

A POPMUSIC matheuristic for the capacitated vehicle routing problem

Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa

▶ **To cite this version:**

Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa. A POPMUSIC matheuristic for the capacitated vehicle routing problem. Computers and Operations Research, 2021, 136 (105475), 10.1016/j.cor.2021.105475 . hal-02994210

HAL Id: hal-02994210
https://inria.hal.science/hal-02994210

Submitted on 7 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A modern POPMUSIC matheuristic for the capacitated vehicle routing problem

Eduardo Queiroga*[1], Ruslan Sadykov†[2], and Eduardo Uchoa‡[3]

[1]Instituto de Computação, Universidade Federal Fluminense, Av. Gal. Milton Tavares de Souza, s/n, São Domingos, 24210-346, Niterói, Brazil
[2]INRIA Bordeaux, Sud-Ouest, 200 Avenue de la Veille Tour, 33405 Talence, France
[3]Departamento de Engenharia de Produção, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, Brazil

November 7, 2020

Abstract

This work proposes a partial optimization metaheuristic under special intensification conditions (POPMUSIC) for the classical capacitated vehicle routing problem (CVRP). The proposed approach uses a branch-cut-and-price algorithm as a powerful heuristic to solve subproblems whose dimensions are typically between 25 and 200 customers. The whole algorithm can be seen as the application of local search over very large neighborhoods, starting from a single initial solution. The main computational experiments were carried out on instances having between 302 and 1000 customers. Using initial solutions generated by some of the best available metaheuristics for the problem, POPMUSIC was able to obtain consistently better solutions for long runs of up to 32 hours. In a final experiment, starting from the best known solutions available in CVRP library (CVRPLIB), POPMUSIC was able to find new best solutions for several instances, including some very large ones.

1 Introduction

The *Capacitated Vehicle Routing Problem* (CVRP), introduced by Dantzig and Ramser (1959), is one of the most widely studied problems in combinatorial optimization and operations research. The CVRP is the prototypical vehicle routing problem. New ideas are often first proposed and tested on CVRP and then generalized to other routing variants. It can be defined as follows. Let $G = (V, E)$ be a complete undirected graph, such that $V = \{0, 1, \ldots, n\}$ is the set of vertices and E is the set of edges, where vertex 0 represents a depot and $V_+ = \{1, \ldots, n\}$ a set of customers. There is a non-negative cost c_{ij} for each edge $\{i, j\} \in E$ and a demand d_i for each customer $i \in V_+$. The vehicle capacity is denoted by Q. A route is a path that begins and ends at the depot. A solution consists of a set of routes that respect the following constraints: (i) each customer must be visited exactly once by one of the routes; (ii) the sum of the customer demands in a route can not exceed the vehicle capacity. The objective is to find a set of routes with the minimum total cost.

Given that CVRP is NP-hard, most of the algorithms proposed for this problem are heuristics and metaheuristics (Laporte et al., 2014). The best performing published algorithms are: the

*eduardoqueiroga@id.uff.br
†ruslan.sadykov@inria.fr
‡uchoa@producao.uff.br

iterated local search with set partitioning (ILS-SP) (Subramanian et al., 2013), *knowledge-guided local search* (KGLS) (Arnold and Sörensen, 2019), *hybrid genetic search* (HGS) (Vidal et al., 2012), *slack induction by string removals* (SISR) (Christiaens and Vanden Berghe, 2020), and *fast ILS localized optimization* (FILO) (Accorsi and Vigo, 2020). ILS-SP combines the well-known ILS (Lourenço et al., 2019) with a *set partitioning* (SP) model. The SP model attempts to build unexplored solutions from the set of routes associated with the local minima found by previous runs of the local search. KGLS presents an efficient GLS with three complementary operators using ideas from sequential search and pruning, as well as a problem-specific knowledge to penalize "bad" edges. HGS is a population-based evolutionary search that also makes use of local search (in a step called *education*) and a sophisticated mechanism for controlling population diversity. Among the key components of HGS, we can mention the management of a subpopulation with infeasible solutions, as well as the individual evaluation (a.k.a. *fitness*) driven by the solution cost and its contribution to population diversity. SIRS is a *ruin & recreate* local search guided by simulated annealing (SA) (Kirkpatrick et al., 1983). The ruin procedure removes *strings* (sequence of consecutive customers) from routes (inducing a *capacity slack*), whereas the recreate procedure reinserts the removed customers in the ruined solution in a greedy manner. Finally, FILO is a scalable metaheuristic that employs novel and existing acceleration techniques during the main iterative part based on ILS, whereas it uses an SA-based acceptance criterion to get a continuous diversification.

On the other hand, the exact methods for CVRP have advanced considerably in recent years (Poggi and Uchoa, 2014; Costa et al., 2019). The state-of-the-art results are achieved by *branch-cut-and-price* algorithms (Pecin et al., 2014, 2017a; Pessoa et al., 2020), which combine column and cut generation with several additional mechanisms. According to the experiments carried out in Uchoa et al. (2017), this type of algorithm is able to produce optimal solutions for almost all instances with up to 250 customers, and in some cases, it can solve even larger instances (the largest one already solved has 654 customers). An important observation on the behavior of modern *branch-cut-and-price* algorithms for CVRP, explored in this work, is the following: while instances with more than 200 customers usually take hours or even days to be solved, many instances with up to 150 customers can be solved in few minutes, and many instances with up to 100 customers can be solved in seconds.

The algorithms that hybridize metaheuristics with mathematical programming approaches (Jourdan et al., 2009) are often known as *matheuristics*. Such methods have already been proposed for several optimization problems, including vehicle routing (Archetti and Speranza, 2014; Leggieri and Haouari, 2018). According to Archetti and Speranza (2014), one of the types of *matheuristics* is based on the decomposition of the original problem into smaller subproblems that can be solved (optimally or sub-optimally) through mathematical programming models. This work proposes a simple Partial OPtimization Metaheuristic Under Special Intensification Conditions (POPMUSIC) (Taillard and Voss, 2002) for the CVRP that uses a modern branch-cut-and-price algorithm to solve subproblems (exactly or heuristically). The general idea of POPMUSIC is to optimize subproblems, defined by parts of a solution until a local minimum is reached. This type of algorithm has been shown to be effective for different problems (Taillard and Voss, 2002), including vehicle routing variants (Ostertag et al., 2009; Lalla-Ruiz and Voß, 2020) and the famous traveling salesman problem (Taillard and Helsgaun, 2019).

The remainder of this paper is organized as follows. In Section 2, the proposed POPMUSIC matheuristic for the CVRP is presented. Section 3 describes the modifications to the published branch-cut-and-price algorithm used for solving the subproblems. Section 4 presents and analyses the results of extensive computational experiments. Finally, in Section 5, the final conclusions are presented, as well as suggestions for future work.

2 A POPMUSIC matheuristic for the CVRP

Algorithm 1 shows the pseudocode of the proposed POPMUSIC matheuristic for the CVRP, which has four inputs: (i) an initial solution S; (ii) an algorithm $\mathcal{A}$ to solve subproblems;

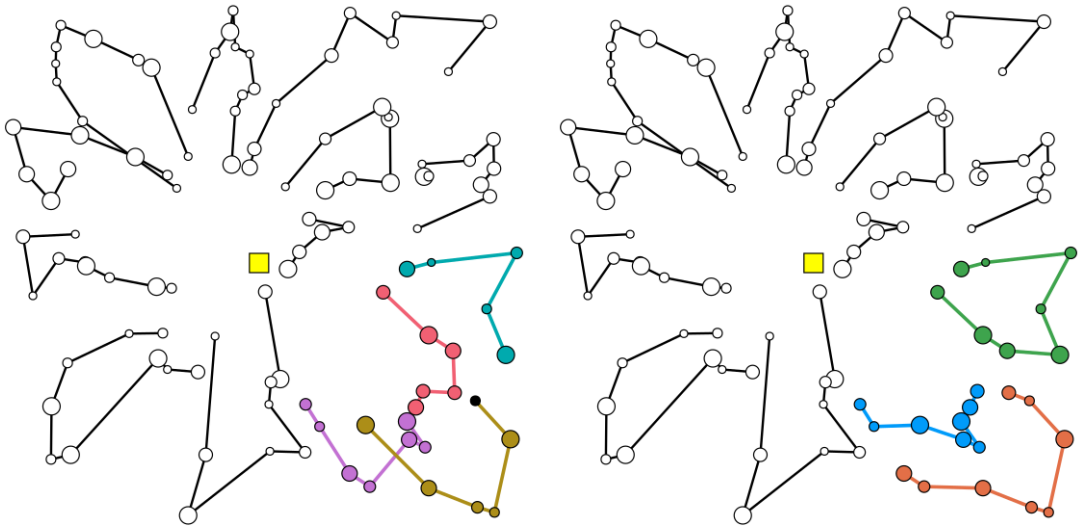
(iii) initial value α for the current *target dimension* dim_{sp} (upper limit on the dimension of subproblems); (iv) step size δ to increase dim_{sp}. The algorithm's output is a (possibly) improved solution S obtained after solving a sequence of subproblems. A solution S is a set $\{r_1, \ldots, r_m\}$ of m routes, whereas the set of customers visited by a route r is denoted by $C(r)$. A set $V_{sp} \subseteq V_+$ represents the CVRP subproblem associated with the subgraph $G[\{0\} \cup V_{sp}]$. We will refer to solutions for subproblems as subsolutions. In addition, in the description of the algorithm, we will consider that $c_{ji} = c_{ij}, \forall \{i, j\} \in E$, and $c_{ii} = 0, \forall i \in V_+$.

Algorithm 1: A POPMUSIC matheuristic for the CVRP

```
1  Data: V, E, c, d, Q
2  Input parameters: initial solution S, algorithm A, α, δ
3  Output: (Possibly) improved solution S
4  dim_sp ← α
5  Π ← ∅
6  while time limit is not exceeded and dim_sp ≤ |V+| do
7      L ← a vector of vertices in V+ in random order
8      for z = 1, 2, ..., n do
9          i ← L[z]
10         /* Build the subproblem for seed i */
11         V_sp ← ∅
12         R ← ∅
13         while |V_sp| < dim_sp do
14             r̂ ← argmin_{r∈S, r∉R} { min_{j∈C(r)} c_ij }
15             if |V_sp| + |C(r̂)| ≤ dim_sp then
16                 V_sp ← V_sp ∪ C(r̂)
17                 R ← R ∪ {r̂}
18             else
19                 Go to the line 21
20         /* If the same or a larger subproblem has not yet been solved, solve V_sp */
21         if V_sp ⊈ V' for all (V', S') ∈ Π then
22             Let S_sp be the subsolution for V_sp in S
23             Π ← Π ∪ (V_sp, S_sp)
24             Solve V_sp with the algorithm A using cost(S_sp) as the initial upper bound
25             Let S'_sp be the subsolution found by the algorithm A, if any
26             if S'_sp is found and cost(S'_sp) < cost(S_sp) then
27                 Replace (V_sp, S_sp) by (V_sp, S'_sp) in Π
28                 Update S by replacing subsolution S_sp by S'_sp
29                 Go to the line 7
30     /* Increase the current target dimension */
31     dim_sp ← dim_sp + δ
```

The algorithm keeps the current target dimension dim_{sp}, which is the upper limit for $|V_{sp}|$. dim_{sp} is initialized to α at line 4. The set $\mathbf{\Pi}$, initialized at line 5, keeps all subproblems already explored during the search together with their subsolutions. Formally, $\mathbf{\Pi}$ is a set of all pairs (V', S'), such that subproblem with set $V' \subset V_+$ of vertices is already solved, and S' is its subsolution. At first, a random permutation of the customers in V_+ produces the array L (line 7). For a given value of dim_{sp}, each customer $i \in V_+$ is used as a *seed* to construct a subproblem V_{sp} at lines 11–19. A subproblem V_{sp} with at most dim_{sp} customers is constructed iteratively by including the routes in the current solution S that are closest to vertex i. The distance from i to each route $r \in S$ is determined by the smallest cost of edges connecting i and the vertices in r (line 14). The routes already included in V_{sp} are stored in R to avoid repetitions (line 17). The construction of subproblem V_{sp} is finished when the next selected route $\hat{r}$ cannot be included in subproblem due to the upper limit dim_{sp} on the subprobem dimension (line 19). Figure 1a illustrates the construction of the subproblem for an instance having 109 customers

and the current target dimension $dim_{sp} = 30$. First, the route containing the seed (in black) is included in the subproblem. The second selected route is the red one, while the third is the blue one, and the fourth is the purple one. Adding a fifth route would exceed dim_{sp}, so the obtained subproblem has 24 customers.



(a) Initial solution and a constructed subproblem. Seed customer is marked in black.

(b) Improved solution after finding a better subsolution

Figure 1: Constructing and solving a subproblem. Depot is the yellow square, and customers are circles with diameter proportional to its demand. For the sake of visualization, the edges adjacent to the depot are not depicted.

The algorithm solves generated subproblem V_{sp} only if it is neither equal nor contained in any subproblem V' already solved before (line 21). Indeed, the $\mathbf{\Pi}$-based condition avoids wasting time on subproblems, i.e., current subsolutions of which are unlikely to be improved because the same or a larger subproblem has been solved already. The solved subproblems together with their solutions are added to set $\mathbf{\Pi}$ at line 23. At line 24, the algorithm $\mathcal{A}$ tries to improve the subsolution S_{sp} of S for current subproblem V_{sp}. As algorithm $\mathcal{A}$, we use a branch-cut-and-price based heuristic described in Section 3. It is important here to use the cost of the known solution S_{sp} for subproblem V_{sp} to improve the performance of the branch-cut-and-price algorithm. Finally, if the solution S'_{sp} found by $\mathcal{A}$ is better than S_{sp}, then S is updated, and the search is restarted for the same target dimension dim_{sp}: all customers will be used again as seeds without increasing dim_{sp}. Figure 1b depicts an example of such an improved solution. If all seeds fail to produce an improving subsolution, then the target dimension dim_{sp} is increased by δ, so that larger subproblems can be explored (line 31). The algorithm is interrupted when the time limit is reached or when the target dimension exceeds the number of customers (line 6). From now on, we refer to Algorithm 1 as `POP`.

3 A branch-cut-and-price heuristic to solve subproblems

The algorithm $\mathcal{A}$ in `POP`, used for solving the subproblems, is an adaptation of the generic Branch-Cut-and-Price (BCP) algorithm proposed by Pessoa et al. (2020), which is a state-of-the-art exact algorithm for many VRP variants, including the CVRP. BCP is a well-known technique that incorporates column and cut generation in a branch-and-bound procedure. In particular, the BCP by Pessoa et al. (2020) includes advanced elements, such as: (i) ng-path relaxation (Baldacci et al., 2011); (ii) rank-1 cuts with limited memory (Jepsen et al., 2008; Pecin

et al., 2014, 2017b; Bulhões et al., 2018); (iii) path enumeration (Baldacci et al., 2008; Contardo and Martinelli, 2014); (iv) rounded capacity cuts (Laporte and Nobert, 1983); (v) bucket graph based bi-directional labeling algorithm (Sadykov et al., 2020); (vi) edge elimination based on reduced costs (Irnich et al., 2010; Sadykov et al., 2020). The reader is referred to Pessoa et al. (2020) for more details about the BCP algorithm.

Since the methodology proposed in this work is a matheuristic one, optimality does not need to be preserved by the BCP. Thus, we turn the BCP algorithm into a heuristic (named BCP_H) by:

- Imposing a branch-and-bound node limit of 10 and time limit of 3,600 seconds;
- Using the *false gap mechanism*, described next;
- Using a restricted master heuristic, described below.

As mentioned above, the BCP algorithm uses an elimination procedure that removes edges from graph G by exploiting reduced cost arguments. In particular, if the minimum reduced cost of a path passing by an edge $e \in E$ is not smaller than the gap between the current upper bound and the lower bound obtained by the column generation procedure, then edge e can safely be removed from the graph G, as no improving solution contains this edge. Removing edges makes subsequent calls to the labeling algorithm used for solving the pricing problem faster.

In addition to the edge elimination, path enumeration is also dependent on the gap. This procedure tries to enumerate all possible paths with reduced cost smaller than the current gap between upper and lower bounds. If path enumeration is successful (i.e., the number of enumerated paths is less than, say, one million; enough to store them in a table), the pricing problem from now on is solved by inspection. The inspection of enumerated paths is usually much faster to perform than to call the labeling algorithm. If the number of enumerated routes is sufficiently small (less than 10,000), the current node in the search tree can be finished by adding all enumerated routes to the restricted master and solving it as an IP (using a general solver like CPLEX).

The previous two paragraphs show the importance of having very good upper bounds (and, therefore, smaller gaps) for reducing the running time of the BCP algorithm. In fact, that is why `POP` solves smaller subproblems first (easy even with not so good upper bounds), so the solution of larger subproblems can benefit from already improved upper bounds. To further reduce the running time, the false gap mechanism artificially decreases the gaps when performing edge elimination and route enumeration. The false gap is defined as $FG = (UB - LB)/FGF$, where the false gap factor $FGF > 1$ is a parameter. Application of the false gap mechanism can result in removing edges or paths which participate in an improving solution. However, experiments indicate that such an outcome occurs rarely when one uses a moderate value for FGF (we tested $FGF = 3$).

Another difference from the default BCP algorithm by Pessoa et al. (2020) consists in using an additional heuristic (similar to the one proposed in Pessoa et al. (2009)). It is called after the convergence of column and cut generation at every node of the search tree. The idea is to further decrease the false gap (dividing it by two in each iteration) until it is possible to complete the path enumeration. Then, the 10,000 routes with smaller reduced costs are used to create an IP that is solved by CPLEX.

4 Computational experiments

The proposed algorithm `POP` was coded in Julia language version 1.4.2. The algorithm BCP_H to solve subproblems was obtained by parameterizing the CVRP demo application of VRPSolver (Bulhões et al., 2020). The parameters are described in Section 4.3. VRPSolver, freely available for academic use, implements the generic BCP algorithm proposed by Pessoa et al. (2020). It

makes use of the BaPCod C++ library (Vanderbeck et al., 2018) as a BCP framework combined with the C++ implementations by (Sadykov et al., 2020) for solving pricing problems, route enumeration, and separation of rank-1 cuts. It also uses CVRPSEP package (Lysgaard, 2003) for separating rounded capacity cuts. Finally, VRPSolver uses CPLEX 12.9 to solve the LP relaxations and the MIPs over the enumerated paths.

All experiments with POP were performed on a 2 Deca-core Haswell Intel Xeon E5-2680 v3 server with 2.50 GHz and 128 GB of RAM, where each algorithm was executed on a single thread for each instance. Parallel runs for several different instances were performed on the same machine to speed up the experiments, effectively reducing the amount of memory allocated to each process.

4.1 Benchmark instances

The tests were performed on the 57 largest instances of the benchmark set X (Uchoa et al., 2017), ranging from 303 to 1001 vertices. Indeed, set X is currently the main benchmark used to assess the performance of all recent exact and heuristic algorithms for the CVRP. We skipped the 43 instances with less than 300 customers because most of those instances are now relatively easy for modern heuristics and even for modern exact algorithms. In fact, 39 of them have proved optimal solutions.

For a deeper analysis of some experiments, we split the 57 instances into two subsets: the subset X_S of 29 instances with $n/K_{min} \leq 10.8$ (i.e., instances with short routes), and the subset X_L composed by the other 28 instances (i.e., instances with long routes). The K_{min} value is an instance attribute that means the minimum possible number of routes that a solution can have. For example, the instance X-n561-k42 belongs to X_L because $n/K_{min} = 561/42 = 13.6 > 10.8$. Extensive experiments presented in Pecin et al. (2017a) indicate that modern branch-cut-and-price algorithms for CVRP, like the one we use to solve subproblems in POP, perform considerably better on instances with shorter routes. Therefore, route size is a factor that is likely to affect the overall performance of POP.

Moreover, in the preliminary experiments used for calibration, we consider a small representative subset X_R having only seven instances. The choice of X_R is described in A.

The gap of a solution S is calculated as $100 \cdot ((cost(S) - \mathrm{BKS})/\mathrm{BKS})$, where BKS is the best known solution in the CVRPLIB[1], only disregarding the solutions found by executions based on the proposed POP approach. Several optimization groups compete for improving the best known solutions for the instances in CVRPLIB. In fact, there were 24 updates in 2020 by seven distinct groups. Updating a BKS in CVRPLIB does not require the publication of an article; one only has to send the improved solution to be checked, even if the improvement is by only one unit. It is not necessary to describe how the solution was obtained. The competing groups may perform long runs of their methods, try several random number seeds, and even resort to special calibration. Thus, those BKSs are likely to be very close to optimum values.

4.2 Obtaining an initial solution

The initialization of POP is a critical issue. Preliminary experiments showed that it did not work so well as a stand-alone algorithm. It means that if it is initialized with a low-quality solution S obtained by a simple constructive heuristic, the overall performance of POP is not competitive with the best existing heuristics. In fact, we are proposing POP essentially as an effective way of improving solutions that are already reasonably good, possibly obtained by running some metaheuristic.

We report results obtained by different variants of POP[1], which uses the HGS metaheuristic by Vidal et al. (2012) to obtain the initial solution. However, as shown in B, the HGS is more effective if the entire algorithm is restarted (with a different random number seed) after $50,000$

[1]BKSs available in the CVRP Library (http://vrp.atd-lab.inf.puc-rio.br/) on October 31, 2020

iterations without any improvement (a method hereafter called HGS^r). Notation $\texttt{POP}^1_t$ defines the variant that starts POP with the solution obtained by HGS^r in t hours. We tested 4 values for t: 0.01 (36 seconds), 0.125 (450 seconds), 0.5 (1800 seconds), and 2 (7,200 seconds). Of course, the initialization time is included in the overall time. For example, in variant $\texttt{POP}^1_{0.5}$, which is run for 32 hours, HGS^r obtains the initial solution in 0.5 hours, and then POP spends 31.5 hours improving the initial solution.

The results obtained by several variants $\texttt{POP}^1$ over the time horizon of 32 hours are compared with those by HGS^r itself. We also perform some comparisons with a second metaheuristic, the ILS-SP proposed by Subramanian et al. (2013). As shown in Uchoa et al. (2017), although the HGS is on average substantially better than the ILS-SP, there are some instances (usually those with very short routes) where the ILS-SP is superior.

4.3 Parameterization of the subproblem solver

The default parameterization of the VRPSolver CVRP demo is calibrated to find optimal solutions for hard instances having around 200-300 customers. As POP needs to solve many smaller problems, we propose an alternative parameterization that works better inside POP. C presents the default and the proposed parameterizations, whose performances are compared in Figure 2 by running $\texttt{POP}^1_{0.5}$ on the X_R instances over 8 hours. The convergence curves of the algorithms show the average gap found at different times.

The figure shows the superior performance obtained when the heuristic version BCP_H is used to solve the subproblems. BCP_H is obtained from the exact BCP using the proposed parameterization and by setting three additional parameters: `RCSPfalseGapFactor` to 3 (this activates the false gap mechanism described in Section 3), `MaxNbOfBBtreeNodeTreated` to 10 (maximum number of nodes in the branch-and-bound tree), and `GlobalTimeLimit` to 3600 seconds (maximum time for solving a subproblem) in the proposed parameterization. All POP results hereafter are obtained using algorithm BCP_H as the subproblem solver.

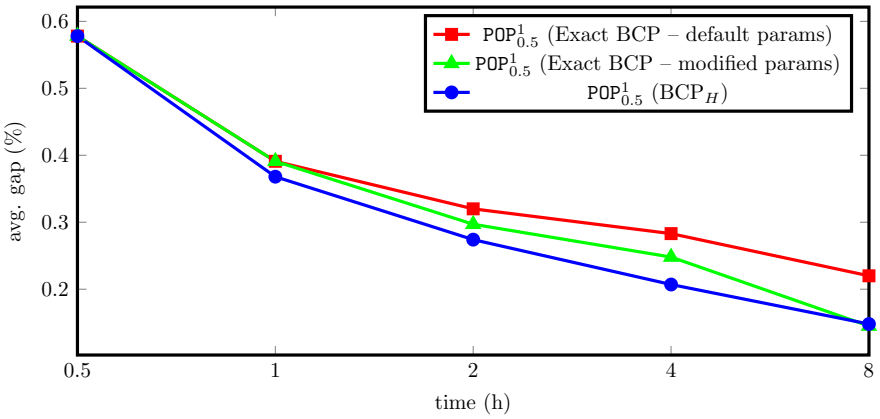


Figure 2: $\texttt{POP}^1_{0.5}$ with three different parameterizations of VRPSolver. The time axis is on a log_2 scale.

4.4 Calibrating parameters α and δ

Table 1 shows the performance of $\texttt{POP}^1_{0.5}$ for different values of parameters α and δ. Each setting was applied to the X_R instances over the horizon of 8 hours. The setting $(\alpha = 50, \delta = 40)$ achieved the best performance for 2, 4, and 8 hours. Therefore, all POP results below are obtained with parameterization $(\alpha = 50, \delta = 40)$.

Table 1: Avg. gap (%) of $\texttt{POP}^1_{0.5}$ on X_R instances for different values of α and δ.

Time (h)	$\alpha = 25$ $\delta = 10$	$\alpha = 25$ $\delta = 25$	$\alpha = 25$ $\delta = 40$	$\alpha = 50$ $\delta = 10$	$\alpha = 50$ $\delta = 25$	$\alpha = 50$ $\delta = 40$	$\alpha = 75$ $\delta = 10$	$\alpha = 75$ $\delta = 25$	$\alpha = 75$ $\delta = 40$
1	0.399	**0.380**	0.448	0.387	0.390	0.383	0.475	0.460	0.440
2	0.327	0.303	0.294	0.310	0.308	**0.263**	0.344	0.328	0.326
4	0.272	0.245	0.244	0.274	0.242	**0.209**	0.241	0.236	0.248
8	0.198	0.170	0.165	0.213	0.193	**0.162**	0.183	0.214	0.198

4.5 Comparison of the algorithms ILS-SP, HGS^r, *and* $\texttt{POP}^1$ *over 32 hours*

Figure 3 and Table 2 show the gap convergence curves for HGS^r and $\texttt{POP}^1$ over the horizon of 32 hours.

Table 2: Average gap (%) of HGS^r and $\texttt{POP}^1$ executions at different times.

Instances	Time (h)	HGS^r	$\texttt{POP}^1_{0.01}$	$\texttt{POP}^1_{0.125}$	$\texttt{POP}^1_{0.5}$	$\texttt{POP}^1_{2}$
All	0.01	1.996	1.996	–	–	–
	0.125	0.836	1.180	0.836	–	–
	0.25	0.626	0.931	**0.564**	–	–
	0.5	0.484	0.708	**0.457**	0.484	–
	1	0.396	0.579	0.355	**0.317**	–
	2	0.330	0.420	0.271	**0.240**	0.330
	4	0.283	0.293	0.206	0.182	**0.171**
	8	0.236	0.201	0.164	0.138	**0.126**
	16	0.210	0.137	0.117	0.099	**0.090**
	32	0.184	0.091	0.084	0.076	**0.064**
X_S	0.01	1.790	1.790	–	–	–
	0.125	0.704	0.754	0.704	–	–
	0.25	0.547	0.524	**0.398**	–	–
	0.5	0.425	0.350	**0.303**	0.425	–
	1	0.345	0.268	**0.228**	0.232	–
	2	0.298	0.196	**0.163**	0.178	0.298
	4	0.262	**0.101**	0.114	0.131	0.108
	8	0.196	**0.058**	0.069	0.084	0.081
	16	0.176	**0.039**	0.048	0.056	0.055
	32	0.162	**0.017**	0.022	0.036	0.037
X_L	0.01	2.209	2.209	–	–	–
	0.125	0.973	1.621	0.973	–	–
	0.25	**0.708**	1.354	0.735	–	–
	0.5	0.546	1.080	0.617	0.546	–
	1	0.448	0.902	0.487	**0.404**	–
	2	0.363	0.652	0.383	**0.305**	0.363
	4	0.306	0.492	0.301	**0.235**	0.237
	8	0.278	0.349	0.263	0.193	**0.173**
	16	0.244	0.238	0.190	0.144	**0.127**
	32	0.206	0.167	0.149	0.117	**0.092**

The performance of $\texttt{POP}^1_{0.01}$ deserves a separate analysis. It illustrates the behavior of POP as an "almost stand-alone" heuristic, starting from a medium quality solution. Such solutions can be rapidly obtained by any modern metaheuristic for the CVRP. The initial solutions provided by running HGS^r for 36 seconds have an average gap of about 2% from the BKS.

- The performance of $\texttt{POP}^1_{0.01}$ on instances with shorter routes (set X_S) is very good. After 900 seconds, it already provides solutions that are significantly better than those from HGS^r. After 4 hours, it is also consistently better than $\texttt{POP}^1_{0.125}$, $\texttt{POP}^1_{0.5}$, and $\texttt{POP}^1_{2}$ and reaches the excellent average gap of 0.017% in 32 hours. It is quite interesting to note that the final gap after 32 hours obtained by $\texttt{POP}^1_t$ on instances X_S gets *worse* as t increases. It seems that worse initial solutions used in $\texttt{POP}^1_{0.01}$ are still flexible enough to be transformed into good final solutions by the POP local search mechanism. On the other hand, the much better initial solutions used by $\texttt{POP}^1_{0.5}$, and $\texttt{POP}^1_{2}$ seem to be biased towards certain local minima that may not be so globally good.
- On the other hand, $\texttt{POP}^1_{0.01}$ performs poorly on instances with long routes (set X_L). It

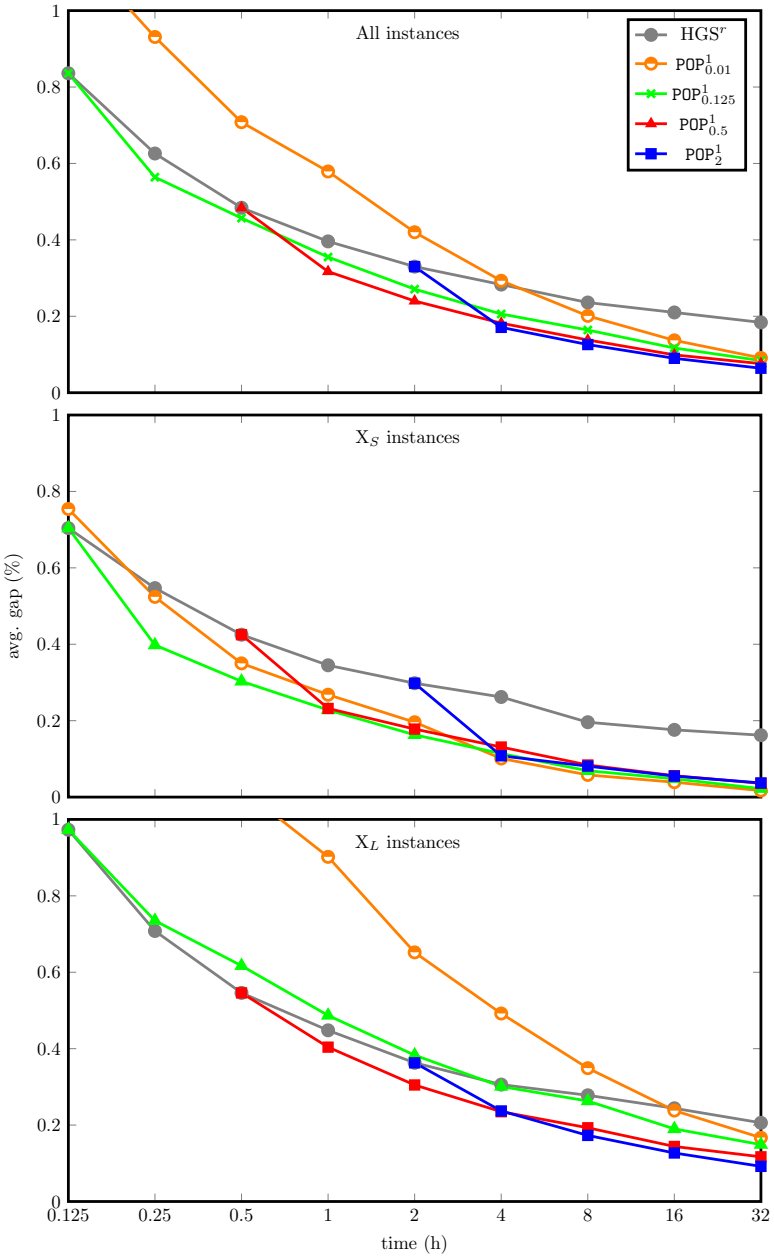


Figure 3: Convergence curves of POP^1 and HGS^r.

Table 3: Best solutions found by ILS-SP, HGS^r, and POP^1 after 32 hours.

Instance	BKS	ILS-SP	HGS^r	POP$^1_{0.01}$	POP$^1_{0.125}$	POP$^1_{0.5}$	POP$^1_2$
X-n303-k21	21736	21840	21739	21863	21837	21750	21751
X-n308-k13	25859	25881	25861	25876	25876	25876	25862
X-n313-k71	94044	94105	94046	94053	94053	94046	94046
X-n317-k53	78355*	78355	78355	78355	78355	78355	78355
X-n322-k28	29834*	29872	29848	29887	29887	29887	29880
X-n327-k20	27532	27743	27555	27573	27576	27573	27576
X-n331-k15	31102*	31108	31103	31103	31103	31103	31103
X-n336-k84	139135	139253	139210	139164	**139111**	139175	**139125**
X-n344-k43	42056	42096	42069	**42055**	**42050**	**42050**	42056
X-n351-k40	25919	26131	25935	**25896**	**25896**	**25896**	**25896**
X-n359-k29	51505	51997	51521	51583	51583	51583	51505
X-n367-k17	22814	22912	22814	22814	22821	22821	22814
X-n376-k94	147713*	147713	147713	147713	147713	147713	147713
X-n384-k52	65941	66382	66048	65941	65999	65956	65947
X-n393-k38	38260*	38273	38260	38260	38260	38260	38260
X-n401-k29	66163	66614	66222	66181	66220	66257	**66156**
X-n411-k19	19718	19811	**19717**	**19712**	19718	**19712**	**19712**
X-n420-k130	107798*	107798	107813	107798	107798	107798	107798
X-n429-k61	65449	65759	65489	65527	65467	65467	65455
X-n439-k37	36391*	36402	36395	36395	36395	36395	36395
X-n449-k29	55233	56131	55336	55332	55236	55259	55258
X-n459-k26	24139	24421	24184	24193	24208	24209	24160
X-n469-k138	221824*	221940	222203	221824	221824	221824	221824
X-n480-k70	89458	89821	89542	**89449**	**89449**	**89449**	**89449**
X-n491-k59	66510	67128	66633	66555	66539	66572	66514
X-n502-k39	69230	69315	69254	69232	69232	69232	69232
X-n513-k21	24201	24275	24201	24248	24249	24201	24201
X-n524-k153	154593*	154698	154774	154593	154593	154593	154593
X-n536-k96	94921	95697	95059	94948	**94915**	95205	95205
X-n548-k50	86700*	86710	86737	86701	86701	86701	86701
X-n561-k42	42717	43016	42744	42758	42773	42758	42758
X-n573-k30	50673	51074	50782	50807	50882	50742	50735
X-n586-k159	190423	190767	190581	**190365**	**190340**	**190375**	**190379**
X-n599-k92	108489	109147	108781	108498	108558	108517	**108462**
X-n613-k62	59535	60318	59671	59561	59544	59606	59656
X-n627-k43	62164	62762	62369	62182	62213	62245	62266
X-n641-k35	63694	64449	64019	63773	63989	63919	63863
X-n655-k131	106780*	106780	106810	106780	106780	106780	106780
X-n670-k130	146332	147286	147144	146346	146340	146411	146461
X-n685-k75	68205	68682	68436	68260	68315	68318	68354
X-n701-k44	81934	82907	82310	82085	81984	81970	82021
X-n716-k35	43412	44091	43572	43443	43491	43498	43489
X-n733-k159	136250	136900	136365	**136245**	**136237**	136278	**136223**
X-n749-k98	77365	78177	77706	77380	77399	**77360**	**77342**
X-n766-k71	114454	115413	114701	114573	114707	114640	114682
X-n783-k48	72445	73627	72809	72696	72592	72605	72704
X-n801-k40	73305	73939	73548	73446	73445	73368	73362
X-n819-k171	158247	159249	158696	**158128**	**158191**	**158222**	**158211**
X-n837-k142	193810	194901	194264	193820	**193793**	**193800**	193822
X-n856-k95	88965	89143	89062	89030	89030	89030	89030
X-n876-k59	99299	100357	99748	99583	99437	99479	99428
X-n895-k37	53860	54777	54266	54112	54080	54125	54045
X-n916-k207	329247	330773	329902	329305	**329213**	329305	329289
X-n936-k151	132725	134564	133440	132859	132882	132863	132942
X-n957-k87	85465	85887	85633	85485	85468	85492	85473
X-n979-k58	118987	120015	119339	119430	119059	119073	119040
X-n1001-k43	72359	73810	72766	72714	72506	72460	72486
Avg. gap (%)		0.629	0.184	0.091	0.084	0.076	**0.064**
Median gap (%)		0.607	0.117	0.027	0.031	0.032	**0.009**
Avg. gap (%) in X_S		0.468	0.162	**0.017**	0.022	0.036	0.037
Median gap (%) in X_S		0.474	0.117	0.005	**0.000**	0.002	**0.000**
Avg. gap (%) in X_L		0.796	0.206	0.167	0.149	0.117	**0.092**
Median gap (%) in X_L		0.779	0.138	0.150	0.135	0.091	**0.073**

takes 4 hours to obtain an average gap of 0.492%, and it reaches the performance of HGS^r only after 16 hours. It is also consistently worse than POP^1_t, for $t \in \{0.125, 0.5, 2\}$.

The variants POP$^1_{0.125}$, POP$^1_{0.5}$, and POP$^1_2$ have a more robust performance. When the complete instance set X is considered, all of them are consistently better than HGS^r alone (i.e., after POP starts, their average gaps are smaller at all times). This is also true when X_S and X_L instances are considered separately. The only exception is the variant POP$^1_{0.125}$, which requires four hours to overcome HGS^r on X_L instances.

Table 3 reports the best solutions found by the algorithms ILS-SP, HGS^r, and POP^1 in 32

hours. BKSs marked with a * are proven optimal solutions. Solutions marked in bold are improvements over the BKSs. The variant $\texttt{POP}^1_2$ achieved the best average and median final gaps, with the exception of the average gap for instances X_S, where it is worse than the variants $\texttt{POP}^1_{0.01}$, $\texttt{POP}^1_{0.125}$, and $\texttt{POP}^1_{0.5}$.

4.6 Comparison of the algorithms HGS20 *and* $\texttt{POP}^2$ over 32 hours

When the work described in this article was already advanced, we were told[2] about the existence of a new implementation of HGS. The new version, specialized to CVRP, is faster and includes one additional neighborhood called SWAP*. We will refer to that yet unpublished algorithm as HGS20. In fact, the performance of HGS20 is much superior to HGS^r, and thus it can definitely be considered as a state-of-the-art metaheuristic for CVRP. In this section, we test if POP can still improve HGS20 solutions.

On September 17, 2020, Thibaut Vidal kindly sent us the detailed results of ten 20-hour runs of the algorithm HGS20 on each of the X instances. Those runs are performed on Intel Xeon Gold 6148 @2.40GHz processors (PassMark single thread rating 2056) that are roughly equivalent to our processors (PassMark single thread rating 1840). Moreover, we have also received the solutions obtained after 0.125, 0.5, and 2 hours. Thus, we use them as initial solutions in the variant $\texttt{POP}^2_t$. Figure 4 depicts the performance of the HGS20 and the variants $\texttt{POP}^2$ over 32 hours (note that HGS stops at 20 hours). We now analyze these results.

- For each running time, HGS20 obtains solutions with about half of the average gap of the solutions obtained by HGS^r, which is a remarkable improvement.
- Considering all X instances, the variant $\texttt{POP}^2_{0.125}$ is consistently worse than HGS20 alone, producing inferior solutions for all times.
- Considering all X instances, the variant $\texttt{POP}^2_{0.5}$ is slightly better than the HGS20 alone. On X_L instances, it only starts to be better at 16 hours.
- Finally, the variant $\texttt{POP}^2_2$ is clearly better than the HSG20 alone, even on X_L instances. This indicates that the proposed approach POP is indeed powerful. It is able to improve solutions obtained by a highly performing metaheuristic, at least in long runs (more than 2 hours).

Table 5 reports instance-by-instance statistics on the solutions found by the HGS20 (after 20 hours) and the variants $\texttt{POP}^2$ (after 20 and 32 hours). For the HGS20, the average cost and the best cost among ten runs are provided. For the variant $\texttt{POP}^2_2$, we give the average cost and the best cost for three runs. We did not have computational resources for running each instance ten times. In order to provide a direct comparison between methods, we also computed the average cost and the best cost of the first three runs of the HGS20.

It is obvious that one can always obtain better solutions by performing multiple runs of any randomized method and picking the best one. But it is still interesting to note that the variant $\texttt{POP}^2_2$ seems to be particularly well suited for performing multiple runs. In fact, for this variant, the average gap for a single 32-hour run is 0.042%, while the average gap for the best of only three runs decreases to 0.018%, a very substantial decrease. For the instances in the set X_S, the approach produces a remarkable gap of -0.001%.

[2]Personal communication from Thibaut Vidal.

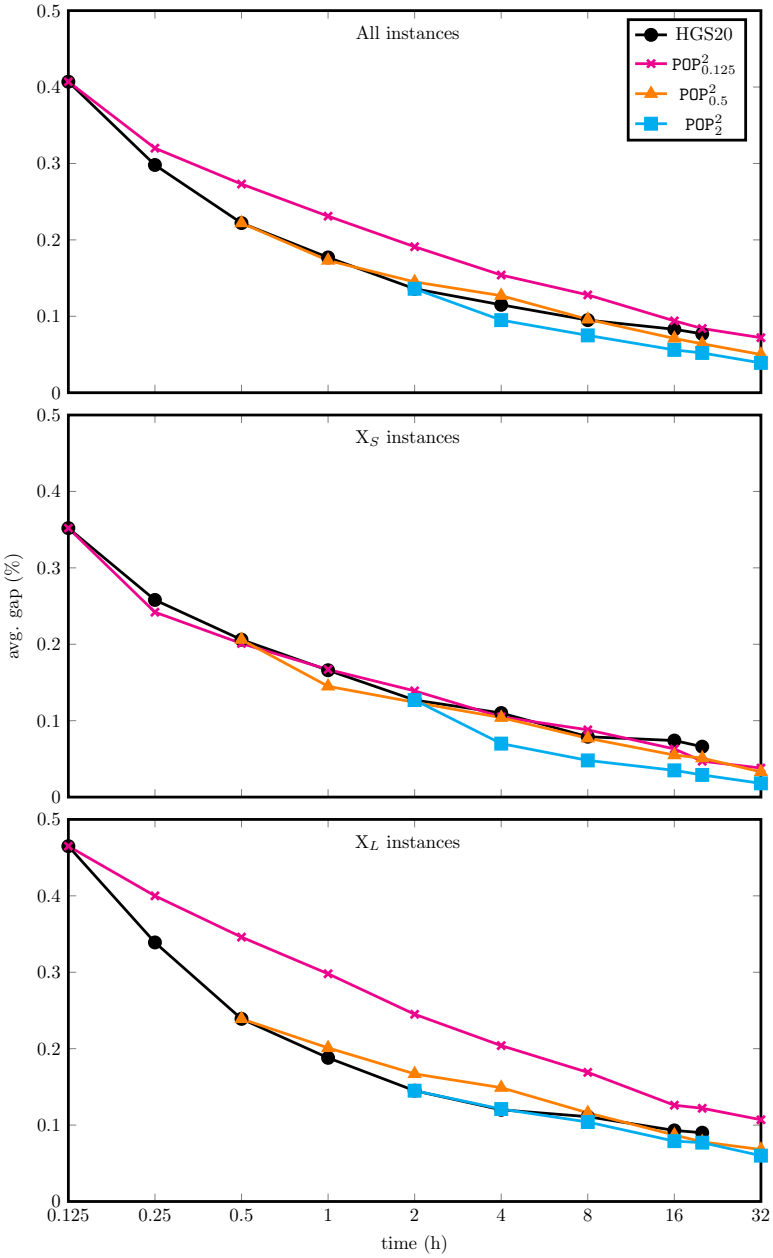


Figure 4: Convergence curves of POP^2 and HGS20.

Table 4: Average gap (%) of HGS20 and $\texttt{POP}^2$ executions at different times.

Instances	Time (h)	HGS20	$\texttt{POP}^2_{0.125}$	$\texttt{POP}^2_{0.5}$	$\texttt{POP}^2_{2}$
All	0.125	0.407	0.407	–	–
	0.25	**0.298**	0.320	–	–
	0.5	0.222	0.273	0.222	–
	1	0.177	0.231	**0.173**	–
	2	0.136	0.191	0.145	0.136
	4	0.115	0.154	0.127	**0.095**
	8	0.095	0.128	0.096	**0.075**
	16	0.083	0.094	0.071	**0.056**
	20	0.077	0.084	0.064	**0.052**
	32	–	0.072	0.050	**0.039**
X_S	0.125	0.352	0.352	–	–
	0.25	0.258	**0.242**	–	–
	0.5	0.206	**0.201**	0.206	–
	1	0.166	0.167	**0.145**	–
	2	0.127	0.139	**0.124**	0.127
	4	0.110	0.105	0.104	**0.070**
	8	0.079	0.088	0.077	**0.048**
	16	0.074	0.063	0.055	**0.035**
	20	0.066	0.047	0.051	**0.029**
	32	–	0.038	0.033	**0.018**
X_L	0.125	0.465	0.465	–	–
	0.25	**0.339**	0.400	–	–
	0.5	0.239	0.346	0.239	–
	1	**0.188**	0.298	0.201	–
	2	0.145	0.245	0.167	0.145
	4	**0.120**	0.204	0.149	0.121
	8	0.111	0.169	0.116	**0.104**
	16	0.093	0.126	0.087	**0.079**
	20	0.090	0.122	0.078	**0.077**
	32	–	0.107	0.068	**0.060**

Table 5: Detailed statistics for HGS20 and $\texttt{POP}_2^2$. Best gaps for 20 hours are underlined.

Instance	BKS	HGS20				POP_2^2			
		20 hours				20 hours		32 hours	
		avg. cost (10×)	best (10×)	avg cost (3×)	best (3×)	avg cost (3×)	best (3×)	avg cost (3×)	best (3×)
X-n303-k21	21736	21737.4	21736	21737.3	21736	21738.0	21738	21738.0	21738
X-n308-k13	25859	25859.0	25859	25859.0	25859	25859.7	25859	25859.7	25859
X-n313-k71	94044	94044.0	94044	94044.0	94044	94044.0	94044	94044.0	94044
X-n317-k53	78355*	78355.0	78355	78355.0	78355	78355.0	78355	78355.0	78355
X-n322-k28	29834*	29834.0	29834	29834.0	29834	29834.0	29834	29834.0	29834
X-n327-k20	27532	27532.0	27532	27532.0	27532	27532.0	27532	27532.0	27532
X-n331-k15	31102*	31102.0	31102	31102.0	31102	31102.3	31102	31102.3	31102
X-n336-k84	139135	139155.8	139137	139156.0	139137	139147.3	**139125**	139147.3	**139125**
X-n344-k43	42056	42053.6	**42050**	42051.7	**42050**	42052.0	**42050**	42052.0	**42050**
X-n351-k40	25919	25925.2	**25909**	25917.3	**25909**	25903.7	**25896**	25903.7	**25896**
X-n359-k29	51505	51535.2	51513	51534.0	51513	51518.7	51505	51518.7	51505
X-n367-k17	22814	22814.0	22814	22814.0	22814	22814.0	22814	22814.0	22814
X-n376-k94	147713*	147713.0	147713	147713.0	147713	147713.0	147713	147713.0	147713
X-n384-k52	65941	65977.1	65957	65979.3	65978	65971.0	65941	65971.0	65941
X-n393-k38	38260*	38260.0	38260	38260.0	38260	38260.0	38260	38260.0	38260
X-n401-k29	66163	66196.9	66180	66203.7	66192	66188.7	66180	66185.0	66178
X-n411-k19	19718	19712.8	**19712**	19712.0	**19712**	19713.7	**19712**	19713.7	**19712**
X-n420-k130	107798*	107804.7	107798	107802.0	107798	107822.0	107798	107798.0	107798
X-n429-k61	65449	65455.4	65449	65451.7	65449	65459.0	65455	65459.0	65455
X-n439-k37	36391*	36394.5	36391	36395.0	36395	36395.0	36395	36395.0	36395
X-n449-k29	55233	55294.1	55265	55282.7	55268	55291.3	55272	55288.0	55262
X-n459-k26	24139	24140.1	24139	24141.0	24139	24157.3	24139	24157.3	24139
X-n469-k138	221824*	221939.6	221848	221936.7	221855	221839.3	221824	221839.3	221824
X-n480-k70	89458	89459.2	**89457**	89461.3	**89457**	89457.0	**89449**	89449.0	**89449**
X-n491-k59	66510	66561.2	66521	66560.3	66521	66520.7	**66489**	66520.7	**66489**
X-n502-k39	69230	69228.5	**69227**	69229.3	**69228**	69226.0	**69226**	69226.0	**69226**
X-n513-k21	24201	24201.0	24201	24201.0	24201	24201.0	24201	24201.0	24201
X-n524-k153	154593*	154605.0	154605	154605.0	154605	154593.0	154593	154593.0	154593
X-n536-k96	94921	94991.5	94940	94991.0	94972	94948.0	**94915**	94943.0	**94915**
X-n548-k50	86700*	86710.0	86704	86706.0	86704	86700.7	86700	86700.7	86700
X-n561-k42	42717	42720.7	42717	42720.3	42717	42717.0	42717	42717.0	42717
X-n573-k30	50673	50747.1	50736	50742.0	50739	50741.0	50739	50738.3	50733
X-n586-k159	190423	190398.9	**190340**	190422.0	**190407**	190359.7	**190349**	190337.0	**190316**
X-n599-k92	108489	108554.2	108490	108562.0	108518	108486.0	**108457**	108484.7	**108453**
X-n613-k62	59535	59619.0	59549	59636.0	59602	59586.7	59536	59582.0	59536
X-n627-k43	62164	62273.3	62241	62275.3	62264	62264.7	62224	62254.0	62223
X-n641-k35	63694	63789.4	63738	63791.7	63758	63815.0	63763	63798.7	63763

Continued on next page

Table 5 – continued from previous page

Instance	BKS	HGS20				POP_2^2			
		20 hours				20 hours		32 hours	
		avg. cost (10×)	best (10×)	avg cost (3×)	best (3×)	avg cost (3×)	best (3×)	avg cost (3×)	best (3×)
X-n655-k131	106780*	106787.6	106780	106789.7	106786	106780.0	106780	106780.0	106780
X-n670-k130	146332	146641.3	146510	146642.7	146624	146514.3	146404	146514.0	146404
X-n685-k75	68205	68312.0	68272	68324.0	68317	68295.0	68257	68281.0	68257
X-n701-k44	81934	82107.6	81998	82152.0	82123	82115.3	82030	82080.0	82030
X-n716-k35	43412	43468.3	43446	43481.0	43460	43455.7	43445	43433.3	**43409**
X-n733-k159	136250	136306.9	136281	136304.0	136298	136213.7	**136195**	136213.7	**136195**
X-n749-k98	77365	77563.4	77463	77543.0	77463	77379.3	**77350**	77342.0	**77294**
X-n766-k71	114454	114687.2	114635	114689.0	114640	114678.7	114658	114627.0	114597
X-n783-k48	72445	72649.8	72550	72665.0	72620	72572.0	72524	72563.7	72515
X-n801-k40	73305	73377.2	73308	73366.7	73353	73387.3	73385	73349.0	73313
X-n819-k171	158247	158331.3	158263	158318.0	158263	158328.7	158298	158298.3	**158225**
X-n837-k142	193810	194023.3	193973	193985.0	193973	193822.0	**193756**	193813.7	**193739**
X-n856-k95	88965	88986.4	88966	88983.7	88966	88989.7	88989	88989.7	88989
X-n876-k59	99299	99557.8	99490	99540.0	99510	99447.0	99428	99419.7	99405
X-n895-k37	53860	54041.2	54007	54028.3	54007	54021.3	53969	54000.0	53960
X-n916-k207	329247	329565.5	329481	329552.7	329539	329325.0	329288	329304.0	329249
X-n936-k151	132725	133116.7	132998	133161.7	133124	132933.7	132900	132898.0	132861
X-n957-k87	85465	85505.4	85473	85504.3	85496	85496.0	85492	85494.3	85487
X-n979-k58	118987	119154.9	119120	119159.0	119130	119146.0	119038	119125.7	119022
X-n1001-k43	72359	72614.5	72541	72625.7	72541	72485.7	72449	72458.0	72427
Avg. gap (%)		0.079	0.043	0.080	0.060	0.052	0.027	**0.042**	**0.018**
Median gap (%)		0.047	0.008	0.045	0.014	0.016	0.000	**0.011**	**0.000**
Avg. gap (%) in X_S		0.068	0.031	0.068	0.048	0.029	0.007	**0.021**	**-0.001**
Median gap (%) in X_S		0.042	0.008	0.040	0.010	0.009	0.000	**0.002**	**0.000**
Avg. gap (%) in X_L		0.092	0.055	0.093	0.073	0.076	0.048	**0.064**	**0.037**
Median gap (%) in X_L		0.049	0.007	0.051	0.026	0.038	0.018	**0.034**	**0.010**

4.7 Directly improving BKSs in CVRPLIB

In the final experiment, we run algorithm POP using the BKS in CVRPLIB as the initial solution. Besides testing the open X instances, we also test very large instances with up to 30,000 customers in the XXL set (Arnold et al., 2019) and also the open instances in the Golden set (Golden et al., 1998). Table 6 presents the improved BKSs found by POP after 32 hours for X and Golden instances, and after 96 hours for the XXL instances.

Table 6: BKSs directly improved by POP

Instance	BKS	Improved BKS
X-n536-k96	94868	**94864**
X-n733-k159	136190	**136188**
X-n766-k71	114454	**114418**
X-n783-k48	72394	**72393**
X-n936-k151	132725	**132715**
X-n979-k58	118987	**118976**
X-n1001-k43	72359	**72355**
Antwerp1	477306	**477277**
Brussels1	501854	**501771**
Flanders1	7241290	**7240874**
Ghent1	469586	**469532**
Leuven1	192851	**192848**
Golden_16	1611.70	**1611.28**

5 Concluding remarks

In this work, we propose POP, a POPMUSIC matheuristic for the classical and highly competitive CVRP. The algorithm is designed to improve a reasonably good initial solution given as an input. The results show that our approach outperforms one of the best published metaheuristics for the CVRP in medium and long runs. POP matheuristic is also competitive in long runs with a state-of-the-art unpublished heuristic, which is specialized to the CVRP. The results are especially good for instances with relatively short routes. Moreover, several best known solutions were improved for literature instances with up to 20,000 customers. This shows a very good scalability of the approach.

POP matheuristic exploits a characteristic of the modern exact algorithms for vehicle routing problems, and in particular, those for the CVRP. If a tight upper bound on the optimum value is provided, those exact algorithms are usually capable of solving to optimality medium-size instances with up to 100–150 customers in a few minutes. Instances with less than 100 customers are usually solved in seconds. Thus, exact (or almost exact, the case of BCP_H) approaches become competitive with the best heuristics for solving such instances. An important advantage of exact approaches is that they "know" when to stop after proving that an improving solution does not exist, whereas traditional heuristics do not possess that information.

POP has interesting features that may be explored in future works:

- It is very different from other existing and well-performing heuristics for the CVRP. This means that their strengths and weaknesses may be complementary. This opens possibilities for many types of hybridization. It could be something simple, like just determining the instance characteristics (besides route size) that make it more or less suited to POP, in order to decide which method should be applied. But it could be something deeper, a full integration where POP and some traditional metaheuristic could take turns on improving parts of a solution and exchange information. This seems to be quite a promising direction for research.

- It is easy to implement, provided that an exact (or nearly exact) code for solving the subproblems is at hand. Given that VRPSolver branch-cut-and-price algorithm is available for academic use and can solve many routing variants other than CVRP, it is natural to try algorithms similar `POP` on those variants. Of course, there is no guarantee that a straightforward adaptation will obtain good results. Thus, there is room for research on extensions of `POP` that are more suited for other particular routing problems.
- It is naturally parallelizable. The current sequential version of `POP` may take a few hours to obtain high-quality solutions and can not be used in practical situations that require faster solutions. That limitation could be much reduced by solving several subproblems in parallel, a natural feature of any POPMUSIC approach. In fact, the larger the instance, the more parallelizable the method becomes.

The last remark is that the underlying implementation of the BCP solver used in `POP` was not changed by us other than by modifying external parameters. Thus, there is a large potential to improve the efficiency of `POP` by "going inside the black box". A property that may be exploited is that the subproblems to be solved are often very similar to some already solved subproblem. Keeping information from previous runs, like the generated columns and cuts, may accelerate the algorithm.

Acknowledgements

This study was financed in part by CAPES - Finance Code 001, by Capes PrInt UFF no 88881, by CNPq grant 313601/2018-6, and by FAPERJ grant E-26/202.887/2017.

This research was also supported by *Programa Institucional de Internacionalização* (PrInt) from CAPES as part of the project REMATCH (process number 88887.310261/2018-00).

The experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/`).

We would like to thank Anand Subramanian for kindly sending us his code.

We are grateful to Thibaut Vidal for kindly providing us the results of his new heuristic and for insightful discussions.

References

Luca Accorsi and Daniele Vigo. A fast and scalable heuristic for the solution of large-scale capacitated vehicle routing problems. Technical report, University of Bologna, 2020.

Claudia Archetti and M. Grazia Speranza. A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2(4):223–246, Nov 2014. ISSN 2192-4414.

Florian Arnold and Kenneth Sörensen. Knowledge-guided local search for the vehicle routing problem. *Computers & Operations Research*, 105:32 – 46, 2019. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2019.01.002.

Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers & Operations Research*, 107:32 – 42, 2019. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2019.03.006.

Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, Oct 2008. ISSN 1436-4646. doi: 10.1007/s10107-007-0178-5.

Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283, 2011. doi: 10.1287/opre.1110.0975.

Teobaldo Bulhões, Guillaume Marques, Artur Pessoa, Eduardo Queiroga, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. VRPSolver: a branch-cut-and-price based exact solver for vehicle routing and some related problems. `https://vrpsolver.math.u-bordeaux.fr/`, 2020. Accessed: 2020-10-16.

Teobaldo Bulhões, Artur Pessoa, Fábio Protti, and Eduardo Uchoa. On the complete set packing and set partitioning polytopes: Properties and rank 1 facets. *Operations Research Letters*, 46 (4):389 – 392, 2018. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2018.04.006.

Jan Christiaens and Greet Vanden Berghe. Slack induction by string removals for vehicle routing problems. *Transportation Science*, 54(2):417–433, 2020.

Claudio Contardo and Rafael Martinelli. A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optimization*, 12:129 – 146, 2014. ISSN 1572-5286. doi: https://doi.org/10.1016/j.disopt.2014.03.001.

Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1): 80–91, 1959.

Bruce L. Golden, Edward A. Wasil, James P. Kelly, and I-Ming Chao. *The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results*, pages 33–56. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5755-5. doi: 10.1007/978-1-4615-5755-5_2.

Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, 22(2):297–313, 2010.

Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

Laetitia Jourdan, Matthieu Basseur, and E-G Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. ISSN 0036-8075. doi: 10.1126/science.220.4598.671.

Eduardo Lalla-Ruiz and Stefan Voß. A POPMUSIC approach for the multi-depot cumulative capacitated vehicle routing problem. *Optimization Letters*, 14(3):671–691, Apr 2020. ISSN 1862-4480.

G. Laporte and Y. Nobert. A branch and bound algorithm for the capacitated vehicle routing problem. *Operations-Research-Spektrum*, 5(2):77–85, Jun 1983. ISSN 1436-6304. doi: 10.1007/BF01720015.

Gilbert Laporte, Stefan Ropke, and Thibaut Vidal. *Chapter 4: Heuristics for the Vehicle Routing Problem*, pages 87–116. 2014. doi: 10.1137/1.9781611973594.ch4.

Valeria Leggieri and Mohamed Haouari. A matheuristic for the asymmetric capacitated vehicle routing problem. *Discrete Applied Mathematics*, 234:139 – 150, 2018. ISSN 0166-218X. doi: https://doi.org/10.1016/j.dam.2016.03.019. Special Issue on the Ninth International Colloquium on Graphs and Optimization (GO IX), 2014.

Helena Ramalhinho Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer, 2019.

Jens Lysgaard. Cvrpsep: A package of separation routines for the capacitated vehicle routing problem. Tech. report, Aarhus University, Denmark, 2003.

A Ostertag, K F Doerner, R F Hartl, E D Taillard, and P Waelti. Popmusic for a real-world large-scale vehicle routing problem with time windows. *Journal of the Operational Research Society*, 60(7):934–943, 2009.

Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. In Jon Lee and Jens Vygen, editors, *Integer Programming and Combinatorial Optimization*, pages 393–403, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07557-0.

Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, 9(1):61–100, Mar 2017a. ISSN 1867-2957.

Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. Limited memory rank-1 cuts for vehicle routing problems. *Operations Research Letters*, 45(3):206 – 209, 2017b. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2017.02.006.

Artur Pessoa, Eduardo Uchoa, and Marcus Poggi de Aragão. A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks: An International Journal*, 54(4):167–177, 2009.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, 183(1):483–523, Sep 2020. ISSN 1436-4646. doi: 10.1007/s10107-020-01523-z.

Marcus Poggi and Eduardo Uchoa. Chapter 3: New exact algorithms for the capacitated vehicle routing problem. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 59–86. SIAM, 2014.

Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. A bucket graph based labeling algorithm with application to vehicle routing. *Transportation Science*, Ahead of Print, 2020.

Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. *Computers & Operations Research*, 40(10):2519 – 2531, 2013. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2013.01.013.

E. Taillard and K. Helsgaun. POPMUSIC for the travelling salesman problem. *European Journal of Operational Research*, 272(2):420 – 429, 2019. ISSN 0377-2217.

Éric D. Taillard and Stefan Voss. *Popmusic — Partial Optimization Metaheuristic under Special Intensification Conditions*, pages 613–629. Springer US, Boston, MA, 2002. ISBN 978-1-4615-1507-4.

Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845 – 858, 2017. ISSN 0377-2217.

F. Vanderbeck, R. Sadykov, and I. Tahiri. BaPCod – a generic branch-and-price code. Available at `https://realopt.bordeaux.inria.fr/?page_id=2`, 2018.

Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012. doi: 10.1287/opre.1120.1048.

A A representative small subset of X

A minimum subset of the instances X which covers all the characteristics considered in Uchoa et al. (2017) is described below:

- Route size (interval for n/K_{min}):
  - $[3, 5]$: X-n469-k138
  - $(5, 8]$: X-n670-k130
  - $(8, 11]$: X-n393-k38
  - $(11, 14]$: X-n561-k42
  - $(14, 17]$: X-n979-k58
  - $(17, 20]$: X-n801-k40
  - $(20, 25]$: X-n716-k35
- Depot positioning:
  - Random: X-n670-k130, X-n716-k35
  - Center: X-n393-k38, X-n561-k42
  - Corner: X-n469-k138, X-n801-k40, X-n979-k58
- Customers distribution:
  - Random: X-n469-k138, X-n670-k130, X-n801-k40
  - Clustered: X-n716-k35, X-n979-k58
  - Random-clustered: X-n393-k38, X-n561-k42
- Customers demands:
  - Unitary: X-n801-k40
  - Small values, large CV[3]: X-n561-k42
  - Small values, small CV: X-n393-k38
  - Large values, large CV: X-n716-k35
  - Large values, small CV: X-n469-k138
  - Depending on quadrant: X-n979-k58
  - Many small values, few large values: X-n670-k130

Thus, the subset X_R is composed by: X-n393-k38, X-n469-k138, X-n561-k42, X-n670-k130, X-n716-k35, X-n801-k40, X-n979-k58. The reader is referred to Uchoa et al. (2017) for a detailed description of all characteristics.

B Comparison of HGS and HGS^r

Figure 5 compares the performance of a single execution of HGS and HGS^r for all X instances over 8 hours. The convergence curves of both algorithms report the average gap (considering the gap obtained for each instance) found at different times. The final average gaps obtained by HGS and HGS^r were 0.281% and 0.236%, respectively.

[3]Coefficient of variation

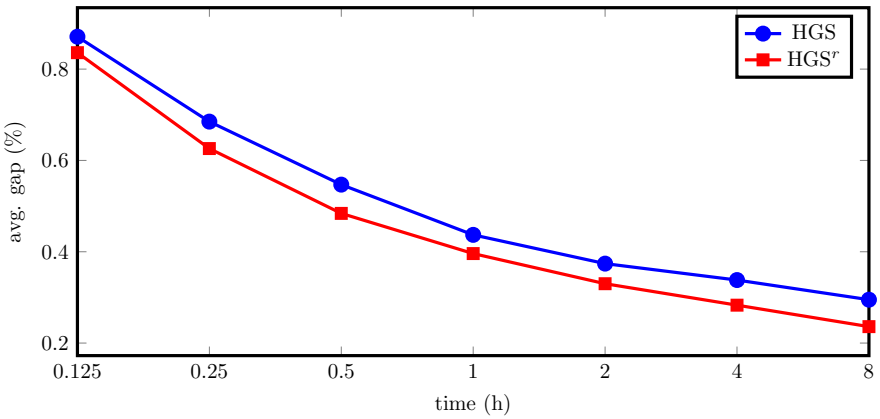


Figure 5: Comparison of HGS and HGS^r w.r.t. the convergence curve based on the average gap for all X instances over 8 hours. The time axis is on a log_2 scale.

C VRPSolver Parameterizations

Table 7 shows the default VRPSolver CVRP parameterization, as well as the changed parameterization in the version used to solve subproblems in `POP`. The reader is referred to the documentation of Bulhões et al. (2020) to obtain the description of each parameter. The parameters which have special notation indicated in the second column of Table 7 are also described in Pessoa et al. (2020). Default values for the last four parameters are not defined because these parameters are not active when the restricted master heuristic is not used. `RCSPmaxNumOfEnumSolsForEndOfNodeMIP` is a previously undocumented VRPSolver parameter. If the number of enumerated routes gets becomes smaller than `RCSPmaxNumOfEnumSolutionsForMIP`, at any point of a node solution, then the node is immediately solved by MIP.

Table 7: Default and used parameters of the VRPSolver CVRP application used in

Parameter	Notation	Default value	Used value
`RCSPhardTimeThresholdInPricing`	τ^{hard}	25 secs	8 secs
`RCSPstopCutGenTimeThresholdInPricing`	τ^{soft}	10 secs	3 secs
`RCSPnumberOfBucketsPerVertex`	τ^{hard}	25	50
`RCSPmaxNumOfLabelsInEnumeration`	ω^{labels}	$5 \cdot 10^6$	$3 \cdot 10^5$
`RCSPmaxNumOfEnumeratedSolutions`	ω^{routes}	$5 \cdot 10^6$	10^6
`RCSPmaxNumOfEnumSolutionsForMIP`	ω^{MIP}	10^4	$5 \cdot 10^3$
`RCSPmaxNumOfEnumSolsForEndOfNodeMIP`	–	10^4	10^4
`RCSPuseBidirectionalSearch`	ϕ^{bidir}	2	1
`RCSPrankOneCutsMemoryType`	θ^{mem}	0	0
`CutTailingOffThreshold`	δ^{gap}	0.015	0.03
`StrongBranchingPhaseOneCandidatesNumber`	ζ_1^{num}	100	50
`StrongBranchingPhaseOneTreeSizeEstimRatio`	ζ_1^{estim}	0.2	0.2
`StrongBranchingPhaseTwoCandidatesNumber`	ζ_2^{num}	5	3
`StrongBranchingPhaseTwoTreeSizeEstimRatio`	ζ_2^{estim}	0.02	0.02
`MaxTimeForRestrictedMasterIpHeur`	χ^{rm}	-1 (off)	40
`CallFrequencyOfRestrictedMasterIpHeur`	–	–	1
`MIPemphasisInRestrictedMasterIpHeur`	–	–	1
`RCSPmaxNumOfLabelsInHeurEnumeration`	–	–	10^5
`MaxNumEnumSolsInRestrictedMasterIpHeur`	–	–	10^4

Table 8 shows the additional parameters used for obtaining BCP_H. While they reduce running times, the optimal solution of a subproblem may be missed.

Table 8: Additional parameters for obtaining BCP_H

Parameter	Notation	Default value	Used value
`GlobalTimeLimit`	–	∞	3600 secs
`MaxNbOfBBtreeNodeTreated`	–	∞	10
`RCSPfalseGapFactor`	–	0 (off)	3