column generation based heuristic approach, International Telecommunications Network Strategy and Planning Symposium (NETWORKS), IEEE Xplore 2010. p. 1–6.
[16] Santos D, de Sousa A, Alvelos F, Pióro M. Optimizing network load balancing: an hybridization approach of metaheuristics with column generation. Telecommunication Systems, published online 2011:1–10, http://dx.doi.org/10.1007/s11235-011-9604-3.
[17] Ogryczak W, Śliwiński T. On solving linear programs with the ordered weighted averaging objective. European Journal of Operational Research 2003;148(1):80–91.
[18] Ogryczak W, Pióro M, Tomaszewski A. Telecommunications network design and max-min optimization problem. Journal of Telecommunications and Information Technology 2005;3:43–56.
[19] Pióro M, Medhi D. Routing, flow and capacity design in communication and computer networks. Morgan Kaufmann; 2004.
[20] Santos D, de Sousa A, Alvelos F, Dzida M, Pióro M. Optimization of link load balancing in multiple spanning tree routing networks. Telecommunication Systems 2010;48(1-2):109–24.
[21] de Sousa A, Santos D, Matos P, Madeira J. Load balancing optimization of capacitated networks with path protection. Electronic Notes in Discrete Mathematics 2010;36:1249–56.