ISSN 0103-9741

Monografias em Ciência da Computação

nº 11/13

Efficient Elementary and Restricted Non-Elementary Route Pricing

Diego Pecin

Marcus Poggi

Rafael Martinelli

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 11/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena July, 2013

Efficient Elementary and Restricted Non-Elementary Route Pricing [*]

Diego Pecin Marcus Poggi
Rafael Martinelli[1]

[1] École Polytechnique de Montréal and GERAD,
3000 chemin de la Côte-Ste-Catherine, Montréal (Québec), Canada H3T 2A7

dpecin@inf.puc-rio.br , poggi@inf.puc-rio.br , rafael.martinelli@gerad.ca

Abstract. Column generation is present in the current most efficient approaches to routing problems. Set partitioning formulations model routing problems by considering all possible routes and selecting a subset of them that visits all customers. These formulations often produce tight linear relaxation lower bounds and require column generation for their pricing step. The lower bounds in the resulting branch-and-price algorithm are tighter when only elementary routes are considered, but this leads to a harder pricing problem. Balancing the pricing problem with route relaxations has become crucial for the efficiency of the branch-and-price approach for routing problems. Recently, Baldacci, Mingozzi and Roberti proposed the ng-routes as a compromise between elementary and non-elementary routes, known as q-routes. The ng-routes are non-elementary routes with the restriction that following a customer it is not allowed to visit another customer that was visited before only if it belongs to a dynamically computed set. This dynamic set is obtained from ng-sets of given size, associated to each customer, which is usually composed by the closest ones. The larger the size of the ng-sets, the closer the ng-route is to an elementary route. This work presents an efficient pricing algorithm for ng-routes and extends this algorithm to pricing elementary routes. Therefore we address the SPPRC and the ESPPRC problems. The proposed algorithm combines Decremental State-Space Relaxation (DSSR) technique with completion bounds. This allows strengthening the domination rule, drastically reducing the total number of labels. We apply this algorithm for the Generalized Vehicle Routing Problem (GVRP). Experimental results are also presented for the Capacitated Vehicle Routing Problem (CVRP). We report for the first time experiments with ng-set sizes up to sixty-four, obtaining several new best lower bounds for the GVRP, specially for large instances. The proposed algorithm is capable to price elementary routes even for CVRP instances with 200 customers, a result which doubles the size of the ESPPRC instances solved so far.

Keywords: Vehicle Routing, Column Generation, Non-Elementary Routes, Elementary Routes.

Resumo. Geração de colunas está atualmente presente nas abordagens mais eficientes para os problemas de roteamento. Formulações de particionamento de conjuntos modelam problemas de roteamento considerando todas as rotas possíveis e selecionando um subconjunto delas que visita todos os clientes. A relaxação linear destas formulações

[*] The contribution by Diego Pecin, Marcus Poggi and Rafael Martinelli has been partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processes numbers 141538 / 2010-4, 309337 / 2009-7 and 140849 / 2008-4. These supports are gratefully acknowledged.

frequentemente produzem bons limites inferiores, especialmente quando apenas rotas elementares são consideradas nos modelos, mas neste caso o subproblema de geração de colunas é de resolução difícil. Assim, balancear a dificuldade do subproblema de geração de colunas com a relaxação das rotas é fundamental para a eficiência dos algortimos de branch-and-price para problemas de roteamento. Recentemente, Baldacci, Mingozzi and Roberti propuseram as ng-rotas como um compromisso entre rotas elementares e rotas não-elementares, conhecidas como q-rotas. As ng-rotas são rotas não-elementares na qual um cliente não pode ser revisitado (formando um ciclo) se este cliente pertencer a um conjunto calculado dinamicamente. Este conjunto dinâmico é obtido a partir de ng-conjuntos de determinado tamanho, associados a cada cliente, e são geralmente compostos pelos clientes mais próximos. Quanto maior for o tamanho dos ng-conjuntos, mais próximo a ng-rota é de uma rota elementar. Este trabalho apresenta um algoritmo de pricing eficiente para ng-rotas e estende este algoritmo para pricing de rotas elementares. Portanto, aborda-se os problemas SPPRC e ESPPRC. O algoritmo proposto combina a técnica de relaxação decrescente do espaço de estados (*Decremental State-Space Relaxation* – DSSR) com limites de complemento. Isto permite reforçar a regra de dominância, reduzindo drasticamente o número total de rótulos na programação dinâmica. O algoritmo final é aplicado ao Problema de Roteamento de Veículos Generalizado (*Generalized Vehicle Routing Problem* – GVRP). Resultados computacionais são também apresentados para o Problema de Roteamento de Veículos com Restrição de Capacidade. Reporta-se pela primeira vez na literatura experimentos com ng-conjuntos de tamanho até sessenta e quatro, o que permite obter vários novos limites inferiores para o GVRP, especialmente para instâncias grandes. O algoritmo proposto é capaz de obter rotas elementares para instâncias do CVRP com 200 clientes, um resultado que dobra o tamanho das instâncias do ESPPRC resolvidas atomento.

Palavras-chave: Roteamento de Veículos, Geração de Colunas, Rotas Não-Elementares, Rotas Elementares.

Contents

1 Introduction

Since the work of Christofides et al. [1] on the Capacitated Vehicle Routing Problem (CVRP), column generation has become a technique widely applied for solving different routing problems exactly. Nowadays it is present in almost all of the current most efficient approaches to routing problems. These approaches use integer and mixed integer programming formulations with variables associated to the set of all possible routes. These formulations are set partitioning based for these constraints impose the selection of the route to serve each customer. We refer to it as SPP formulation. The resolution of its linear relaxation requires the use of column generation techniques. The pricing subproblem to be solved is the Elementary Shortest Path Problem with Resource Constraints (ESPPRC).

The ESPPRC is a shortest path problem on a graph where the customers have amount of resources which are consumed during its visit. The resource constraints require that the total of resources consumed by any feasible solution does not exceed the existing limits. There may be edges with negative cost and these edges may generate negative cycles, but since a feasible solution must be an elementary path, revisiting a customer is strictly forbidden.

The ESPPRC is a hard to solve $\mathcal{NP}$-Hard problem [2]. In general, the current best performing algorithms have acceptable processing times when the optimal solution is a path with at most fourteen customers. We refer to Pugliese and Guerriero [3] for a review of the approaches proposed along the last three decades. Most of the main ideas in its resolution can be found in their work and in Feillet et al. [4], Chabrier [5], Righini and Salani [6,7] and Boland et al. [8].

Instead of solving the ESPPRC, the original work of Christofides et al. [1] solves its relaxation, the Shortest Path Problem with Resource Constraints (SPPRC). This relaxation does allow revisiting a same customer in a route. The resulting non-elementary routes are often called q-routes. However, the resource constraints are the same as the ESPPRC and therefore, every time a customer is visited, its resources consumption are accounted and the total consumption must still respect the existing limits.

This relaxation has some interesting properties. First, differently from the original problem, this one can be solved in pseudo-polynomial time using a dynamic programming algorithm. Besides, even relaxing the elementarity constraint, the SPP bounds found by its linear relaxation are usually strong, especially when there are few customers per route. Furthermore, in order to strengthen the bounds of the linear relaxation, Christofides et al. [1] also showed that cycles of size two can be forbidden almost with no extra effort.

However, even with the 2-cycle elimination restriction, the linear relaxation is still weak, a behavior which motivated researchers to look for better cycle elimination devices to turn the non-elementary routes closer to elementary routes, without dealing with the whole complexity of the ESPPRC. The work of Irnich and Villeneuve [9] devised an algorithm which solves the SPPRC forbidding cycles of an arbitrary size. This algorithm is significantly more complicated, resulting in a complexity which grows factorially with the size of the cycles being forbidden. On account of this, their method quickly becomes impractical. Eliminating cycles of size four or more is already too time consuming compared to the bound improvement obtained. Such behavior was verified in practice by Fukasawa et al. [10] for the CVRP.

Recently Baldacci et al. [11] proposed a compromise between routes and q-routes: the *ng*-routes. The *ng*-routes are restricted non-elementary routes built accordingly to customer sets, the *ng*-sets, which are associated to each customer and act like their "memory". Therefore, when a path reaches a given customer, it "forgets" if another customer was visited if it does not belong to the *ng*-set of the current customer. Moreover, the further extension is only allowed to customers which are not "remembered". The *ng*-sets are usually composed by the closest customers and clearly the larger the size of these sets, the harder is to solve the problem. This is due to the fact that the *ng*-routes generated are going to be increasingly closer to elementary routes.

Although the SPPRC with *ng*-routes can be solved in pseudo-polynomial time for a fixed *ng*-set size, as far as we know, there is no work which solves this problem for *ng*-sets larger than 20. This is due to the exponential-sized data structure used by Baldacci et al. [11] in order to speed up the algorithm.

1.1 Contributions of this Paper

This work aims at efficiently solving the SPPRC with restricted non-elementary routes and the ESPPRC. The first improvement is obtained by adapting Righini and Salani's [7] Decremental State-Space Relaxation (DSSR) technique to the SPPRC with *ng*-routes. This technique was firstly proposed for the ESPPRC, where the elementarity restriction is relaxed and the problem is then solved iteratively, rebuilding the restrictions as needed, until the optimal solution is found. The main difference of our algorithm is instead of relaxing the elementarity of the routes, we relax the restriction imposed by the *ng*-sets.

Next, we accelerate this approach using completion bounds. Since an iteration of the DSSR is a relaxation for its next iteration, the completion bounds estimate lower bounds for completing the paths being built on a given iteration and, given an upper bound for the optimal solution (which is usually equal to zero for pricing algorithms), it avoids the extension of paths which may exceed the known upper bound on the next DSSR iteration.

These two techniques were already used together in the work of Pecin [12]. In this work, a column generation procedure which uses the ESPPRC as the pricing subproblem was proposed for the CVRP. Using the algorithm of Fukasawa et al. [10], instances with up to 100 customers could be solved to optimality pricing only elementary routes.

Finally, we show how our algorithm for the SPPRC with restricted non-elementary routes can be easily extended to generate only elementary routes. We also highlight the two new elements existing in our approach that allows us to double the size of the ESPPRC instances solved so far.

The proposed algorithms are then applied to the Generalized Vehicle Routing Problem (GVRP), where customers to be visited are clusters of vertices. Each cluster has an associated demand and can contain one or more vertices. The demand of each cluster must be fully collected in exactly one vertex of the cluster. This problem is a generalization of the CVRP and the Traveling Salesman Problem (TSP) and, as in the classical CVRP, identical vehicles are given, routes must start and end at the depot and the capacity of the vehicle must not be exceeded. We report experiments showing that our algorithms are able to solve the SPPRC with *ng*-set sizes up to 64 and the ESPPRC for hard instances of both GVRP and CVRP. The CVRP instances are solved through reducing them to GVRP instances. The results of the column generation algorithm also give a clear idea of the gains in lower bounds comparing the SPPRC with different *ng*-set sizes and also with the ESPPRC, as well as the time required for computing them. In addition, several new best lower bounds are found for the GVRP, especially for large instances.

This paper is organized as follows. Next Section presents the ESPPRC and explains the required mathematical notation. The *ng*-route relaxation is described in Section 3. In Section 4, we explain the techniques used to solve the *ng*-route relaxation. In section 5, we show how our algorithm can be used to obtain only elementary routes and we highlight the main elements that allowed us to build a very efficient method for solving the ESPPRC. Section 6 presents the Generalized Vehicle Routing Problem formally. Section 7 reports computational results for both GVRP and CVRP. Finally, Section 8 presents some conclusions.

2 Elementary Shortest Path Problem with Resource Constraints

Let $G = (V, A)$ be a graph with arc set A and vertex set V, which is composed by the set of customers $\mathcal{C}$ plus a source vertex s and a destination vertex t, and let $\mathcal{R}$ be a set of resources. For each arc $(i, j) \in A$, let c_{ij} be the cost of the arc and w^r_{ij} be the consumption of the edge, for each $r \in \mathcal{R}$. For each pair $i \in \mathcal{C}$ and $r \in \mathcal{R}$ let a^r_i and b^r_i be two non-negative values, such that the total resource consumption along a path from s to i must belong to the interval $[a^r_i, b^r_i]$. The ESPPRC aims to find a minimum cost elementary path from s to t which satisfies all resource constraints.

The resources constraints can model different types of restrictions. For instance, most of vehicle routing problems considers that the vehicles have a known capacity and it cannot be exceeded in a single route. Other problems have time windows, which requires the route to visit a customer in a given interval of time. Moreover, one can also view the elementarity constraint as resource constraints, where each customer defines a binary resource and when a route visits a customer, it consumes all of the associated resource.

In this work, we deal only with the capacity constraint, besides the obvious elementarity constraint. Thus, the customer set has an associated demand function $d : \mathcal{C} \to \mathbb{Z}$ and there is a global capacity limit Q, which no feasible solution may exceed. Since we apply our algorithm for routing problems, we can consider the source and the destination vertices as a single vertex called the depot and labeled 0. Therefore, the solution of the problem is a route instead of a path. This is straightforward because most of the applications for the ESPPRC is a pricing routine embedded in a column generation scheme which solves some kind of vehicle routing problem.

3 The *ng*-Route Relaxation

Recently, the ng-route relaxation was introduced in the work of Baldacci et al. [11] for the CVRP and the CVRP with Time Windows (CVRPTW), and it was later extended to the GVRP by Bartolini et al. [13], where it was used to solve transformed CARP instances. This new relaxation aims at having a better compromise between efficiently price non-elementary routes and obtaining good lower bounds.

For each customer $i \in \mathcal{C}$, let $N_i \subseteq \mathcal{C}$ be a subset of customers which have a relationship with i. A possible representation for this relationship can be a neighborhood relationship, i.e., N_i contains the nearest customers of i, including i. These sets are called *ng*-sets and they contain the customers which customer i is able to "remember". For instance, when a path P is being built, by the time it arrives at customer i, it has a set $\Pi(P)$ which represents its "memory" so far. If customer i belongs to set $\Pi(P)$, the extension is forbidden. On the other hand, if i does not belong to set $\Pi(P)$, the extension is allowed and the set $\Pi(P)$ is updated to "forget" the customers which customer i is not able to "rememeber", i.e., the customers which does not belong to N_i. It is clear that if a customer is "forgotten", it can be used to form a cycle in future extensions of path P. At this point, we can conclude that the size of the ng-sets is an important factor on the quality of solutions, because the larger the ng-sets are, the greater will be the smallest cycles which can appear in a path. The size of each ng-set N_i is limited by $\Delta(N_i)$, which is a parameter defined *a priori*. Obviously, this size also changes the pricing complexity.

Let $P = (0, i_1, \ldots, i_{p-1}, i_p)$ be a path starting at the depot, visiting a sequence of customers and ending at customer i_p, and $\mathcal{C}(P)$ be the set of customers visited by path P. The function $\Pi(P)$ of prohibited extensions (the "memory") of path P can be defined as follows.

$$\Pi(P) = \left\{ i_k \in \mathcal{C}(P)\backslash\{i_p\} : i_k \in \bigcap_{s=k+1}^{p} N_{i_s} \right\} \cup \{i_p\}\,. \tag{1}$$

Given $d(P) = \sum_{i \in \mathcal{C}(P)} d_i$ as the total demand serviced by path P and $c(P)$ as the total cost of path P, let $\mathcal{L}(P) = (i_p, d(P), \Pi(P), c(P))$ be a label associated with a path P, which ends at customer i_p. We say that a label $\mathcal{L}(P)$ can be extended to a customer i_{p+1} if $i_{p+1} \notin \Pi(P)$ and $d(P) + d_{i_{p+1}} \leq Q$. After the extension, the customer i_{p+1} becomes the last customer of a new path $P' = (0, \ldots, i_p, i_{p+1})$ and a new label $\mathcal{L}(P')$ can be obtained from the label $\mathcal{L}(P)$ by the following operations:

$$\mathcal{L}(P') = (i_{p+1}, d(P) + d_{i_{p+1}}, \Pi(P) \cap N_{i_{p+1}} \cup \{i_{p+1}\}, c(P) + c_{i_p i_{p+1}}). \tag{2}$$

These labels are computed using a forward dynamic programming algorithm and, in contrast to the q-route relaxation, it does not result in a pseudo-polynomial complexity. This algorithm is exponential on the size of $\Delta(N_i)$, remaining pseudo-polynomial for fixed $\Delta(N_i)$. Its efficiency depends on the use of some techniques to speed up its execution.

In order to reduce the number of possible paths, a dominance rule is incorporated into the algorithm. Given the labels of two paths $\mathcal{L}(P_1)$ and $\mathcal{L}(P_2)$, path P_1 dominates path P_2 if and only if every possible extension from P_2 can be done from P_1 with a lower or equal total cost. For this to be true, the following three conditions must hold:

(i) $d(P_1) \leq d(P_2)$,

(ii) $c(P_1) \leq c(P_2)$ and

(iii) $\Pi(P_1) \subseteq \Pi(P_2)$.

In addition, Baldacci et al. [11] described another way to improve this dominance rule. When paths with a given capacity d are being computed, the best costs for every $d' < d$ are stored in a dominance list, which is faster to check than to iterate through the dynamic programming matrix for each $d' < d$. We do not use this technique because it is not scalable, since the size of this dominance list is exponential in the value of $\Delta(N_i)$, reaching its size limit when $\Delta(N_i) \approx 13$.

4 An Efficient *ng*-Route Relaxation

As discussed earlier, a basic *ng*-route relaxation implementation does not allow the use of large ng-sets, which weaken the quality of the lower bounds found. To address this issue, we provide an efficient implementation, adapting the Decremental State Space Relaxation (DSSR) for the *ng*-route relaxation. This technique was introduced by Righini and Salani [7] to solve the ESPPRC. The original version of the algorithm helps reducing the number of labels to be managed during the dynamic programming algorithm which builds elementary paths. Firstly, it relaxes the elementarity of the paths and, at each iteration, identifies which customers are being repeated on the best path found and then prohibits the repetition of these customers in subsequent iterations.

The main difference of our algorithm is that instead of relaxing the elementarity of the paths, the new algorithm relaxes the *ng*-set of each customer, therefore relaxing the *ng*-route restrictions.

4.1 Basic Exact Dynamic Programming Algorithm

We start by creating a $(Q+1) \times |\mathcal{C}|$ dynamic programming matrix $\mathcal{M}$, where each entry $\mathcal{M}(d,i)$ is a bucket containing labels representing paths which start at the depot and end at customer i with total demand exactly d. At first, we set $\mathcal{M}(d_i,i)$ with a single label $\mathcal{L}_i = (i, d_i, \{i\}, c_{0i})$, $\forall i \in \mathcal{C}$, and all other entries with no label. Next, a forward dynamic programming is used to fill the matrix $\mathcal{M}$, running from $d = 1$ up to $d = Q$.

Algorithm 1 presents the pseudocode of our basic dynamic programming procedure. When processing the bucket $\mathcal{M}(d,i)$, the algorithm iterates through all labels $\mathcal{L}(P_j)$ belonging to $\mathcal{M}(d-d_i,j)$, for all customers $j \in \mathcal{C}$, such that $d - d_i > 0$. As the basic *ng*-route relaxation algorithm, the extension from $\mathcal{L}(P_j)$ to i can only be done if $i \notin \Pi(P_j)$. If this condition holds, a new label, say $\mathcal{L}(P_i)$, is then created and it must be stored in the bucket $\mathcal{M}(d,i)$. Therefore, this is the right time to check for the dominance rule, which can be verified for all the labels in $\mathcal{M}(d',i), \forall d' \leq d$. This is the place where the dominance list mentioned earlier is used in the work of Baldacci et al. [11]. Surprisingly, we have found that the algorithm runs faster if the dominance rule is tested only for labels of the same bucket, i.e., for the labels from inside $\mathcal{M}(d,i)$. This comes from the fact that labels associated to paths using less capacity are unlikely to dominate others using higher capacity.

Algorithm 1 also presupposes the existence of procedure *buildRoutes*, which take matrix $\mathcal{M}$ as parameter and extracts the best routes from it.

4.2 Improved Exact Dynamic Programming Algorithm

4.2.1 Decremental State-Space Relaxation

The adapted DSSR is an iterative algorithm and it works by relaxing the state space of the original *ng*-sets N_i. At each iteration k, the algorithm uses the subsets $\Gamma_i^k \subseteq N_i$ as a replacement for N_i. These subsets Γ_i^k take the role of N_i in the definition of the function Π, described in (1), and in the creation of new labels, as shown in (2). Initially, the algorithm sets Γ_i^0, $\forall i \in \mathcal{C}$, as an empty set, and executes the basic dynamic programming, Algorithm 1. As the best routes found by this dynamic programming are not necessarily *ng*-routes w.r.t. the original *ng*-sets N_i, they cannot be considered as the result of the pricing without verifying their feasibility. This test is performed and the Γ_i^k subsets are updated if necessary, as described hereafter. If at the end of iteration k some subset Γ_i^k is updated, the dynamic programming algorithm is executed again with new subsets Γ_i^{k+1}.

Let a cycle of customers be defined as a sub-path $H = (i, \ldots, j)$, where $i = j$, and let

Algorithm 1 Basic Dynamic Programming *ng*-Route Algorithm

```
1: procedure DYNAMICPROGRAMMING($\mathcal{M}$, N)
2:     input: matrix $\mathcal{M}$ and ng-sets $N_i \subseteq \mathcal{C}, \forall i \in \mathcal{C}$.
3:     output: the best ng-routes with respect to ng-sets $N_i$.

4:     $\mathcal{M}(d,i) \leftarrow \emptyset, \forall i \in \mathcal{C}, d \in \{0,\dots,Q\}$
5:     $\mathcal{M}(d_i,i) \leftarrow \{(i,d_i,\{i\},c_{0i})\}, \forall i \in \mathcal{C}$
6:     for $d := 1,\dots,Q$ do
7:         for all $i \in \mathcal{C}$ do
8:             if $d - d_i > 0$ then
9:                 for all $j \in \mathcal{C}$ do
10:                    for all $\mathcal{L}(P_j) \in \mathcal{M}(d-d_i,j)$ do
11:                        if $i \notin \Pi(P_j)$ then
12:                            $\mathcal{L}(P_i) \leftarrow (i,d,\Pi(P_j) \cap N_i \cup \{i\}, c(P_j) + c_{ji})$
13:                            $insertLabel \leftarrow$ true
14:                            for all $\mathcal{L}(P_i') \in \mathcal{M}(d,i)$ do
15:                                if $\mathcal{L}(P_i)$ dominates $\mathcal{L}(P_i')$ then
16:                                    delete $\mathcal{L}(P_i')$
17:                                else if $\mathcal{L}(P_i')$ dominates $\mathcal{L}(P_i)$ then
18:                                    $insertLabel \leftarrow$ false
19:                                    break
20:                            if $insertLabel$ then
21:                                $\mathcal{M}(d,i) \leftarrow \mathcal{M}(d,i) \cup \mathcal{L}(P_i)$
22:     return buildRoutes($\mathcal{M}$)
```

$\mathscr{H}(P)$ be the set of all cycles of customers in the path P. In order to evaluate if the best route R_k^* found in the end of iteration k is an *ng*-route, the algorithm must check if there is no cycle $H \in \mathscr{H}(R_k^*)$ which would not be allowed to be created if the original *ng*-sets N_i were being used. This happens only when the customer i (which is repeated in cycle H) is in all N_l, $\forall l \in \mathcal{C}(H)$, i.e., customer i is not "forgotten" by any other customer of the cycle. If any such cycle H is found, we add customer i to subsets Γ_l^{k+1}, $\forall l \in \mathcal{C}(H)$. This prohibits cycle H from appearing in any path obtained in the next iteration. On the other hand, if no such cycle is found at the end of iteration k, then the best route R_k^* is an *ng*-route and the algorithm stops.

Algorithm 2 reports DSSR procedure. The input parameter of the algorithm is the original *ng*-sets N_i, $\forall i \in \mathcal{C}$. The procedures with self-explanatory names *selectBestRoute*, *isNGRoute* and *updateNGSets* are responsible, respectively, to extract the best route of a set of routes, to determine if a given route is a valid *ng*-route with respect to *ng*-sets N_i and to update the subsets Γ_i^k to the next iteration.

Algorithm 2 also uses the procedure *dynamicProgramming*, which is presented in Algorithm 1, in order to obtain *ng*-routes with respect to subsets Γ_i^k passed as input parameters.

It is noteworthy to mention that if the best route found is indeed an *ng*-route, the algorithm can stop and return only this route. However, if the objective is to find a set of feasible solutions, the algorithm can also return this route together with any other route certified to be an *ng*-route. Furthermore, if the best route is not an *ng*-route and one needs to find *any* feasible solution, any route which is indeed an *ng*-route can be returned, even if the best one is not feasible. In this case, we consider it as being a heuristic run of the algorithm, not an exact one. This method is useful to quickly price routes on intermediate iterations of column generation algorithms, where there is no need to generate the optimal solution.

Algorithm 2 Pure DSSR *ng*-Route Algorithm

```
procedure DSSR($\mathcal{M}$, $\mathbf{N}$)
    input: matrix $\mathcal{M}$ and $ng$-sets $N_i \subseteq \mathcal{C}, \forall i \in \mathcal{C}$.
    output: the best $ng$-routes with respect to $ng$-sets $N_i$.

    $\Gamma_i \leftarrow \emptyset, \forall i \in \mathcal{C}$, $ng \leftarrow$ false, $k \leftarrow 0$
    while not $ng$ do
        $\mathcal{R} \leftarrow$ dynamicProgramming($\mathcal{M}$, $\Gamma$)
        $R_k^* \leftarrow$ selectBestRoute($\mathcal{R}$)
        if isNGRoute($R_k^*$) then
            $ng \leftarrow$ true
        else
            updateNGSets($\mathbf{N}$, $R_k^*$)
        $k \leftarrow k + 1$
    return $\mathcal{R}$

procedure isNGRoute($R$)
    for all $H = (v, \dots, v) \in \mathscr{H}(R)$ do
        $forbiddenCycle \leftarrow$ true
        for all $l \in \mathcal{C}(H)$ do
            if $v \notin N_l$ then
                $forbiddenCycle \leftarrow$ false
                break
        if $forbiddenCycle$ then
            return false
    return true

procedure updateNGSets($\mathbf{N}$, $R$)
    for all $H = (v, \dots, v) \in \mathscr{H}(R)$ do
        $forbiddenCycle \leftarrow$ true
        for all $l \in \mathcal{C}(H)$ do
            if $v \notin N_l$ then
                $forbiddenCycle \leftarrow$ false
                break
        if $forbiddenCycle$ then
            for all $l \in \mathcal{C}(H)$ do $\Gamma_l \leftarrow \Gamma_l \cup \{v\}$
```

4.2.2 Completion Bounds

In order to further speed up the algorithm, we calculate completion bounds during the DSSR in a similar manner as done by Pecin [12] for the elementary route. At the end of iteration k, the completion bounds are calculated for each customer $i \in \mathcal{C}$ with every capacity $d \in \{0, \ldots, Q\}$, and then used at iteration $k+1$. As mentioned before, the completion bounds are used to estimate a lower bound on the value of a route during its creation, thus discarding any route which would not lead to a negative reduced cost. Given $T_k^*(d, i)$, the value of the best path which starts at customer i and ends at the depot with total capacity exactly d, the completion bounds $\widehat{T}_k(d, i)$ are calculated as shown in (3) and represent the value of the best path which starts at customer i and ends at the depot with total capacity less than or equal d.

$$\widehat{T}_k(d, i) = \min_{d' \leq d} \left\{ T_k^*(d', i) \right\}. \tag{3}$$

It is important to observe that if the corresponding problem is represented by means of an undirected graph, $T_k^*(d, i)$ can be obtained directly from the dynamic programming matrix. This is true because the value of the best path which starts at the depot and ends at customer i with total capacity exactly d has the same value as $T_k^*(d, i)$, as shown by Baldacci et al. [11]. The reason for this is that if the best forward path contains a cycle, then the best backward path may also contain this cycle and vice versa, because at least one costumer belonging to this cycle must "forget" the customer that repeats. On the other hand, if the problem is represented using a directed graph, in order to obtain these values, the direction of the edges has to be reversed, as also shown by [11]. In this case, the last iteration of the DSSR algorithm has to be executed again before the calculation. This occurs because when a route is traversed in the opposite direction on an asymmetric graph, it does not generate the same cost.

After calculating the completion bounds at the end of iteration k, they can be used at iteration $k+1$ to avoid the extension of a given label $\mathcal{L}(P) = (j, d(P), \Pi(P), c(P))$ to a customer i if the following conditions hold:

$$c(P) + c_{ij} + \widehat{T}_k(Q - d(P), i) \geq 0. \tag{4}$$

This equation calculates a lower bound on the value of the reduced cost of any route the label $\mathcal{L}(P)$ can generate, because $\widehat{T}_k(Q - d(P), i)$ is a lower bound on the value of the best path which would close path P after it be extended to customer i. Obviously, if the value of equation (4) is greater or equal than zero, the label $\mathcal{L}(P)$ cannot generate any route with a negative reduced cost, therefore it can be discarded.

It is interesting to highlight that the completion bounds becomes stronger along the iterations of the DSSR, since iteration k is a relaxation of iteration $k+1$, i.e., since $\Gamma_i^k \subseteq \Gamma_i^{k+1}, \forall i \in \mathcal{C}$. These bounds can be used for other important parts of a branch and cut and price algorithm, such as route enumeration and variable fixing.

Algorithm 3 reports the pseudocode for pricing ng-routes with state space relaxation and completions bounds. Procedures *selectBestRoute*, *isNGRoute* and *updateNGSets* have the same meaning as before and *generateCompletionBounds* is a new one that calculates the bounds as shown in (3). Procedure *BoundedDynamicProgramming* is a slight modification of procedure *dynamicProgramming* of Algorithm 1, targeting only include the completion bounds calculated as shown in (4).

Algorithm 3 DSSR with Completion Bounds ng-Route Algorithm

```
1: procedure DSSRwithBounds($\mathcal{M}$, $\mathbf{N}$)
2:     input: matrix $\mathcal{M}$ and ng-sets $N_i \subseteq \mathcal{C}, \forall i \in \mathcal{C}$.
3:     output: the best ng-routes with respect to ng-sets $N_i$.

4:     $\Gamma_i \leftarrow \emptyset, \forall i \in \mathcal{C}$, $ng \leftarrow$ false, $k \leftarrow 0$
5:     $\widehat{T}(d,i) \leftarrow -\infty, \forall i \in \mathcal{C}, d \in \{0,\dots,Q\}$
6:     while not $ng$ do
7:         $R \leftarrow$ BoundedDynamicProgramming($\mathcal{M}$, $\Gamma$, $\widehat{T}$)
8:         $R_k^* \leftarrow$ selectBestRoute($R$)
9:         if isNGRoute($R_k^*$) then
10:            $ng \leftarrow$ true
11:        else
12:            updateNGSets($\mathbf{N}$, $R_k^*$)
13:            generateCompletionBounds($\mathcal{M}$, $\widehat{T}$)
14:        $k \leftarrow k+1$
15:    return $R$

16: procedure generateCompletionBounds($\mathcal{M}$, $\widehat{T}$)
17:    $\widehat{T}(d,i) \leftarrow \infty, \forall i \in \mathcal{C}, d \in \{0,\dots,Q\}$
18:    $\widehat{T}(0,0) \leftarrow 0$
19:    for $i \in \mathcal{C}$ do
20:        for $d := 1,\dots,Q$ do
21:            $\widehat{T}(d,i) \leftarrow \min(\mathcal{M}(d,i))$
22:            if $\widehat{T}(d-1,i) < \widehat{T}(d,i)$ then
23:                $\widehat{T}(d,i) \leftarrow \widehat{T}(d-1,i)$

24: procedure BoundedDynamicProgramming($\mathcal{M}$, $\Gamma$, $\widehat{T}$)
25:    $\mathcal{M}(d,i) \leftarrow \emptyset, \forall i \in \mathcal{C}, d \in \{0,\dots,Q\}$
26:    $\mathcal{M}(d_i,i) \leftarrow \{(i, d_i, \{i\}, \bar{c}_{0i})\}, \forall i \in \mathcal{C}$
27:    for $d := 1,\dots,Q$ do
28:        for all $i \in \mathcal{C}$ do
29:            if $d - d_i > 0$ then
30:                for all $j \in \mathcal{C}$ do
31:                    for all $\mathcal{L}(P_j) \in \mathcal{M}(d-d_i, j)$ do
32:                        if $i \notin \Pi(P_j)$ then
33:                            if checkCompletionBound($\widehat{T}$, $\mathcal{L}(P_j)$, $i$) then
34:                                $\mathcal{L}(P_i) \leftarrow (i, d, \Pi(P_j) \cap N_i \cup \{i\}, \bar{c}(P_j) + \bar{c}_{ji})$
35:                                $insertLabel \leftarrow$ true
36:                                for all $\mathcal{L}(P_i') \in \mathcal{M}(d,i)$ do
37:                                    if $\mathcal{L}(P_i)$ dominates $\mathcal{L}(P_i')$ then
38:                                        delete $\mathcal{L}(P_i')$
39:                                    else if $\mathcal{L}(P_i')$ dominates $\mathcal{L}(P_i)$ then
40:                                        $insertLabel \leftarrow$ false
41:                                        break
42:                                if $insertLabel$ then
43:                                    $\mathcal{M}(d,i) \leftarrow \mathcal{M}(d,i) \cup \mathcal{L}(P_i)$
44:    return buildRoutes($\mathcal{M}$)
```

4.3 Heuristic Pricing

Even with the improvements described in Section 4.2, the exact ng-route pricing still takes a long time to be executed. In the view of this, a simple but effective heuristic is developed in order to quickly price a large initial set of routes with negative reduced cost. It was based on the heuristic pricing done for the elementary route pricing by Pecin [12]. The purpose of this heuristic is to reduce the number of calls to the exact ng-route pricing. Therefore, the heuristic ng-route pricing is used as a hot-start for the exact ng-route pricing.

The heuristic closely resembles the q-route pricing without eliminating any cycle. The main difference between the pricing algorithms is that when extending one path, the heuristic ng-route pricing respects the ng-sets N_i. Its data structure is also an $(Q+1) \times |\mathcal{C}|$ matrix and each entry consists of just one label. For each customer and each capacity, this label is chosen as the best one with respect to the reduced costs. Also, as the ng-sets N_i must be respected, each label of the dynamic programming matrix must contain the Π sets for each customer and capacity.

Furthermore, since the algorithm described in Section 5 aims to find only elementary routes, we implemented a heuristic pricing exactly like the one described in Pecin [12].

Notice that unlike the exact algorithms, the heuristic algorithms use neither the dominance rules nor the speed up techniques (DSSR and completion bounds) described in Section 4. Nevertheless, they are responsible to obtain about 90% of the routes during the column generation. Moreover, it is straightforward to verify that the resulting complexity of the algorithms is $\mathcal{O}(n^2Q)$.

5 Achieving Elementarity

In order to achieve elementarity we just set the *ng*-sets N_i to include all customers of the instance, for all customer $i \in \mathcal{C}$. Then we apply the same algorithm described in Section 4. This way, the optimal route found in the last iteration of the DSSR algorithm will not contain cycles, and therefore will be an optimal elementary route.

There is a great difference how we increase the state space along the DSSR iterations and the manner done by Righini and Salani [7]. Our algorithm starts with empties $\Gamma_i^0, \forall i \in \mathcal{C}$. At the end of each iteration k, it identifies all cycles on the best solution and the repeated customer of each cycle $H \in \mathscr{H}(R_k^*)$ is inserted on subsets Γ_l^{k+1}, $\forall l \in \mathcal{C}(H)$. Thus, if cycle $H = (v, \ldots, v)$ is identified at the end of iteration k (that is, if cycle H belongs to $\mathscr{H}(R_k^*)$), the next DSSR iterations will not generate any path with a cycle $H^{'} = (v, \ldots, v)$ in which $\mathcal{C}(H^{'}) \subseteq \mathcal{C}(H)$. But note that it is still possible to obtain a path that visits customer v more than once at iteration $k+1$, since this customer is not present in all subsets Γ_i^{k+1}. On the other hand, the DSSR of Righini and Salani [7] is performed prohibiting the customers that repeat on the best route R_k^* from repeating again in subsequent iterations, until an elementary route be found. This is equivalent to insert each repeated customer of each cycle $H \in \mathscr{H}(R_k^*)$ in all subsets Γ_i^{k+1}, rather than just consider this inclusion on subsets Γ_l^{k+1}, $\forall l \in \mathcal{C}(H)$.

We noted on computational experiments that this more aggressive manner of increase the state space is quite dangerous because the whole algorithm fails if the number of labels to be treated in the dynamic programming becomes critical. In contrast, the size of the largest subset Γ_i^k hardly exceed 20 in our algorithm, even for instances with 200 customers.

Another important difference from our approach and the one of Righini and Salani [7] is the way we use the DSSR to calculate completion bounds at each iteration in order to accelerate the subsequent DSSR iterations. Computational results show that the use of completion bounds allow us to solve the column generation for CVRP instances that would not be solved in an acceptable time if they were not used.

6 Application to the Generalized Vehicle Routing Problem

The Generalized Vehicle Routing Problem (GVRP) can be defined as follows. Let $G = (V, E)$ be a graph with vertex set V and edge set E. There is a special vertex 0 called the depot. The vertices are partitioned into disjoint sets, called clusters, $C = \{C_0, C_1, \ldots, C_t\}$, where $C_0 = \{0\}$ contains only the depot. Given the cluster index set $M = \{0, 1, \ldots, t\}$, let $\mu(i) \in M$ be defined, for each vertex $i \in V$, as the index of the cluster which contains i. There exists a demand function $d : M \to \mathbb{Z}^+$ associated with all clusters, in which the depot has demand $d_0 = 0$. These demands are to be serviced by a set $\mathcal{K}$ of identical vehicles with capacity Q, located at the depot. The edge set $E = \{\{i, j\} | i, j \in V, \mu(i) \neq \mu(j)\}$ contains the edges between all pairs of vertices from different clusters. Associated with these edges, there exists a traversal cost function $c : E \to \mathbb{Z}_0^+$. Let $\mathscr{R}$ be the set of all possible closed routes starting and ending at the depot. The objective of the GVRP is to select a subset of k routes from $\mathscr{R}$ which: (i) minimizes the total traversal cost; (ii) the demand from every cluster is serviced by a single vehicle on a single vertex from each cluster; (iii) the total demand serviced by each route does not exceed the vehicle capacity Q.

The GVRP is a generalization of the Capacitated Vehicle Routing Problem (CVRP) and the Generalized Traveling Salesman Problem (GTSP). When all the clusters contain only one vertex, it is simply the CVRP. Similarly, when there is only one vehicle, it is simply the GTSP. It is clear that, when both conditions are true, it is simply the Traveling Salesman Problem (TSP). In the view of this, it is easy to see that any solution given to the GVRP can be directly used to solve these problems. In the sections that follow, any algorithm designed for the GVRP will also be applied to the CVRP in this way.

This problem is strongly $\mathcal{NP}$-hard and has gained attention in the literature in recent years. As far as we know, the first published work to deal with this problem is Ghiani and Improta [14], where a transformation to the Capacitated Arc Routing Problem (CARP) is presented in order to use the existing algorithms for the CARP. One instance was proposed and solved using this approach.

Since then, few works have been published on the GVRP. Recently, the work of Bektaş et al. [15] has proposed four formulations for the GVRP. After extensive experiments with these formulations, a branch-and-cut algorithm was devised using one of them, an undirected formulation with an exponential number of constraints. The reader may consult this paper for further details on these formulations.

7 Experimental Results

For the computational experiments, all algorithms were implemented in C++ using Microsoft Visual C++ 2010 Express and IBM ILOG CPLEX Optimizer 12.5 for solving the formulations. The experiments were conducted on an Intel Core i7-3960X 3.30GHz with 64GB RAM running Linux Ubuntu Server 12.04 LTS and are divided into two parts. First, we compare the three exact pricing algorithms described in this paper, that is, the basic dynamic programming, the pure DSSR and DSSR with completion bounds. This is enough to conclude that the latter is the best one, and this evaluation is done by running each algorithm inside a column generation schema for some classical CVRP instances. All three algorithms were tested using different values of $\Delta(N_i)$, allowing us to analyse the scalability of each one when the state space relaxation is increased. Second, using our best algorithm we price elementary and restricted non-elementary routes for both GVRP and CVRP instances. In order to improve the lower bounds and also show that our algorithm still works well when *robust* cuts (see terminology in [16]) are added in the SPP formulation, we included capacity inequalities and strengthened comb, both described in detail in [17]. We separated and added these cuts in a similar manner as done by Fukasawa et al. [10] for the CVRP.

For all tests, the column generation starts by calling the heuristic pricing at each iteration. The heuristic pricing returns the best 20 routes with negative reduced costs. If the heuristic is no longer capable of finding routes with negative reduced cost, the column generation algorithm calls the exact pricing. If the later succeeds in obtaining at least one route with negative reduced cost, the column generation procedure restarts by calling the heuristic pricing. Otherwise, the column generation stops and the current value is returned as a lower bound. This procedure is stopped prematurely if the time limit of two hours is exceeded.

7.1 Problem Instances

For the GVRP, we applied our algorithms to the instance datasets recently generated by Bektaş et al. [15]. These instance datasets are derived from the CVRP instance datasets `A`, `B`, `P` and `M`. The transformation is performed using a method similar to that of Fischetti et al. [18], which transforms TSP instances into GTSP instances. The number of clusters is $t = \lceil n/\theta \rceil$, where θ is a parameter defined *a priori*. For each original CVRP instance dataset, two new instance datasets were created, using $\theta = 2$ and $\theta = 3$, resulting in 158 GVRP instances. All lower and upper bounds shown in the tables were taken from the work of Bektaş et al. [15].

The name of the GVRP instances follows the general convention of the CVRP instance, although slightly modified in order to include additional parameters used. The general format is X-nY-kZ-CΩ-VΦ, where X corresponds to the type of the instance, Y refers to the number of vertices, Z corresponds to the number of vehicles in the original CVRP instance, Ω is the number of clusters, and Φ is the number of vehicles in the GVRP instance. For all problem instances, we calculate the cost matrix using Euclidean distance rounded to the nearest integer value.

For the CVRP, we used just a representative set extracted from the classical instance datasets `A`, `B`, `E`, `P` and `M`, available at `www.branchandcut.org` [19]. All optimal values shown in the tables was extracted from [10], except the optimal value for instance M-n151-k12, which was first proved by Contardo [20] and the optimal values for instances M-n200-k17 and M-n200-k16, which were first proved by the recent (not yet published) algorithm proposed by Uchoa et al. [21].

7.2 Performance evaluation

This section evaluates the performance of our three ng-route pricing algorithms: the simple dynamic programming, the pure state space relaxation and finally the one that combines the techniques of state space relaxation with completion bounds. These algorithms were tested inside a column generation procedure on CVRP instances. The results are shown in Table 1. Columns `Ins` and `OPT` show the name and the optimum value of each instance. Following these columns, the results for different values of $\Delta(N_i)$ are shown. For each X, where $\Delta(N_i) = X$, `NG=X` consists of four columns, `LB`, `T1`, `T2` and `T3`, which show the lower bound and the total time required to compute it for, respectively, Algorithm 1, Algorithm 2 and Algorithm 3.

The results from Table 1 show that for small ng-set sizes (`NG=8`) the best approach comes from the simple Algorithm 1. This is not a surprise because the maximum number of non-dominated labels per bucket is limited to 2^X, and therefore the total number of labels handled by the algorithm does not explode. In this case, a unique running of the dynamic programming considering the entire state space is in general best than running it in the relaxed state space several times. For an average ng-set size (`NG=16`), the pure DSSR still does not improve the times of Algorithm 1, but the combination of DSSR and completions bounds gives a great improvement, allowing to run all instances in a very reasonable time (except for instances M-n135-k7 and M-n121-k7). For large ng-set sizes (`NG`$\geq$`32`), the pure DSSR significantly outperforms the basic dynamic programming algorithm, but is still a poor algorithm for most instances. However, the DSSR with completions bound drastically improves the times.

We can also note that the bounds for `NG=32` is almost as good as the elementary for most instances. This is an evidence that the routes found by the pricing with large ng-sets are almost elementary when the average size of the routes is up to 12 or 13 costumers, in a time tipically less than the time required if the elementary constraint is imposed to the routes. But this is not necessarily a rule. It is noteworthy to mention that the greater is the ng-set size used, the better is the completion bounds obtained. In some cases, the improved completion bounds resulting from the use of a large ng-set may compensate the additional complexity imposed. This is evidenced in the time for running the instance M-n121-k7, that is significatively greater for `NG=32` than `NG=64`.

On the other hand, our algorithms spend a lot of time trying to solve the column generation for instances M-n135-k7 and M-n121-k7, especially for large ng-set sizes. This is mainly due to the average size of the routes that are part of an optimum solution for these instances, which is greater than 17, causing the number of labels to be treated by the dynamic programming algorithm prohibitive. In particular, instance M-n135-k7 has a car capacity of 2210, a value at least 10 times greater than the capacity of any other instance considered in the tests.

7.3 Other results

Now we show other results obtained with our best pricing algorithm, Algorithm 3. Table 2 presents the results of the column generation for the CVRP considering the separation of capacity and strengthened comb cuts. As done previously, the results are shown for different values of $\Delta(N_i)$. Table 3 shows the results for the elementary routes for the CVRP. The column named `Elem` shows the bounds and times of the column generation without cuts and colunm `Elem+Cuts` presents the results with cuts. Finally, Table 4 shows a summary of the results for the GVRP instances, all with separation of capacity and strengthened comb cuts. Columns `Set`, `Num` and θ show the name of the sets, the number of instances in each set and the θ used in the transformation, respectively. Columns `NG=X` show the average gap and average time of the sets for the column generation with the ng-route pricing for $\Delta(N_i) = X$ (similarly, the results for the elementary routes are presented for each set). These gaps are obtained comparing our values with the upper bounds available in the work of Bektaş et al. [15]. We select some instances where our algorithms performed better than those described by Bektaş et al. [15]. The results for these instances are shown in Table 5. The column `Ins` shows the name of each instance, columns `LB` and `UB` shows the results from [15]. Values in bold are those in which we found the optimal solution with our column generation algorithm.

8 Conclusions

The strength of the ng-route pricing is the ability of adjusting the size of the ng-sets in order to calculate a lower bound, in a reasonable time, which is close as much as possible to the elementary route bound, and this justifies an efficient implementation of the method. In this paper we have presented an efficient ng-route pricing algorithm for ng-set sizes up to sixty-four, a number at least three times greater than we know so far. Furthermore, we showed how our restricted non-elementary route pricing algorithm can be easily extended in order to price only elementary routes. We highlighted the two elements that allowed us to price elementary routes even for CVRP instances with 200 customers, a result which doubled the size of the ESPPRC instances solved so far. The first element is the way we adapt the Decremental State Space Relaxation (DSSR) technique of Righini and Salani [7] for the ng-routes context, thus improving their way of increasing the state space along the DSSR iterations. The second is the combination of the DSSR technique with completion bounds, which are calculated in each iteration of the DSSR for the purpose of accelerating the next iteration.

The final algorithm was tested for pricing elementary and restricted non-elementary routes to a set partitioning formulation for both GVRP and CVRP. For the first one, we could improve the lower bounds for up to 13 instances, since we also separated and added capacity and strengthened comb cuts.

References

[1] CHRISTOFIDES, N.; MINGOZZI, A. ; TOTH, P.. **Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations**. Mathematical Programming, 20:255–282, 1981.

[2] DROR, M.. **Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW**. Operations Research, 42(5):977–978, 1994.

[3] PUGLIESE, L. D. P.; GUERRIERO, F.. **A computational study of solution approaches for the resource constrained elementary shortest path problem.** Annals OR, 201(1):131–157, 2012.

[4] FEILLET, D.; DEJAX, P.; GENDREAU, M. ; GUEGUEN, C.. **An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to some Vehicle Routing Problems**. Networks, 44(3):216–229, 2004.

[5] CHABRIER, A.. **Vehicle routing problem with elementary shortest path based column generation**. Comput. Oper. Res., 33(10):2972–2990, Oct. 2006.

[6] RIGHINI, G.; SALANI, M.. **Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints**. Discret. Optim., 3(3):255–273, Sept. 2006.

Table 1: Results for selected CVRP instances

		NG=8				NG=16				NG=32				NG=64			
Ins	OPT	LB	T1	T2	T3	LB	T1	T2	T3	LB	T1	T2	T3	LB	T1	T2	T3
A-n62-k8	1288	1250.24	1.03	2.04	1.06	1254.83	4.86	4.77	1.20	1254.83	4177.10	27.82	1.50	1254.83	—	1888	1.89
A-n63-k10	1314	1286.58	0.56	1.79	0.56	1286.81	3.25	8.92	0.86	1286.83	180.98	142.41	0.88	1286.83	—	—	0.99
A-n64-k9	1401	1368.24	0.88	1.75	0.88	1374.49	3.64	5.42	1.27	1376.90	177.98	48.47	1.06	1376.90	—	1653.9	1.63
A-n69-k9	1159	1129.97	1.08	2.10	0.99	1131.33	6.32	8.18	1.12	1131.34	2002.60	361.66	1.19	1131.34	—	—	1.81
A-n80-k10	1763	1729.81	1.77	3.64	1.76	1731.45	5.94	8.42	1.97	1731.58	720.96	36.20	2.25	1731.58	—	6768.7	3.68
B-n50-k8	1312	1266.45	0.57	0.91	0.45	1266.63	4.36	2.58	0.51	1266.64	1521.4	8.97	0.60	1266.64	—	411.25	0.66
B-n68-k9	1275	1198.20	1.06	2.50	0.76	1203.78	36.68	21.20	1.56	1204.00	—	606.00	1.55	1204.00	—	—	2.22
B-n78-k10	1221	1166.73	1.70	4.66	1.57	1167.46	47.20	19.39	1.77	1167.52	—	281.12	1.82	1167.52	—	1564.7	2.92
E-n51-k5	521	517.14	0.70	1.27	0.58	517.14	5.04	4.41	0.76	517.14	1988.10	100.58	0.83	517.14	—	—	0.89
E-n76-k7	682	664.20	2.90	5.70	2.62	664.82	10.25	19.22	2.80	665.59	—	494.64	4.10	665.59	—	—	4.67
E-n76-k8	735	717.82	1.85	3.98	1.68	718.74	10.61	12.25	2.20	718.78	2795.30	239.64	2.55	718.78	—	—	2.78
E-n76-k10	830	811.77	1.16	2.55	1.07	811.87	2.39	4.28	1.04	812.48	715.69	26.60	1.32	812.48	—	142.4	1.51
E-n76-k14	1021	1001.85	0.57	1.63	0.59	1002.77	0.88	2.37	0.56	1002.77	16.83	3.86	0.60	1002.77	—	7.21	0.77
E-n101-k8	815	789.37	7.33	14.31	6.93	790.71	32.91	40.80	7.97	790.99	—	2462	9.14	790.99	—	—	10.99
E-n101-k14	1067	1047.20	1.93	4.86	1.91	1048.45	7.15	9.06	2.46	1050.42	726.02	35.01	2.50	1050.42	—	381.71	2.89
F-n135-k7	1162	1134.79	2055.2	7298.6	3818.5	1135.9	—	—	3925.1	1136.73	—	—	7289.8	—	—	—	—
M-n121-k7	1034	1026.37	27.00	69.16	28.34	1029.21	—	—	382.15	1029.75	—	—	7403.2	1029.79	—	—	1815.00
M-n151-k12	1015	995.73	19.88	41.87	22.11	996.64	68.14	100.37	26.45	997.28	—	1076.9	31.31	997.43	—	—	37.58
M-n200-k16	1274	1250.23	38.51	102.65	61.32	1250.87	134.38	255.91	61.27	1251.36	—	2449.20	62.95	1252.05	—	—	99.47
M-n200-k17	1275	1252.52	38.96	94.25	53.85	1252.87	129.51	193.66	58.34	1253.43	—	2950.6	67.14	1254.01	—	—	87.89
P-n50-k8	631	614.64	0.24	0.62	0.24	615.55	0.52	1.41	0.26	615.55	24.65	4.15	0.24	615.55	—	17.08	0.30
P-n70-k10	827	809.29	0.57	1.63	0.55	810.61	2.40	4.09	0.76	810.94	163.22	11.90	0.76	810.94	—	31.72	1.01

Table 2: Results for selected CVRP instances

		NG=8		NG=16		NG=32		NG=64	
Ins	OPT	LB	Time	LB	Time	LB	Time	LB	Time
A-n62-k8	1288	1280.75	2.37	1281.99	2.85	1282.08	2.78	1282.09	3.33
A-n63-k10	1314	1302.42	1.00	1303.85	1.10	1303.89	1.27	1303.97	1.74
A-n64-k9	1401	1387.8	1.72	1389.4	1.50	1389.97	1.73	1390.02	2.294
A-n69-k9	1159	1143.8	1.35	1145.22	1.36	1145.38	1.89	1145.42	2.48
A-n80-k10	1763	1756.46	3.43	1756.97	3.94	1756.9	3.96	1756.8	5.48
B-n50-k8	1312	1303.48	0.75	1303.62	0.90	1303.76	0.99	1303.77	1.017
B-n68-k9	1275	1263.75	2.91	1263.92	3.90	1263.92	3.30	1263.93	4.14
B-n78-k10	1221	1217.06	3.77	1217.38	3.10	1217.32	4.16	1217.62	5.73
E-n51-k5	521	519.26	1.12	519.45	1.27	519.52	1.44	519.49	1.55
E-n76-k7	682	671.10	3.92	671.32	4.49	671.94	6.04	671.93	6.21
E-n76-k8	735	726.98	4.01	726.99	3.45	727.18	4.02	727.37	4.41
E-n76-k10	830	817.34	1.71	817.42	1.83	818.01	1.88	818.01	2.35
E-n76-k14	1021	1006.94	0.75	1007.5	0.75	1007.5	0.75	1007.54	0.94
E-n101-k8	815	804.51	16.23	805.00	17.05	805.08	17.66	805.14	21.35
E-n101-k14	1067	1053.70	2.70	1054.09	2.89	1055.01	3.14	1055.01	3.27
F-n135-k7	1162	1160.29	7209.5	1160.57	6854.70	1160.67	7289.60	—	—
M-n121-k7	1034	1032.48	45.898	1032.57	565.14	1032.85	2873.2	—	—
M-n151-k12	1015	1000.39	30.82	1001.50	33.94	1002.20	38.98	1002.42	49.07
M-n200-k16	1274	1252.66	61.46	1253.04	67.76	1253.31	81.40	1254.16	91.43
M-n200-k17	1275	1255.27	68.16	1255.22	64.34	1255.63	70.67	1256.21	91.33
P-n50-k8	631	617.16	0.35	618.02	0.37	618.02	0.38	618.04	0.42
P-n70-k10	827	814.33	1.03	815.29	1.16	815.38	1.30	815.39	1.57

Table 3: Results for selected CVRP instances

		Elem		Elem+Cuts	
Ins	OPT	LB	Time	LB	Time
A-n62-k8	1288	1254.83	1.48	1282.09	2.92
A-n63-k10	1314	1286.83	0.88	1303.87	1.33
A-n64-k9	1401	1376.9	1.34	1390.12	1.85
A-n69-k9	1159	1131.34	1.65	1145.24	1.85
A-n80-k10	1763	1731.58	3.34	1756.8	5.06
B-n50-k8	1312	1266.64	0.54	1303.74	0.74
B-n68-k9	1275	1204	1.56	1263.93	3.47
B-n78-k10	1221	1167.52	2.27	1217.25	4.97
E-n51-k5	521	517.14	0.71	519.48	1.28
E-n76-k7	682	665.58	4.68	671.94	5.92
E-n76-k8	735	718.77	2.44	727.19	4.29
E-n76-k10	830	812.47	1.26	818.01	2.08
E-n76-k14	1021	1002.77	0.62	1007.5	0.80
E-n101-k8	815	790.98	11.01	805.11	22.79
E-n101-k14	1067	1050.42	3.67	1055.01	4.25
F-n135-k7	1162	—	—	—	—
M-n121-k7	1034	1029.79	1301.8	1033.23	7225.7
M-n151-k12	1015	997.43	52.04	1002.58	57.52
M-n200-k16	1274	1252.05	140.92	1254.16	149.73
M-n200-k17	1275	1254.01	140.89	1255.94	147.55
P-n50-k8	631	615.54	0.23	618.04	0.34
P-n70-k10	827	810.94	0.84	815.29	1.24

Table 4: Summary of results for GVRP instances

			NG=8		NG=16		NG=32		NG=64		Elem	
Set	Num	θ	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
A	27	2	0.48%	0.29	0.40%	0.29	0.40%	0.26	0.40%	0.26	0.40%	0.22
B	23	2	0.15%	0.34	0.13%	0.34	0.12%	0.31	0.13%	0.32	0.13%	0.28
M	4	2	1.16%	18.11	1.12%	21.09	1.09%	27.18	1.08%	34.38	1.09%	24.40
P	24	2	0.43%	2.35	0.38%	3.48	0.38%	4.92	0.38%	92.79	0.38%	4.54
A	27	3	0.37%	0.19	0.25%	0.18	0.24%	0.15	0.24%	0.16	0.24%	0.13
B	23	3	0.23%	0.21	0.22%	0.18	0.22%	0.18	0.22%	0.20	0.22%	0.16
M	4	3	0.65%	10.39	0.53%	14.76	0.50%	23.42	0.50%	18.00	0.50%	15.31
P	24	3	0.44%	1.21	0.40%	3.14	0.40%	3.95	0.40%	12.11	0.40%	4.64

Table 5: Results for selected GVRP instances

	Best Bounds		NG=8		NG=16		NG=32		NG=64		Elem	
Ins	LB	UB	LB	Time	LB	Time	LB	Time	LB	Time	LB	Time
A-n63-k9-C21-V3	625.6	642	629.7	0.25	636.3	0.22	636.3	0.20	636.3	0.22	636.3	0.15
A-n80-k10-C27-V4	679.4	710	706.5	0.55	708.8	0.51	708.8	0.49	708.8	0.53	708.8	0.39
A-n80-k10-C40-V5	957.4	997	982.5	1.13	982.7	1.22	982.8	1.34	983.4	1.11	983.4	0.91
M-n121-k7-C61-V4	707.7	719	710.4	26.09	710.7	37.84	710.9	58.22	710.9	83.58	710.9	40.73
M-n151-k12-C51-V4	465.6	483	482.4	10.44	**483.0**	11.32	**483.0**	13.21	**483.0**	11.25	**483.0**	12.69
M-n151-k12-C76-V6	629.9	659	649.3	13.21	649.6	16.44	649.9	13.43	650.2	14.74	650.0	13.84
M-n200-k16-C67-V6	563.1	605	592.5	15.02	593.6	15.57	594.2	19.24	594.2	19.11	594.2	13.86
M-n200-k16-C100-V8	744.9	791	777.7	29.07	778.4	25.78	778.7	32.05	778.8	33.64	778.8	32.51
P-n50-k8-C25-V4	378.4	392	385.7	0.14	385.8	0.22	385.8	0.17	385.8	0.18	385.8	0.17
P-n55-k15-C28-V8	545.3	555	**555.0**	0.06	**555.0**	0.06	**555.0**	0.05	**555.0**	0.06	**555.0**	0.04
P-n60-k10-C30-V5	433	443	435.6	0.29	435.4	0.32	435.3	0.27	435.3	0.29	435.3	0.21
P-n60-k15-C20-V5	380	382	380.3	0.09	381.5	0.08	381.5	0.10	381.5	0.10	381.5	0.07
P-n60-k15-C30-V8	553.9	565	564.7	0.13	564.8	0.10	564.8	0.09	564.8	0.09	564.8	0.07

[7] RIGHINI, G.; SALANI, M.. **New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem**. Networks, 51(3):155–170, 2008.

[8] BOLAND, N.; DETHRIDGE, J. ; DUMITRESCU, I.. **Accelerated label setting algorithms for the elementary resource constrained shortest path problem**. Operations Research Letters, 34(1):58–68, 2006.

[9] IRNICH, S.; VILLENEUVE, D.. **The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for k $\geq$ 3**. INFORMS Journal on Computing, 18(3):391–406, 2006.

[10] FUKASAWA, R.; LONGO, H.; LYSGAARD, J.; POGGI DE ARAGÃO, M.; REIS, M.; UCHOA, E. ; WERNECK, R. F.. **Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem**. Mathematical Programming, 106(3):491–511, 2006.

[11] BALDACCI, R.; MINGOZZI, A. ; ROBERTI, R.. **New route relaxation and pricing strategies for the vehicle routing problem**. Operations Research, 59(5):1269–1283, 2011.

[12] PECIN, D. G.. **Uso de rotas elementares no CVRP**. Master's thesis, Instituto de Informática, Universidade Federal de Goiás, 2010.

[13] BARTOLINI, E.; CORDEAU, J.-F. ; LAPORTE, G.. **Improved lower bounds and exact algorithm for the capacitated arc routing problem**. Technical Report CIRRELT-2011-33, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation, 2011.

[14] GHIANI, G.; IMPROTA, G.. **An efficient transformation of the generalized vehicle routing problem**. European Journal of Operational Research, 122(1):11–17, 2000.

[15] BEKTAŞ, T.; ERDOĞAN, G. ; RØPKE, S.. **Formulations and branch-and-cut algorithms for the generalized vehicle routing problem**. Transportation Science, 45(3):299–316, 2011.

[16] DE ARAGAO, M. P.; UCHOA, E.. **Integer program reformulation for robust branch-and-cut-and-price algorithms**. In: IN PROCEEDINGS OF THE CONFERENCE MATHEMATICAL PROGRAM IN RIO: A CONFERENCE IN HONOUR OF NELSON MACULAN, p. 56–61, 2003.

[17] LYSGAARD, J.; LETCHFORD, A. N. ; EGLESE, R. W.. **A new branch-and-cut algorithm for the capacitated vehicle routing problem**. Mathematical Programming, 100(2):423–445, June 2004.

[18] FISCHETTI, M.; SALAZAR-GONZÁLEZ, J. J. ; TOTH, P.. **A branch-and-cut algorithm for the symmetric generalized traveling salesman problem**. Operations Research, 45(3):378–394, 1997.

[19] www.branchandcut.org. Visited on March, 2012.

[20] CONTARDO, C.. **A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints**. Technical report, Archipel-UQAM 5078, Universite du Quebec a Montreal, Canada, 2012.

[21] UCHOA, E.; PECIN, D.; PESSOA, A. ; POGGI, M.. **New Exact Approaches for the Capacitated Vehicle Routing Problem**. In: EUROPEAN CONFERENCE ON OPERATIONS RESEARCH XXVI, 2012.