RAIRO-Oper. Res. 55 (2021) S719–S726
https://doi.org/10.1051/ro/2020002

RAIRO Operations Research
www.rairo-ro.org

A BRANCH-AND-CUT AND MIP-BASED HEURISTICS FOR THE PRIZE-COLLECTING TRAVELLING SALESMAN PROBLEM

Glaubos Clímaco[1,*], Luidi Simonetti[1] and Isabel Rosseti[2]

Abstract. The Prize Collecting Traveling Salesman Problem (PCTSP) represents a generalization of the well-known Traveling Salesman Problem. The PCTSP can be associated with a salesman that collects a prize in each visited city and pays a penalty for each unvisited city, with travel costs among the cities. The objective is to minimize the sum of the costs of the tour and penalties, while collecting a minimum amount of prize. This paper suggests MIP-based heuristics and a branch-and-cut algorithm to solve the PCTSP. Experiments were conducted with instances of the literature, and the results of our methods turned out to be quite satisfactory.

Mathematics Subject Classification. 90C10, 90C11, 90C08, 90C05.

Received August 12, 2019. Accepted December 23, 2019.

1. Introduction

The prize-collecting traveling salesman problem (PCTSP) is a var of the well-known traveling salesman problem (TSP), and so an NP-hard problem. In the PCTSP, a salesman collects a prize p_i in each city visited and pays a penalty w_i for each city not visited, considering travel costs c_{ij} between the cities. The objective function is to minimize the sum of travel costs and penalties paid, while including in the tour enough cities to collect a minimum prize $p_{\min}$. In this tour, each city can be visited at most one time. Figure 1 illustrates a PCTSP solution containing seven vertices, and for visual purposes, consider a Euclidean distance between the vertices.

The PCTSP was initially formulated by Balas [3] as a model for scheduling the daily operations of a steel rolling mill, and since then has been studied by several researchers to solve problems such as, the helicopter tour planning for offshore oil platforms, and the design of tourist routes [4]. Several authors proposed different approaches to tackle PCTSP. Fischetti and Toth [12] presented some bounding procedures, as well as a branch-and-bound algorithm, and applied it to instances with up to 200 nodes.

Bienstock *et al.* [7] devised a new formulation for PCTSP which uses cut-set constraints for sub-cycle eliminations in the route. Bienstock *et al.* [7] equally developed an approximation algorithm with constant bound that combines a linear relaxation, with the Christofides algorithm [14], widely used for resolution of the TSP.

Keywords. prize-collecting traveling salesman problem, MIP heuristics, Branch-and-cut.

[1] Universidade Federal do Rio de Janeiro, PESC-COPPE, Cidade Universitária, Centro de Tecnologia, Bloco H, 21941-972 Rio de Janeiro, Brazil.

[2] Universidade Federal Fluminense, Instituto de Computação, Rua Passo da Pátria 156 São Domingos, 24210-240 Niterói, RJ, Brazil.

[*]Corresponding author: glaubos@cos.ufrj.br

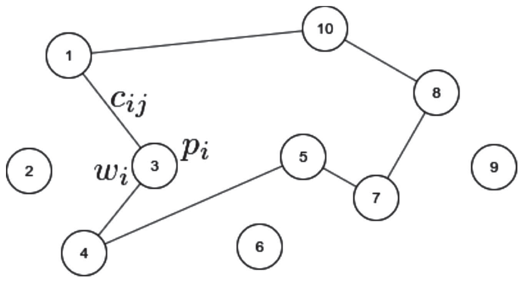

Figure 1. A PCTSP solution with seven vertices.

Dell'Amico *et al.* [11] presented a Lagrangian heuristic to obtain an upper bound, by relaxing the minimum prize constraint, dualizing it in a Lagrangian way, and then applying the sub-gradient method. Awerbuch *et al.* [2] elaborated an algorithm of complexity $(\log_2 n)$ for the Quota TSP. This last problem is similar to the PCTSP, with the only difference being that it does not consider penalties for unvisited cities. Therefore, with some transformations, the algorithm proposed by these authors also includes the PCTSP. Torres *et al.* [23] proposed GRASP [21] heuristics for the orienteering problem (OP) and the PCTSP, and a new formulation for the PCTSP, based on flow constraints for the sub-cycle elimination.

Chaves *et al.* [8] proposed: (i) a new mathematical formulation by combining both models of Torres *et al.* [23] and Balas [3]; and (ii) two heuristics based on clustering search to solve the PCTSP. These heuristics are similar, and their main difference is how initial solutions are built. The first heuristic generates solutions making use of an evolutionary process (ECS), while the second one creates initial solutions employing a GRASP heuristic associated with a VNS/VND [17] method (CS*).

Lastly, to the best of our knowledge, the last work addressing the PCTSP was carried out by Pedro *et al.* [19], in which the authors proposed a solution through a Tabu Search (TS) [16], which incorporates the GENIUS [13] heuristic to its construction phase, and a 2-opt local search.

The scientific contributions of this work can be summarized as: a branch-and-cut algorithm, which has obtained unknown optimal solutions so far; and two new MIP-based heuristics. The remainder of this paper is organized as follows. In Section 2 an integer linear programming (ILP) formulation of Bienstock *et al.* [7] is described. In Sections 3 and 4, the proposed approaches are presented. In Section 5, some numerical results are exhibited to evaluate our proposals. Lastly, in Section 6, some conclusions and subsequent investigations are drawn.

2. Mathematical formulation

In this section we present the mathematical formulation of Bienstock *et al.* [7]. Let $G = (V, E)$ be a complete and undirected graph, in which V and E are the sets of vertices and edges, respectively. Associated with each edge $e = (i, j) \in E$ there is a cost c_e satisfying the triangle inequality, and for each vertex $i \in V$ there is a non-negative penalty w_i and a prize p_i.

The formulation requires the binaries variables $x_{ij} = 1$ if edge $(i, j) \in E$ is part of the tour; or $x_{ij} = 0$, otherwise; and $y_i = 1$, if vertex $i \in V$ is visited, and $y_i = 0$ otherwise. We also define $\delta(S) = \{(i, j) \in E \mid i \in S, j \in V \setminus S\}$, $\delta(i)$ as a set of all incidents edges to the vertex i, and a root vertex u. Hence, the PCTSP can be formulated as follows:

$$(P_1) \qquad \text{Min} \sum_{e\in E} c_e x_e + \sum_{i\in V} w_i(1 - y_i) \tag{2.1}$$

$$\text{subject to:} \sum_{e\in\delta(i)} x_e = 2y_i, \qquad \forall i \in V \tag{2.2}$$

$$\sum_{i\in V} p_i y_i \geq p_{\min} \tag{2.3}$$

$$\sum_{e\in\delta(S)} x_e \geq 2y_i, \qquad \forall (S \subset V \setminus \{u\}, i \in S) \tag{2.4}$$

$$y_i \in \{0,1\}, \qquad \forall i \in V \tag{2.5}$$

$$x_e \in \{0,1\}, \qquad \forall e \in E. \tag{2.6}$$

The objective function (2.1) aims to minimize the sum of travel costs and penalties. The degree constraints (2.2) ensure that a feasible solution goes exactly once through each visited vertex. Constraint (2.3) imposes a minimum bound on the collected prize. The cut-set constraints (2.4) ensure the connectivity of the route, and along with constraints (2.2), are responsible for eliminating sub-cycles in the route. Finally, constraints (2.5) and (2.6) impose that all variables need to be 0-1.

3. Branch-and-cut algorithm

The model proposed by Bienstock *et al.* [7] contains an exponential number of cut-set constraints (2.4). However, the violation of such constraints can be checked in polynomial time, which allows us to apply a branch-and-cut algorithm. In this section, we present an exact B&C for the PCTSP, based on formulation (2.1)–(2.6).

As usual, the family of exponential size constraints (2.4), are initially relaxed. Then, after each LP iteration, they are separated to detect whether constraints of this family are violated by the current LP solution. If so, the detected violated constraints are integrated to the current relaxation, and the strengthened relaxation is solved.

We used the Mixed Integer Programming (MIP) solver Gurobi to solve the LP relaxations and to implement the search tree. Thus, the strategy described above has been incorporated into an enumeration scheme, in which it is applied not only to the root node of the enumeration tree, but also to all generated nodes. We now describe the separation procedure.

3.1. Separation procedure

The algorithm starts with all integrality conditions relaxed and only a subset of constraints. In the initial relaxation, we include all constraints of P_1, except for the cut-set (2.4). For the separation procedure, let $G^* = (V^*, E^*)$ be the support graph associated with an optimal solution $S = (x^*, y^*)$ to the current relaxation, in which $V^* = \{i \in V : y_i^* > 0\}$ and $E^* = \{e \in E : x_e^* > 0\}$.

Consider x_e^* as the capacity value of edge $e \in E^*$. The cut-set constraints are separated when the LP solution S is integer or fractional, and for each case, we proceeded as follows. For any $i \in V \setminus \{u\}$, a minimum (u, i)-cut is found in G^* through a maximum-flow/min-cut algorithm; if the output is less than $2y_i$, then we have a violated constraint. In this case, the next step is to add this constraint to the model and resolve it.

For the min-cut procedure, we used the Goldberg's pre-flow push algorithm [9] which runs in $O(|V|^2\sqrt{|E|})$ for each vertex. Therefore, we obtain an overall complexity of $O(|V|^3\sqrt{|E|})$ to find a violated cut-set.

4. MIP heuristics

Mixed Integer Programming (MIP) is a powerful tool to model and solve hard combinatorial optimization problems, and since MIP-solving is NP-hard [24], heuristic methods for it are of high interest. Heuristic methods are often used, since in several cases, they provide feasible solutions quickly, in practice generally meeting customer demand, and avoiding unnecessary explorations of nodes in the branch-and-bound tree. In the following, we present the proposed MIP-based heuristics for the PCTSP.

4.1. Relax-and-fix

The relax-and-fix (RF) is a heuristic that generates a solution by solving several small MIPs. It is performed by fixing or relaxing most of the binary variables, enforcing only a few to be integer and then optimizing them [22]. In a preliminary experiment, we obtained good dual bounds by solving a particular relaxation of the model P_1, in which each variable x_e was relaxed and the domain of all variables y_i was maintained. We called this new model P_2, and it can be stated as follows:

$$\begin{aligned} (P_2) \qquad & \text{Min} \quad \sum_{e\in E} c_e x_e + \sum_{i\in V} w_i(1-y_i) \\ & \text{subject to } ((2.2)-(2.5)) \text{ plus:} \\ & x_e \in \mathbb{R}_+, \qquad\qquad \forall e \in E. \end{aligned} \tag{4.1}$$

The RF proposed in this paper, aims to find a feasible solution by successive applications of a B&B algorithm followed by fixations of variables x_e, in increasingly restrictive versions of the model P_2. Initially, a B&B is applied on P_2 and then, when it is finished, a relaxed variable x_e^* is fixed according to a fractionality criterion. Then, we have an updated version of P_2, with $x_e = 1$ or $x_e = 0$. Once updated, the model P_2 is solved again and a new variable is fixed. This process is repeated until there are no more variables x_e with fractional values. The RF always provided a feasible solution in our experiments, therefore, no recovery strategy was needed in case the current fixing of the variables turns out to be infeasible. It is noteworthy that although MIP solver applies a B&B to P_2, the branching is only performed on the y variables.

The pseudo-code of Algorithm 1 presents the relax-and-fix heuristic devised. In line 1, a solution $S(x, y)$ is obtained by applying a B&B on P_2. In line 2, the set $\bar{I}$ is initialized containing all the indexes of variables with fractional values in x. From lines 3 to 9, while $\bar{I}$ is not empty *i.e.*, there are variables of x with fractional values, the following steps are performed: (i) the variable with the lowest fractionality (the closest variable to a binary value) is chosen and then fixed in the model with the nearest binary value; and in case of ties, the variable with the lowest edge cost c_e is chosen; (ii) the model P_2 is updated, since variable x_e has now been fixed; (iii) the model P_2 that has just been updated, is solved; and (iv) the set $\bar{I}$ is updated. In line 10, the final solution is returned.

Algorithm 1. RF (P_2).

```
1: $S(x,y) \leftarrow$ solve($P_2$)
2: $\bar{I} \leftarrow$ index set of all fractional variables of $x$
3: while $\bar{I} \neq \emptyset$ do
4:     select $e$ from $\bar{I}$ such that $x_e$ has the lowest fractionality
5:     fix $x_e$ to the nearest binary
6:     update($P_2$)
7:     $S(x,y) \leftarrow$ solve($P_2$)
8:     update($\bar{I}$)
9: end while
10: return $S$
```

4.2. Diving heuristic

The state-of-the-art MIP solvers make use of LP-based branch-and-bound techniques, in which the branching process focuses on two different aims: on the one hand, fractional variables should be driven towards integrality to find feasible solutions. On the other hand, the dual bound should be raised to prove the optimality [1]. To achieve feasible solutions quickly, it is desirable to focus on bounding variables such that the number of fractional variables decreases [5].

Following this idea, the diving heuristics try to bound/fix LP-solution variables to promising values. The name of these heuristics came from the fact that they "quickly go down" the branch-and-bound tree. In this paper, we propose a MIP heuristic based on the guided diving heuristic (GD) [6], that makes use of a guide solution to fix variables. In the following, we describe how the diving idea was implemented in our approach.

The GD heuristic consists of performing a B&B algorithm to the model P_1 once, and while the B&B runs, the diving procedure is applied whenever the current node LP-solution $S^*(x^*, y^*)$ is found and there is a guide $\tilde{x}$, the incumbent (integer) solution obtained during the B&B itself. Implementation details of the diving procedure are presented on the Algorithm 2.

Algorithm 2. Diving $(\hat{x}^*, \tilde{x})$.

```
1:  $j \leftarrow 0$, $frac \leftarrow \infty$;
2:  while $(j < |E|)$ do
3:      if $(0 < x^*_j < 1)$ then
4:          if $(|x^*_j - \tilde{x}_j| < frac)$ then
5:              $frac \leftarrow |x^*_j - \tilde{x}_j|$
6:              $ind \leftarrow j$.
7:          end if
8:          $j \leftarrow j + 1$;
9:      end if
10: end while
11: Fix $x^*_{ind}$ to the value of $\tilde{x}_{ind}$
12: Update($P_1$)
```

The Algorithm 2 receives as parameters, the LP-solution of the current tree node and the guide solution. From lines 2 to 10, we obtain the variable x^*_j with the lowest fractionality $|x^*_j - \tilde{x}_j|$ (x^*_j with integer values are not considered); and in line 11, x^*_j is fixed in the model P_1 as its respective value in $\tilde{x}_j$ (either 0 or 1). Finally, the model P_1 is updated and the B&B continues, hopefully faster from now on. As for the RF, the application of a recovery strategy was unnecessary either.

5. Computational results

To validate the presented proposals, experiments were conducted with instances proposed by Chaves *et al.* [8]. These instances consist of problems with $|V| \in \{11, 21, 31, 51, 101, 251, 501\}$, randomly generated at the following intervals. Travel cost between vertices: $c_{ij} \in [50, 1000]$; penalty associated to each vertex: $w_i \in [1, 750]$; prize associated to each vertex: $p_i \in [1, 100]$. The minimum prize to be collected, $PRIZE$, represents 75% of the sum of the prizes of all vertices. These test problems are available at `http://www.lac.inpe.br/~lorena/instancias.html`.

All approaches introduced in this paper were implemented in the *C++* language and compiled with *g++ 5.4.0*. The tests were run on an Intel Core i7-6900 K @ 3.20 GHz machine with 16 GB of RAM under the operating system Linux Ubuntu 16.04 LTS with parallel processing features disabled. For the implementation and execution of the mathematical formulations, the Gurobi solver (Version 6.5.2) was used, with its heuristics, cuts and preprocessing disabled. Also, a one-hour processing timeout was determined.

5.1. Mathematical formulations

The flow-based formulations of Chaves *et al.* [8] and Torres *et al.* [23] were implemented in this work, and then compared to the proposed B&C. Results of this comparison are reported in Table 1, in which the first column indicates the name of the instances and for each model, the solution value and spent time (in seconds) are presented. The best results are highlighted in bold, and the "–" symbol indicates that no solution was found within one-hour CPU time.

We remark that, in the literature, the formulation of Chaves *et al.* [8] does not present better results than those shown in this paper, even with a time-limit greater than 42 h, and the Torres' model has never been tested with these instances before.

Table 1. Results for the mathematical formulations.

Instance	B&C		Torres		Chaves	
	Sol.	T(s)	Sol.	T(s)	Sol.	T(s)
v10	**1765**	**0.00**	**1765**	0.03	**1765**	0.03
v20	**2302**	**0.01**	**2302**	0.70	**2302**	1.22
v30a	**3582**	**0.00**	**3582**	2.66	**3582**	4.43
v30b	**2515**	**0.00**	**2515**	3.67	**2515**	6.41
v30c	**3236**	**0.03**	**3236**	12.63	**3236**	15.47
v50a	**4328**	**0.10**	**4328**	314.74	**4328**	422.65
v50b	**3872**	**0.12**	**3872**	375.30	**3872**	688.77
v100a	**6762**	**0.27**	7633	3600.00	7454	3600.00
v100b	**6760**	**0.17**	7290	3600.00	7668	3600.00
v250a	**14 083**	**12.67**	–	3600.00	–	3600.00
v250b	**13 632**	**8.35**	14 935	3600.00	14 715	3600.00
v500a	**25 848**	**102.06**	–	3600.00	–	3600.00
v500b	**26 389**	**130.17**	–	3600.00	–	3600.00

From Table 1, one realizes that the B&C approach outperforms the other exact approaches in the literature, both in solution quality and processing time. The formulation of Bienstock *et al.* [7] along with the proposed separation procedure, was able to solve all the 13 test problems proposed by Chaves *et al.* [8], proving optimality in six cases, in which the optimal solution was unknown until now. On the other hand, the formulations of the literature were able to solve only instances with up to 50 vertices, considering the time limit of one hour. Regarding the literature models, the Chaves' formulation required a higher CPU time than Torres', but in general, these two models presented similar performance.

5.2. Heuristics

After testing the formulations, experiments were conducted with the proposed heuristics. Despite the tabu search (TS) [19] reached better-quality solutions than CS* for a few instances, herein we consider CS* as the baseline heuristic for comparisons, since Pedro *et al.* [19] do not present the CPU time spent by their heuristic.

As the original source-code of CS* was unavailable and the experiments reported in Chaves *et al.* [8] have been made on a different processor, we followed the idea of Pinto *et al.* [20] by considering an approximate scale ratio to compare the speed of both processors. Concerning the single-rating performance, once the algorithms were tested on single-thread environments, the respective ratio for the two CPU models is $463/2184 \approx 0.21$, according to the PassMark benchmark [18].

A comparison between the proposed heuristics and CS* are presented in Table 2. The symbol "?" represents the lack of value reported in the literature for the respective instance, the best results are bold-faced, and the running times for CS* are those reported by Chaves *et al.* [8], multiplied by the scale factor 0.21.

From Table 2, one can observe that in terms of the best solution obtained, both RF and GD heuristics outperformed the CS* in all cases, and GD achieved an optimal solution for all of them, except for the instance v30c. In general, we can realize that the proposed heuristics were able to improve the solutions obtained by CS*, mainly for larger instances, in a shorter CPU time.

6. Conclusions

In this paper, we addressed the PCTSP which represents an extension of well-known TSP and very applicable in the real world. To handle such a problem, we have proposed some simple and easy-to-implement solution methods that additionally provide to be computationally feasible. As an exact approach, three formulations of the literature were implemented and tested. The formulation of Bienstock *et al.* [7] has an exponential number of

Table 2. Results for the heuristics (original running times of CS* multiplied by 0.21).

		CS*			RF		GD	
Inst.	OPT	Best Sol.	Avg. Sol.	Avg. T (s)	Sol.	T (s)	Sol.	T (s)
v10	1765	**1765**	1765.0	**0.01**	**1765**	**0.01**	**1765**	**0.01**
v20	2302	**2302**	2302.0	0.20	**2302**	**0.01**	**2302**	0.02
v30a	3582	**3582**	3591.0	0.22	**3582**	**0.01**	**3582**	0.02
v30b	2515	**2515**	2515.0	0.24	**2515**	**0.02**	**2515**	**0.02**
v30c	3236	**3236**	3241.0	0.25	**3236**	**0.04**	3245	**0.04**
v50a	4328	**4328**	4346.0	8.10	**4328**	0.14	**4328**	**0.10**
v50b	3872	**3872**	3881.0	7.87	**3872**	**0.06**	**3872**	0.11
v100a	6762	6832	6906.0	151.77	6764	**0.18**	**6762**	0.41
v100b	6760	6782	6844.0	?	6771	0.51	**6760**	**0.21**
v250a	14 083	15 162	15 284.0	245.21	**14 083**	**6.04**	**14 083**	7.79
v250b	13 632	14 078	14 176.0	?	13643	55.54	**13 632**	**3.67**
v500a	25 848	28 213	28 462.0	434.32	**25 848**	**60.50**	**25 848**	79.99
v500b	26 389	28 133	28 334.0	?	**26 389**	**9.12**	**26 389**	110.92

cut-set constraints, so a branch-and-cut algorithm along with a polynomial separation procedure was proposed. In terms of non-exact approaches, we devised two MIP-based heuristics, which make use of the linear relaxation of the problem along with bound and diving procedures.

The results of experiments with the instances of Chaves *et al.* [8] showed that the B&C algorithm solved all instances in a much shorter time than the others approaches from literature, and yet proved unprecedented optimality for some instances. Moreover, the literature formulations were unable to solve instances with more than 50 vertices.

Concerning the non-exact approaches, the results of our MIP heuristics outperformed the state-of-the-art heuristic, achieving the optimal solution for almost all cases. We highlight the guided diving heuristic, in which solved many instances faster than the B&C itself.

The B&C was able to satisfactorily solve all test-problems within a reasonable time (at most 130.17 s). However, we hypothesize that for larger instances, B&C might be incapable of finding optimal solutions due to machine memory issues. In this case, it would be interesting to apply the GD heuristic, which also handled the instances very well, but in a shorter CPU time.

We also believe our B&C has great potential to be hybridized with other heuristics to tackle more difficult problems. Hence, as future work, we intend to follow the hybridization idea of Clímaco *et al.* [10] and combine the B&C algorithm with some heuristic based on metaheuristic, such as genetic algorithms [15], GRASP [21] or tabu search [16].

References

[1] T. Achterberg and T. Berthold, Hybrid branching. In: International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems. Springer (2009) 309–311.

[2] B. Awerbuch, Y. Azar, A. Blum and S. Vempala, New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.* **28** (1998) 254–262.

[3] E. Balas, The prize collecting traveling salesman problem. *Networks* **19** (1989) 621–636.

[4] E. Balas, The Prize Collecting Traveling Salesman Problem and its Applications. Springer, Boston, MA (2007).

[5] T. Berthold, Primal heuristics for mixed integer programs (2006).

[6] T. Berthold, Heuristics of the branch-cut-and-price-framework scip. In: Operations Research Proceedings 2007. Springer (2008) 31–36.

[7] D. Bienstock, M.X. Goemans, D. Simchi-Levi and D. Williamson, A note on the prize collecting traveling salesman problem. *Math. Program.* **59** (1993) 413–420.

[8] A.A. Chaves and L.A.N. Lorena, Hybrid algorithms with detection of promising areas for the prize collecting travelling salesman problem. In: *Fifth International Conference on Hybrid Intelligent Systems (HIS'05). IEEE* (2005) 49–54.

[9] L. Ciupala and E. Ciurea, About preflow algorithms for the minimum flow problem. *WSEAS Trans. Comput. Res.* **3** (2008) 35–42.

[10] G. Clímaco, I. Rosseti, L. Simonetti and M. Guerine, Combining integer linear programming with a state-of-the-art heuristic for the 2-path network design problem. *Int. Trans. Oper. Res.* **26** (2018) 615–641.

[11] M. Dell'Amico, F. Maffioli, and A. Sciomachen, A lagrangian heuristic for the prize collectingtravelling salesman problem. *Ann. Oper. Res.* **81** (1998) 289–306.

[12] M. Fischetti and P. Toth, An additive approach for the optimal solution of the prize collecting traveling salesman problem. *Veh. Routing: Methods Stud.* **231** (1988) 319–343.

[13] M. Gendreau, A. Hertz and G. Laporte, New insertion and postoptimization procedures for the traveling salesman problem. *Oper. Res.* **40** (1992) 1086–1094.

[14] M.T. Goodrich and R. Tamassia, The christofides approximation algorithm. In: Algorithm Design and Applications. Wiley (2015) 513–514.

[15] O. Kramer, Genetic Algorithm Essentials. In Vol. 670 of *Studies in Computational Intelligence*. Springer (2017).

[16] M. Laguna, Tabu search. In: Handbook of Heuristics (2018) 741–758.

[17] N. Mladenović, D. Urosevic, S. Hanafi and A. Ilic, A general variable neighborhood search for the one-commodity pickup-and-delivery travelling salesman problem. *Eur. J. Oper. Res.* **220** (2012) 270–285.

[18] PassMark, CPU Benchmarks. Available from: https://www.cpubenchmark.net/compare.php (2019).

[19] O. Pedro, R. Saldanha and R. Camargo, A Tabu Search Approach for the Prize Collecting Traveling Salesman Problem. *Electron. Notes Discrete Math.* **41** (2013) 261–268.

[20] B.Q. Pinto, C.C. Ribeiro, I. Rosseti and A. Plastino, A biased random-key genetic algorithm for the maximum quasi-clique problem. *Eur. J. Oper. Res.* **271** (2018) 849–865.

[21] M.G.C. Resende and C.C. Ribeiro, Optimization by GRASP. Springer (2016).

[22] C.F.M. Toledo, M. da Silva Arantes, M.Y.B. Hossomi, P.M. França and K. Akartunalı, A relax-and-fix with fix-and-optimize heuristic applied to multi-level lot-sizing problems. *J. Heuristics* **21** (2015) 687–717.

[23] R.D. Torres and J.A.M. Brito, Problemas de coleta de prêmios seletiva. In: Simpósio Brasileiro de Pesquisa Operacional (SBPO) **35** (2003) 1359–1371.

[24] L.A. Wolsey and G.L. Nemhauser, Integer and Combinatorial Optimization. John Wiley & Sons (2014).