INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 30 (2023) 831–857
DOI: 10.1111/itor.12985

A GRASP/Path-Relinking algorithm for the traveling purchaser problem

Daniel Cuellar-Usaquén, Camilo Gomez and David Álvarez-Martínez*

Department of Industrial Engineering, Universidad de Los Andes, Carrera 1 Este #19A-40, Bogota, D.C., Colombia
E-mail: dh.cuellar@uniandes.edu.co [Cuellar-Usaquén]; gomez.ch@uniandes.edu.co [Gomez]; d.alvarezm@uniandes.edu.co [Álvarez-Martínez]

Received 29 February 2020; received in revised form 26 March 2021; accepted 28 March 2021

Abstract

The Traveling Purchaser Problem (TPP) is a generalization of the TSP that consists in choosing which nodes (markets) to visit to create a tour that allows to buy a set of products at minimum transportation and purchasing cost. The TPP has gained attention due to the computational challenges it poses and the potential applications it can support in today's technology-driven industry. This paper presents a GRASP-based methodology for the TPP based on three constructive procedures (*route-first*, *purchase-first*, and *purchase-and-route*) and two local search operators (*insert* and *remove*). The methodology is strengthened with a Path Relinking strategy to improve the GRASP performance by re-combining a set of elite solutions and with a Filtering strategy to improve the algorithm's efficiency by avoiding local search operations on the least promising solutions. The algorithm is tested with 855 instances of the asymmetric TPP and 190 instances of the symmetric TPP. Computational results prove the benefit of including the Path Relinking and Filtering strategies and suggest that the purchase-first constructive procedure is the most competitive in terms of objective function value with little extra effort in execution time with respect to the other constructive procedures. Our results outperform published results for the asymmetric TPP in a statistically significant way and show competitive performance for the symmetric TPP.

Keywords: GRASP; Path-Relinking; traveling purchaser problem

1. Introduction

The Traveling Purchaser Problem (TPP) considers a set of products to be purchased and a set of markets that offer the products at different prices. The objective is to find a tour that minimizes the cost of buying all the products and the cost of visiting the selected markets. Formally, consider a depot ($\{0\}$), a set M of markets, and a set K of products. The TPP can be defined on a graph

* Corresponding author.

$\mathcal{G} = (\mathcal{V}, \mathcal{A})$, where the set of nodes includes the depot and the markets (i.e., $\mathcal{V} := M \cup \{0\}$), and the set of arcs defines a complete graph (i.e., $\mathcal{A} := \{(i, j) : i, j \in \mathcal{V}, i \neq j\}$). Parameter p_{ik} is the price of buying product k from market i, whereas c_{ij} denotes the transportation cost between nodes i and j. The TPP can be classified into *symmetric* or *asymmetric* depending on the structure of the distance matrix (i.e., whether $c_{ij} = c_{ji}$). Additionally, the TPP can be classified as unrestricted when individual markets are assumed to suffice the demand of products; this is equivalent to considering a unit demand. The opposite case, known as the restricted TPP, implies that several markets may have to be visited to comply with demand constraints. In this paper, we focus on a solution algorithm for the Unrestricted Asymmetric TPP. However, for the sake of completeness, we include computational experiments for symmetric TPP instances as well. Further details about the TPP and its variants can be found in Manerba et al. (2017).

The TPP has received increasing attention in the scientific literature because of the challenge of combining decisions on which markets to visit, in which order, and from which market to buy each product. The TPP integrates features from procurement and routing problems and is known as a generalization of the Traveling Salesman Problem (TSP) and the Uncapacitated Facility Location Problem, thus, being an NP-hard problem (Manerba et al., 2017). Furthermore, the TPP was initially defined as a job scheduling problem (Burstall, 1966) and later re-interpreted as a routing problem (Ramesh, 1981). These similarities of the TPP with other classical problems expand its potential domains of application, thus, motivating specialized algorithms for the TPP.

The growing access to real-time data and ubiquitous computing capabilities has led to the widespread use of decision-support tools that create value opportunities for businesses and individuals. Two application cases motivate our work on the TPP. First, we devised a navigation tool for smart shopping, which helps solve trade-offs between purchase savings and their associated transportation costs in large, congested cities. A TPP-based application allows citizens to evaluate the savings obtained by visiting specific markets (e.g., outlet stores) in contrast with the costs incurred in terms of time, gas, and potentially other externalities (e.g., increased pollution and congestion). The proposed product is currently at a *proof-of-concept* stage precisely because we lacked an accurate and fast TPP solver to make the application viable in practice. As a second example, we have explored the TPP in collaboration with a leading Latin-American company in virtual marketplaces and delivery services. Their transition to offering products such as home appliances (in addition to restaurant products) creates a context in which the choice of retailer can generate savings in purchasing for both the company and its customers. However, the choice of retailer must also consider implications on routing, as travel times are critical to improving on-time delivery metrics and couriers' efficiency. The asymmetric version of the TPP is relevant in both cases as distance, but most importantly, travel time is different when driving in different directions in actual cities. Similarly, we adopt the unrestricted version of the TPP, as it is relatively safe to assume that retail stores have enough supply to satisfy the demand of individual citizens (both when purchasing directly or via a marketplace).

The TPP, with its integrated procurement and routing features, has the potential to impact the operation of companies in the current economy (e.g., Industry 4.0), as long as efficient solution algorithms become available. Several approaches have been proposed to solve the TPP (see Section 2 for an overview of the related work), but there is still a need for efficient algorithms that can pave the way for practical applications where solutions are needed within short time-frames and limited computational resources. For instance, although exact methods have been used to solve

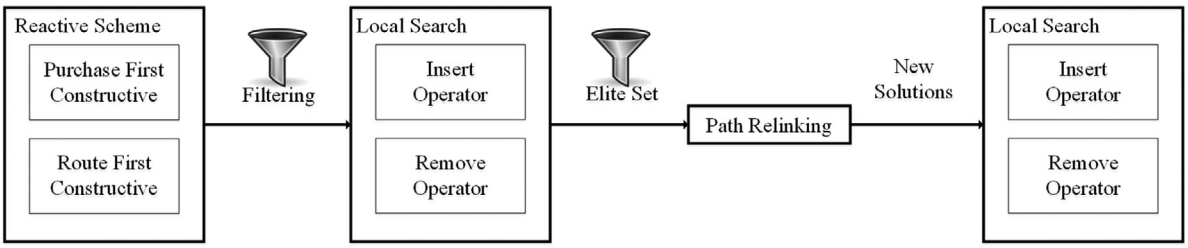

Fig. 1. Overall representation of the proposed methodology.

relatively large TPP instances, their computational efficiency is not suitable for real-time applications (TPP instances above 200 markets and 200 products may take between one and several days, according to our own runs and those reported by Bernardino and Paias (2018)). As a result, a variety of approximate methods have been proposed, with Bernardino and Paias (2018) being the latest work reporting benchmark results. Among existing approaches, Greedy Randomized Adaptive Search Procedures (GRASP) have been largely under-explored, with Ochi et al. (2001) being the only GRASP implementation for the TPP to the best of our knowledge. We see great potential in a GRASP implementation that takes advantage of the last two decades of research on constructive procedures, local search operators, and improvement strategies.

We propose a GRASP-based methodology with three constructive procedures (purchase-first, route-first, and purchase-and-route) and two local search operators (insert and remove), consistently with recent literature. We implement a reactive version of the GRASP that allows adjusting its greediness, depending on the solution quality it achieves. Furthermore, we include: (i) a filtering strategy to avoid efforts in local search for poorly performing candidate solutions; and (ii) a Path-Relinking strategy to further improve the quality of the GRASP by combining pairs of well-performing candidate solutions from an elite set and applying local search operators on the newly obtained solutions. Figure 1 summarizes the proposed methodology.

The main contribution of this work is the overall methodological framework rather than a specific building block. For instance, the use of Path Relinking on top of the GRASP proves relevant to obtain high-quality solutions, while filtering reduces the heavy computational effort imposed by such configuration. Similarly, the choice of simple yet effective local search operators is crucial for convergence. Sophisticated operators can be a burden on the already intensive local search procedure, without significant improvements in quality necessarily. Instead, we keep local search fast and focus on quality during Path Relinking. Finally, the adaptive adjustment of the GRASP's greediness provides a healthy balance between diversification and intensification. The latter tackles local minima, a pervasive issue in metaheuristics. Although these features exist in the literature, their coherent assembling into an efficient algorithm for the TPP is novel and can fit other problems. We believe the proposed framework can help practitioners solve real-world problems and bring under-explored solution architectures to the research community.

We test our methodology based on three sets of instances known as Class 6, Class 1, and Class 3 TPP instances. First, we use the Class 6 instances proposed by Singh and van Oudheusden (1997) for the unrestricted asymmetric TPP. We use these instances to test several variants of the proposed methodology. Specifically, we evaluate the performance of the three proposed constructive

procedures and the effect of including the filtering and path relinking strategies. Then, we compare our results with those of Bernardino and Paias (2018), the latest article reporting results for the Class 6 instances. The obtained solutions outperform published results with statistical significance, even under our algorithm's least competitive configuration. Second, and to test our algorithm's performance beyond the asymmetric TPP, we carry out comparisons for the Class 1 and Class 3 instances from Singh and van Oudheusden (1997), which correspond to symmetric versions of the TPP. We report comparisons with Bernardino and Paias (2018) and Goerler et al. (2013) for Class 1 instances, and with Riera-Ledesma and Salazar-González (2005), Bontoux and Feillet (2008), and Goldbarg et al. (2009) for Class 3 instances. The proposed algorithm performs well when applied to the symmetric TPP, achieving improved best-known solutions (BKS) and new optimal solutions for Class 1 instances, and matching several BKS for Class 3 instances (and obtaining a new one).

We provide a literature overview in Section 2, describe the proposed methodology in Section 3, and present computational results in Section 4. Section 5 concludes the paper and discusses opportunities for future work.

2. Related work

Exact and heuristic methods have been developed for the TPP since the 1970s. A recent literature review by Manerba et al. (2017) provides a comprehensive historical account of contributions and a classification of exact and heuristic approaches proposed for the TPP and its variants up to 2017. We will focus on recent contributions most closely related to our approach. For an in-depth discussion of the literature, see Manerba et al. (2017).

From the exact optimization perspective, one of the most notable contributions is the branch-and-cut approach by Laporte et al. (2003), which dynamically introduces sub-tour elimination constraints and includes the use of valid inequalities, allowing to solve instances of the symmetric TPP considering up to 200 markets and 200 products. Following the previous approach, Riera-Ledesma and Salazar-González (2006) proposed a branch-and-cut algorithm for the asymmetric version of the TPP. Other exact methods include constraint programming approaches as proposed by Cambazard and Penz (2012) for the symmetric TPP, and dynamic programming approaches as the one presented by Gouveia et al. (2011), which relies on a subgradient procedure and a Lagrangian greedy heuristic, achieving small gaps for instances of up to 300 markets. Exact methods, however, face limitations in practice, as they rely on specialized software, are computationally intensive, and often require long computation times to deliver feasible solutions (let alone optimal ones). Results from Bernardino and Paias (2018), as well as our own, indicate that medium-large instances may take between a day and a week to solve (on current-day computers), which is prohibitive for practical applications.

Given the NP-hard nature of the TPP, most attention has been given to approximate approaches. The most common constructive methods and local search operators are outlined below (for further discussion, see Manerba et al., 2017). Regarding constructive methods, Golden et al. (1981) proposed the Generalized Savings Heuristic (GSH), which starts with the market that offers more products at the lowest price and iteratively adds new markets based on the cheapest insertion that maximizes savings. Similarly, Ong (1982) proposed the Tour Reduction Heuristic, which starts with a tour over a subset of markets that can satisfy the demand of products, and removes markets

based on the transportation savings and potential increase in purchase cost. Pearn (1991) proposed the Commodity Adding Heuristic, which starts with a single-market tour for the first product and adds new products according to the convenience of purchasing them at markets already in the tour. The most common local search operators are based on adding (Add) and dropping (Drop) markets from a given tour (Voß, 1996a), as well as combining both moves (Drop-Add and Add-Drop) (Voß, 1996b). Some variations and additional moves have been introduced to enhance performance, including exchanges of markets (Boctor et al., 2003), randomized versions of the basic moves (e.g., Add-Random, Drop-Random), or adaptations from TSP heuristics (e.g., Add-Geni, Drop-Geni), as used by Ochi et al. (2001). More recent operators include l-Consecutive Exchange (Riera-Ledesma and Salazar-González, 2005); changing position of l-consecutive suppliers (Teeninga and Volgenant, 2004); and product purchasing exchange (Mansini et al., 2005).

Approximate approaches rely on the integration of constructive procedures and local search operators within a broader metaheuristic strategy. These approaches may be broadly summarized into those based on Tabu Search (El-Dean, 2008; Mansini et al., 2005; El-Dean, 2008), Simulated Annealing (Voß, 1996b), Variable Neighborhood Search (Ochi et al., 2001; Mansini et al., 2005), Ant Colony Optimization (Bontoux and Feillet, 2008) and Genetic Algorithms (Ochi et al., 1997; Goldbarg et al., 2009; Almeida et al., 2012). Ochi et al. (2001) propose two GRASP implementations: a GSH-based construction followed by either the Add-Random operator or by an Add-Geni operator. The local search comprised the Add, Drop, Add-Drop, Drop-Add, and Swap operators. The results in Ochi et al. (2001) demonstrate the potential of GRASP-based solution schemes, which motivates our exploration of a GRASP algorithm enhanced with filtering and path relinking strategies, as discussed below.

The following contributions are relevant for this work as they report results for TPP instances that are publicly available and will be used for comparisons in Section 4. Riera-Ledesma and Salazar-González (2005) use the nearest neighbor algorithm and the Lin–Kernighan heuristic (Lin and Kernighan, 1973) on all markets to construct an initial solution. Then, the local search comprised the insertion and the l-Consecutive Exchange operators. Finally, the authors apply perturbations to explore new solutions and then apply local search operators again. Bontoux and Feillet (2008) propose an Ant Colony Optimization approach that uses a dynamic update scheme for the number of ants and the amount of pheromones. The authors incorporate a so-called Anamorphic feature, which allows using different kinds of ants. The following search operators are applied to constructed solutions: 2-opt, insertion, simplification, drop, and dropstar. The dropstar operator is one of the main features of their work, as it uses dynamic programming to obtain the optimal set of nodes to be removed from the tour. Goldbarg et al. (2009) propose a transgenetic algorithm that starts by randomly adding markets to the tour until all purchases are feasible. The authors allow information exchange between individuals in their population, taking inspiration from the processes of endosymbiotic interactions and lateral gene transfer in nature. These processes are implemented based on agents called plasmids (which evaluates and applies best insertion moves) and transposons (which evaluates and applies best remove moves). Goerler et al. (2013) propose a Late Acceptance Hill Climbing algorithm in which the main feature is that comparisons between neighborhoods are delayed, which implies storing previous solutions into a fitness array. The authors obtain initial solutions using the nearest neighbor algorithm on all markets, followed by applying the Add, Drop, and l-consecutive Drop operators. Finally, Bernardino and Paias (2018) propose an approach based on genetic algorithms with three constructive procedures, namely: purchase-first,

route-first, and a third procedure that considers both. The authors use the Drop and Drop-Random operators and apply insertions when necessary to restore feasibility.

Metaheuristic approaches often rely on intensification strategies to enhance their performance. The Path Relinking procedure was proposed by Glover (1997) as an intensification strategy that examines new solutions based on an Elite set provided by a Tabu or Scatter search. The objective is to obtain promising solutions by taking an initial solution (x^i) and making it similar to a target solution (x^j) through sequential search moves. There are variants of the Path Relinking strategy depending on the construction and modification of the Elite set of solutions, the way in which the diversity of solutions is measured, or the movements used to make one solution similar to another. Each of these choices is problem-specific and affects the quality of solutions and computational performance of the algorithm. Examples of these variants may be found in Boudia et al. (2007), Barbalho et al. (2013), and Marzo and Ribeiro (2020). Further details are provided in Section 3, as we adopt the Path Relinking strategy as part of our solution approach.

3. Methodology

We propose a GRASP-based methodology (Section 3.1) that uses three constructive procedures (Section 3.2) and an improvement phase based on two local search operators (Section 3.3). The GRASP implementation is enhanced with reactive and filtering features (Sections 3.2 and 3.3, respectively), and complemented with a path relinking procedure (Section 3.4), as summarized in Fig. 1.

3.1. GRASP algorithm

Algorithm 1 describes the proposed GRASP algorithm, which implements construction and improvement phases for a number of iterations given by the parameter $TotalIter$ (line 1). In the construction phase, the algorithm uses the parameter $ConstructiveOption$ (line 2) to choose between three constructive procedures. The first constructive procedure prioritizes purchasing decisions (line 4), while the second one prioritizes routing decisions (line 6). The third option consists of randomly choosing one of the two previous constructive procedures (line 8), aiming to generate solutions that address both parts of the optimization objective (i.e., purchasing cost and transportation cost). All constructive procedures seek to generate pseudo-random candidate solutions, which are used to populate a Restricted Candidate List (RCL) with solutions sorted from best to worst. The degree of greediness of the algorithm is tuned by adjusting the size of the RCL, that is, by choosing how many of the top candidate solutions are considered in each iteration. Parameter A is a set of coefficients α_i indicating potential choices of RCL sizes (in percentage) while parameter N is the number of training iterations during which different coefficients α_i are tested (see Section 3.2 for details).

For each iteration, the algorithm verifies whether the constructed solution improves the best solution found so far (line 10), in which case the *incumbent* solution must be updated. The filtering strategy is then applied (line 11), meaning that the algorithm determines whether the current solution is promising enough to go into the improvement phase (i.e., it is better than a reference value, as discussed later). The improvement phase implements two search operators (insert and

Algorithm 1. GRASP + Filtering/Path Relinking

Parameters: $TotalIter$: total GRASP iterations, $ConstructiveOption$: flag to select construction method, A : set of α coefficients for the reactive feature, N : number of iterations for α training, $LocalIter$: local search iterations, $Tenure$: TABU duration, E : size of Elite set;
Input: List M : list of markets, **List** K : list of products, **List** p_{ik}: price of the product k from market i, **List** c_{ij}: transportation cost between nodes i and j;
Output: Solution *incumbent*;

1: **for** $i = 1$ **To** $TotalIter$ **do**
2: **switch** ($ConstructiveOption$)
3: **case** 1:
4: **Solution** $S \leftarrow ConstructivePurchase(A, N; M, K, p_{ik}, c_{ij}, i)$
5: **case** 2:
6: **Solution** $S \leftarrow ConstructiveRoute(A, N; M, K, p_{ik}, c_{ij}, i)$
7: **case** 3:
8: **Solution** $S \leftarrow SelectBetweenConstructivePurchaseOrRoute(A, N; M, K, p_{ik}, c_{ij}, i)$
9: **end switch**
10: **Update** *incumbent* **if** S **is a better solution**
11: **if** S **pass the filter then**
12: $Elite \leftarrow LocalSearch(LocalIter, Tenure, E; S.P, S.R, M, p_{ik}, c_{ij})$
13: **end if**
14: **end for**
15: $PathRelinking(LocalIter, Tenure, E; Elite, M, p_{ik}, c_{ij})$
16: **return** *incumbent*

remove) and uses a Tabu search scheme as a transition mechanism to avoid crossed applications of the two operators. At line 12, the *LocalSearch* function returns the updates applied on the $Elite$ set (which contains the most promising solutions) and the *incumbent* (implicitly). Note that solution S is decomposed into purchasing decisions $S.P$ and routing decisions $S.R$ before being sent to the local search procedure. At the end of the iterative process (lines 1–14), the solutions contained in the $Elite$ set are fed to the *PathRelinking* procedure (line 15), which explores new solutions by re-combining those in the $Elite$ set, aiming to improve the *incumbent* solution.

The filtering strategy (line 11 in Algorithm 1) consists of applying the improvement phase only if the solution obtained in the constructive phase is promising with respect to a reference value. Filtering is crucial for computational efficiency as it avoids applying local search operations on poorly performing solutions. Note that in a minimization context, promising solutions are those with objective function values *below* the reference value. The reference value is initialized with the first solution's value and is updated at any iteration in which a new solution reduces the reference value. If a solution's objective is not less or equal the reference value, the local search procedure is not applied, and a rejection counter ($nIter$) is incremented. As good solutions appear, the reference value becomes smaller (in minimization), leading to the rejection of many solutions and compromising diversity in the search. Therefore, when the counter $nIter$ reaches a pre-specified value, $maxFilter$, the reference value is increased following $reference = (1 + \lambda)reference$, where λ is set at 0.2 and $maxFilter = 5$, as reported in Martínez et al. (2015).

The described process requires the calibration of the seven parameters described in Algorithm 1, which will be discussed in depth in each of their corresponding sub-sections. The stopping criterion

Algorithm 2. Purchase-based constructive process

Parameters: A : set of alphas, N : number of iterations for α training;
Input: List M, K, p_{ik}, c_{ij}, i : Current iteration;
Output: Solution S : Solution with the purchase assignment P and market routing R;

1: **for** $k = 1$ **To** $|K|$ **do**
2: **List** $RCL \leftarrow BuildRCL(A, N; M, p_{ik}, k, i)$
3: **Market** $m \leftarrow SelectMarketRandomly(RCL)$
4: $P \leftarrow P \cup m$
5: **end for**
6: $R \leftarrow NearestNeighbourAlgorithm(P, c_{ij})$
7: **return** $S \leftarrow (P, R)$

for Algorithm 1 is twofold: first, the specified number of iterations for construction and improvement phases ($TotalIter$) determines when the GRASP must stop; and second, the size of the elite set (E) determines the number of solutions that must be combined during the Path Reliking procedure, thus, providing a stopping criterion for such stage. The GRASP algorithm receives the following inputs: the list of markets M, the list of products K, the price of products at each market p_{ik}, and the distance matrix describing transportation costs c_{ij} between pairs of nodes (i, j). The algorithm returns the best solution obtained throughout the process (i.e., the *incumbent*, in line 16).

3.2. Constructive phase

We consider three constructive procedures within the GRASP methodology: one that prioritizes purchasing decisions; one that prioritizes routing decisions; and one that randomly prioritizes purchasing decisions and routing decisions.

Algorithm 2 describes the *purchase-first* constructive procedure. This procedure cycles through the list of products K (line 1) and looks for a market in which each product $k \in K$ can be obtained. The choice of the market is pseudo-aleatory, based on a Restricted Candidate List (RCL) that is built as a greedy selection of the markets that offer the product (line 2); an element m is chosen randomly from the RCL (line 3), meaning that, in the current solution, product k will be obtained from market m. The list P contains the decisions on where to buy each product (line 4) and is iteratively populated (lines 1–5). Finally, the routing decisions (list R) are obtained by applying the nearest neighbor algorithm on the list P (line 6) (i.e., those markets from which at least one product is being purchased in the current solution). This constructive procedure returns an S solution, comprised the purchase decisions (P) and the routing decisions (R).

Algorithm 3 describes the *route-first* constructive procedure, which, in contrast to the latter, starts by generating a tour that visits all markets M using the nearest neighbor algorithm (line 1), thus, populating routing decisions first (list R). Then, the list P, containing purchasing decisions, is populated iteratively by creating an RCL for each product and randomly selecting a candidate from it (lines 2–6). Finally, the integration of purchasing and routing decisions is made by intersecting P and R (line 7), that is, taking the complete tour and removing those markets from which no products are purchased while preserving routing order. This constructive procedure returns a solution S (line 8) that integrates purchasing and routing decisions.

Algorithm 3. Route-based constructive process

Parameters: A : set of alphas, N : number of iterations for α training;
Input: List M, K, p_{ik}, c_{ij}, i : Current iteration;
Output: Solution S : Solution with the purchase assignment P and market routing R;

$R \leftarrow NearestNeighbourAlgorithm(M, c_{ij})$
for $k = 1$ **To** $|K|$ **do**
  List $RCL \leftarrow BuildRCL(A, N; M, p_{ik}, k, i)$
  Market $m \leftarrow SelectMarketRandomly(RCL)$
  $P \leftarrow P \cup m$
end for
$R \leftarrow R \cap P$ | **the route order is conserved**
return $S \leftarrow (P, R)$

Algorithm 4. Build RCL

Parameters: $A = \{\alpha_1, \alpha_2, \ldots, \alpha_m\}$: set of alphas, N : number of iterations for α training
Input: List M, p_{ik}, k, i : Current iteration;
Output: List RCL : restricted candidates list of markets for the product k;

$F_\alpha \leftarrow 0, \forall \alpha \in A$
$p_0 = p_\alpha \leftarrow 1/|A|, \forall \alpha \in A$
if $(i \leq N)$ **then**
  $\alpha^* \leftarrow$ **select randomly an alpha from** A **with a probabilty** p_0
else
  $\alpha^* \leftarrow$ **select randomly an alpha from** A **with a probability** p_α
end if
for $m \in M$ **do**
  if k **can be purchased in** m **then**
    $M^{aux} \leftarrow M^{aux} \cup m$
  end if
end for
sort M^{aux} **by lowest-price criteria**
for $i \leftarrow 1$ **To** $|M^{aux}|$ **do**
  if $i \leq \lceil \alpha^* \times |M^{aux}| \rceil$ **then**
    $RCL \leftarrow RCL \cup M_i^{aux}$
  end if
end for
$F_{\alpha^*} = F_{\alpha^*} + 1/q_{\alpha^*}$, **where** q_{α^*} **is the objective function obtained with** α^*
$p_\alpha \leftarrow F_\alpha / \sum_{i=1}^{m} F_i, \forall \alpha \in A$
return RCL
end

Finally, the third constructive procedure (labeled *P&R*) randomly chooses whether to execute Algorithm 2 or Algorithm 3, thus, generating solutions from both. Algorithms 2 and 3 rely on the function *BuildRCL* to construct the Restricted Candidate List.

The reactive feature of the GRASP algorithm is implemented within the *BuildRCL* function (Algorithm 4) and consists in adjusting the number of candidate markets in the *RCL*, allowing to obtain a balance between small RCL sizes (which may result in excessively greedy behavior) and

Algorithm 5. Local Search Heuristic

Parameters: $LocalIter$, $Tenure$, E;
Input: List P, R, M, p_{ik}, c_{ij};
Output: List $Elite$: Set of best solutions;
1: $TL_{insert} \leftarrow TL_{remove} \leftarrow \emptyset$
2: **for** $i = 1$ **To** $LocalIter$ **do**
3: **if** $(rnd() \leq 0, 5)$ **then**
4: $S, TL_{insert} \leftarrow InsertOperator(Tenure; P, R, TL_{remove}, M, p_{ik}, c_{ij})$
5: **else**
6: $S, TL_{remove} \leftarrow RemoveOperator(Tenure; P, R, TL_{insert}, p_{ik}, c_{ij})$
7: **end if**
8: $Decrease(TL_{remove}, TL_{insert})$
9: **Update** *incumbent* **if** S **is a better solution**
10: **Determine if** S **enters the** $Elite$ **not exceeding the size** E
11: **end for**
12: **return** $Elite$

large RCL sizes (which compromise efficiency). Algorithm 4 receives list A as an input that contains a pre-specified number of coefficients ($0 < \alpha_j \leq 1$; $j = 1, \ldots, |A|$), each of which defines one possible size for the RCL. For instance, an $\alpha_j = 0.3$ would lead to an RCL comprising 30% most promising markets. The objective of the reactive mechanism is to intensify the use of those α_j that yield better results. This is done by dynamically adjusting the probability of choosing each α_j as a function of its cumulative performance. All coefficients start with no record of performance (line 1) and, thus, equal probability of being chosen (line 2). The core of this strategy (lines 3–7) is to select one α_j based on a probability distribution p_α, which is adjusted depending on the performance obtained when using each α_j through iterations. Note that during the first N iterations, we use a uniform distribution (p_0) to choose α_j as few observations are available for the estimation of p_α. The list of markets (M) must be processed to identify which markets offer each product k (lines 8–12). The list M^{aux} stores the available markets and must be sorted in ascending order according to the purchasing price for product k (line 13). The RCL is constructed by selecting the first $\lceil \alpha_j \times |M^{aux}| \rceil$ elements from the ordered list of markets M^{aux} (lines 14–18). Then, whenever the GRASP uses a specific α^*, the objective function value q_{α^*} is used to update its cumulative performance F_{α^*} based on the reciprocal of q_{α^*} (line 19), as smaller objective functions imply better performance in minimization. After updating performance, the probabilities of choosing coefficients are normalized to fit a proper discrete probability distribution (line 20). Finally, the RCL is returned (line 21).

3.3. Local search

Algorithm 5 summarizes the local search procedure. The proposed local search operators consist of inserting or removing nodes in the tour (echoing traditional Add and Drop moves in the literature), followed by corresponding adjustments in purchase decisions. In both cases, we consider the current purchasing and routing decisions as inputs ($S.P$ and $S.R$). The local search operators use Tabu lists to keep track of movements that cannot be applied during a number of iterations specified by the

Algorithm 6. Insertion Operator

Parameters: $Tenure$;
Input: **List** P, R, TL_{remove}, M, p_{ik}, c_{ij};
Output: **Solution** S ; **List** TL_{insert};
1: $Candidates \leftarrow \emptyset$
2: **for** $i = 1$ **To** $|M|$ **do**
3: **if** ($i \notin TL_{remove}$ **and** $i \notin R$) **then**
4: $b_{insert}, pos_{insert} \leftarrow FindBestInsertion(i, R, c_{ij})$
5: $b_{purchase}, P_{aux} \leftarrow Purchase(R, i, p_{ik})$
6: $Saving \leftarrow b_{insert} + b_{purchase}$
7: $Candidates \leftarrow AddToCandidates(Saving, i, pos_{insert}, P_{aux})$
8: **end if**
9: **end for**
10: $Candidate \leftarrow SelectBestSaving(Candidates)$
11: $R \leftarrow$ **Insert** $Candidate.i$ **in** R **at** $Candidate.pos_{insert}$
12: $P \leftarrow Candidate.P_{aux}$
13: $S \leftarrow (P, R)$
14: $TL_{insert} \leftarrow AddToTabuListInsert(Candidate.i, Tenure)$
15: **return** S, TL_{insert}

input parameter $Tenure$. The Tabu list that keeps track of forbidden remove moves, TL_{remove}, is used within the insertion operator (to avoid opposite operations). In contrast, the Tabu list of banned insertion moves, TL_{insert}, is used within the removed operator. When the local search starts, both Tabu lists are empty (line 1). For a specified number of local search iterations $LocalIter$ (line 2), either of the two described operators is chosen with equal probability (lines 3–7). In each case, all insertions/deletions are evaluated, and the best one is applied. The *Decrease* function decreases the iteration counter for forbidden movements in the Tabu lists (line 8). The new solution S comprises the purchasing and routing decisions obtained by the chosen operator (these decisions are referred to as P and R within Algorithm 5). If the obtained solution S is better than the best one found so far, the *incumbent* is updated (line 9). The newly generated solution is then evaluated to determine whether it should be included in the $Elite$ set (line 10). The size of the $Elite$ set is controlled through parameter E. Finally, the local search procedure returns the $Elite$ set (line 12).

Algorithm 6 describes the insertion operator, which populates a list of candidate nodes that may be inserted and chooses to insert the one that yields the greatest savings in terms of both routing and purchasing costs. If no positive savings are achieved, we choose the move that damages the objective function the least. The $Candidates$ list is empty when the algorithm starts (line 1). For each market (line 2), the best possible insertion, pos_{insert}, is computed (line 4), along with the routing savings obtained from such insertion, b_{insert}. These are "negative savings" as routing costs increase with the insertion of a node (note that this holds only for instances satisfying the triangular inequality of the distance matrix). Then, purchasing decisions are updated greedily considering the possible insertion of market i (line 5); that is, any product that is cheaper at market i than at any market in the current tour R will now be purchased from i (obtaining a saving $b_{purchase}$ and updated purchasing decisions P_{aux}). The information on savings, best insertion, and updated purchasing decisions, is stored for each candidate (line 7), and the move yielding greater savings is applied (lines 10–13). The algorithm updates and returns the solution S (comprised purchasing and routing decisions, P

Algorithm 7. Remove operator

Parameters: $Tenure$;
Input: List P, R, TL_{insert}, p_{ik}, c_{ij};
Output: Solution S; **List** TL_{remove};

1: $Candidates \leftarrow \emptyset$
2: **for** $i \in R$ **do**
3: **if** $(i \notin TL_{insert})$ **then**
4: $b_{remove} \leftarrow CalculateRouteSavingRemove(i, R, c_{ij})$
5: $R_{aux} \leftarrow R \setminus \{i\}$
6: $b_{purchase}, P_{aux} \leftarrow Purchase(R_{aux}, p_{ik})$
7: **if** P_{aux} is feasible **then**
8: $Saving \leftarrow b_{remove} + b_{purchase}$
9: $Candidates \leftarrow AddToCandidates(Saving, i, P_{aux})$
10: **end if**
11: **end if**
12: **end for**
13: $Candidate \leftarrow SelectBestSaving(Candidates)$
14: $R \leftarrow$ **Remove** $Candidate.i$ **from** R
15: $P \leftarrow Candidate.P_{aux}$
16: $S \leftarrow (P, R)$
17: $TL_{remove} \leftarrow AddToTabuListRemove(Candidate.i, Tenure)$
18: **return** S, TL_{remove}

and R), as well as the Tabu list for insertion TL_{insert} (lines 13–15). Note that markets can only be inserted if they are not currently part of the tour and are not currently in the Tabu List of removing moves TL_{remove} (line 3).

Algorithm 7 describes the remove operator and works similarly to Algorithm 6, with the exception that removing a market might make the solution infeasible (as there may be no other markets offering certain products). For each market i currently in the tour R (line 2) and not in the Tabu list of insert moves TL_{insert} (line 3), the algorithm computes the routing savings of removing the market (line 4). Then, routing decisions are updated R_{aux} (line 5). The removal of a market imposes the need to find a market to purchase the products previously obtained at market i (line 6). If all products can be obtained after removing market i (line 7), the savings and candidate list are updated (lines 8 and 9). Finally, the best remove move is chosen (line 13), and the operation is implemented by updating routing and purchasing decisions (lines 14–16), along with the Tabu list for remove moves (line 17). If no positive savings are achieved, we choose the move that damages the objective function the least. This operator returns the new solution S and the remove Tabu list (line 18).

3.4. Path-Relinking

Algorithm 1 outlines the overall structure of the GRASP procedure, which populates routing and purchasing decisions ($S.R$ and $S.P$) via the constructive phase, as described in Section 3.2. Then, the Filtering strategy is applied, meaning that if the objective function achieved by $S.R$ and $S.P$ is within an acceptable range from the best-known objective value generated by the construction

Algorithm 8. Path Relinking

Parameters: $LocalIter$, $Tenure$, E;
Input: **List** $Elite$, M, p_{ik}, c_{ij};
1: **for** $k = 1$ **To** E **do**
2: **Select randomly** S^i **and** S^j **from the** $Elite$ **by tournament**
3: **while** $S^i.R \neq S^j.R$ **do**
4: $S^i.R \leftarrow$ **make** $S^i.R$ **one node similar to** $S^j.R$
5: $S^i.P \leftarrow Purchase(S^i.R, p_{ik})$
6: **if** $S^i.P$ is feasible **then**
7: $TL_{insert} \leftarrow TL_{remove} \leftarrow \emptyset$
8: $S^{aux} \leftarrow LocalSearch(LocalIter, Tenure, 1; S^i.P, S^i.R, TL_{insert}, TL_{remove}, M, p_{ik}, c_{ij})$
9: **Update** *incumbent* **if** S^{aux} **is a better solution**
10: **end if**
11: **end while**
12: **end for**

process, the local search procedure is applied, as discussed in Section 3.3; the filtering process, thus, avoids the computational effort of local search on nonpromising solutions. If any solution obtained during the local search is better than the best one achieved so far, then the incumbent is updated. Finally, obtained solutions may enter the elite set to be considered in the subsequent improvement step: the Path Relinking strategy. A solution can only be included in the elite set if it outperforms an existing solution in the set and if its routing decisions are different to those already in the set.

The idea of Path-Relinking is to exploit the features of the best performing solutions in the $Elite$ set and combine them into new solutions that can potentially outperform the previous ones, aided by additional local search moves. Algorithm 8 describes the proposed Path-Relinking strategy, which focuses on the $Elite$ set that contains the best performing solutions obtained from the GRASP. A pair of elite routing solutions ($S^i.R$ and $S^j.R$) are chosen via tournament (line 2). The core idea is to obtain new solutions $S^i.R$ by making $S^i.R$ similar to $S^j.R$, including one node at a time until both solutions are equal (line 4). As the set of markets is modified, purchasing decisions $S^i.P$ must be updated according to the new markets in $S^i.R$ (line 5). Note that it is necessary to verify that all products can be obtained from the markets in the updated set $S^i.R$ (lines 8 and 9). A new solution $S^i.P$ may end in infeasibility (line 6), in which case a new attempt is made (line 4). If, on the other hand, $S^i.P$ is feasible, the local search procedure is applied, returning the best solution (S^{aux}) found in the search procedure (lines 7 and 8); if S^{aux} outperforms the *incumbent*, the latter must be updated (line 9). The Path Relinking strategy is applied E times (i.e., $|Elite|$) (line 1).

In this work, the neighborhood movements to obtain new solutions from the $Elite$ set focus on the problem's routing component, as reported in previous research (Campos et al., 2014; Morán-Mirabal et al., 2014; Ho and Szeto, 2016); that is, the procedure makes a solution's route one node similar to a target solution. The Path Relinking strategy can be hybridized with different metaheuristics, with the GRASP procedure being successfully implemented in recent research (Resendel and Ribeiro, 2005; Ribeiro and Resende, 2012). The performance improvement obtained with the Path Relinking procedure responds to its multi-start feature and its ease for parameterization, allowing relinking to explore diverse, high-quality solutions; without the multi-start feature, each iteration generates independent solutions without taking advantage of the promising features of

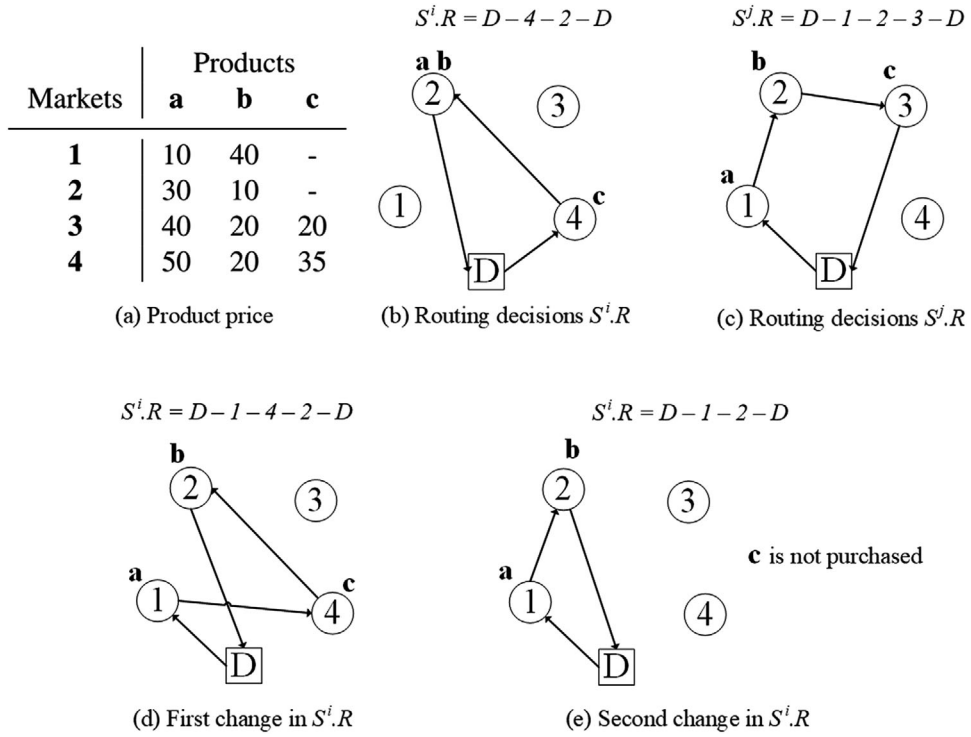

Fig. 2. Illustrative example of the Path Relinking procedure. (a) Product price. (b) Routing decisions $S^i.R$. (c) Routing decisions $S^j.R$. (d) First change in $S^i.R$. (e) Second change in $S^i.R$.

each solution (Festa and Resende, 2013). We propose a solution scheme that uses GRASP followed by Path Relinking, applying a second round of local search movements on the latter's solutions.

Figure 2 illustrates line 4 of Algorithm 8, namely: the process of making the routing decisions in solution $S^i.R$ similar to those of solution $S^j.R$. Consider the set of markets $M \in \{1, 2, 3, 4\}$, and the set of products $K \in \{a, b, c\}$, along with their corresponding purchasing prices. Figure 2b and c show solutions $S^i.R$ and $S^j.R$, respectively, which are chosen randomly from the $Elite$ set. The process of making solution i similar to solution j is performed one node at a time so that in the first step, $S^i.R$ changes to the solution shown in Fig. 2d. In a subsequent step, the solution will change to the one shown in Fig. 2e, which turns out to be infeasible, as product c cannot be obtained from the markets in the route. The subsequent and final step will make $S^i.R$ equal to $S^j.R$.

4. Computational results

This section presents two types of computational experiments. In Section 4.1, we use a well-known set of instances to compare the performance of a group of variants of the proposed methodology in order to determine the best configuration and parameter tuning. In Section 4.2, we compare the two best-performing versions of our algorithm against the best-reported solutions in the literature in

Table 1
Summary of characteristics of the sets of instances and the authors cited in comparisons

Section	Instance description	Author	Solution approach	Implementation details
3.1	Class 6 - UATPP - 855 instances $\|M\| \in [10, 300]$, $\|K\| \in [10, 200]$	This work	GRASP+Filtering\PR	C++. AMD Ryzen 7-3800X @ 3.9GHz Windows 10 with 32 GB RAM
3.2.1	Class 6 - UATPP - 75 instances $\|M\| \in [50, 300]$, $\|K\| \in [50, 200]$	Bernardino et al. (2018)	Genetic Algorithm	MATLAB R2014b. Intel Core i7 - 4790 @ 3.6GHz with 8 GB RAM
3.2.2.1	Class 1 - USTPP - 50 instances $\|M\| = 33$, $\|K\| \in [50, 500]$	Bernardino et al. (2018)	Genetic Algorithm	MATLAB R2014b. Intel Core i7 - 4790 @ 3.6GHz with 8 GB RAM
	- 25 closed instances	Goerler et al. (2013)	Late Acceptance Hill Climbing Algorithm	Java. Intel Core @ 2.50GHz with 8GB RAM
	- 25 open instances			
3.2.2.2	Class 3 - USTPP - 140 instances $\|M\| \in [50, 350]$, $\|K\| \in [50, 200]$	Riera-Ledesma et al. (2005)	Local Search	Celeron @ 500MHz
	- 89 closed instances	Bontoux et al. (2008)	Ant Colony Algorithm	C++. Pentium4 @ 2GHz Linux/Debian
	- 51 open instances	Goldbarg et al. (2009)	Transgenetic Algorithm	Pentium4 @ 2.8 GHz Ubuntu Linux with 512 MB of RAM

order to demonstrate the potential of the proposed methodology. The main instances we use in the experiments come from the Class 6 Asymmetric Instances without Capacities for the TPP, proposed by Singh and van Oudheusden (1997) and available at http://webpages.ull.es/users/jriera/TPP.htm. This set contains 855 randomly generated instances for the asymmetric TPP, with sizes ranging from 10 markets and 10 products to 300 markets and 200 products. We use Class 6 instances in Section 4.1 for calibration and in Section 4.2.1 as the main comparison for the class of problem we aim to solve (the asymmetric TPP). In Section 4.2.2, we extend the experiments to the symmetric TPP to test the versatility of our algorithm. In this case, we use the symmetric TPP instances known as Class 1 and Class 3 instances (Laporte et al., 2003), which are available at http://webpages.ull.es/users/jriera/TPP.htm. Table 1 provides a summary of the characteristics of the instances used in the computational experiments, as well as the methods and computer specifications of the authors cited in comparisons.

In order to measure the quality of solutions against optimal objective function values, we implemented the Mixed Integer Programming (MIP) formulation of the TPP, as presented in Manerba et al. (2017), using the sub-tour elimination constraints proposed by Miller et al. (1960). For all our experiments, the stopping criterion is given by the total number of iterations ($TotalIter$) and the size of the elite set (E), as discussed in Section 3.1. We compute percent optimality gaps for our solutions throughout experiments with respect to the optimal solutions (when known), as shown in

Equation (1). All gap values are expressed as 0–100 percentages. For instance, a value of 0.07 in the tables corresponds to a gap of 0.07%. All reported gaps are average gaps over the set of replicates.

$$Gap(\%) = 100 * \frac{(Incumbent - Optimum)}{Optimum}. \quad (1)$$

4.1. Selecting the best variants of the proposed methodology

We propose a set of experiments to calibrate our algorithm using the subset of Class 6 instances considered in Bernardino and Paias (2018). These instances were solved to optimality, except for the one with $|M| = 300$, $|K| = 200$, and $Id = 1$, for which we report the best objective value obtained before termination due to memory crash.

First, we seek to determine the impact of including the Filtering and Path Relinking strategies in contrast to running the simple GRASP algorithm. Consequently, we compare the computational times and objective function values for four variants of the proposed methodology: the reactive GRASP (labeled as GRASP in results tables); the reactive GRASP improved with Path Relinking (labeled as GRASP\PR); the reactive GRASP improved with Filtering (labeled as GRASP+Filtering); and the reactive GRASP with Filtering and Path Relinking (labeled as GRASP+Filtering\PR). For this comparison, the parameters were tuned as $TotalIter = 4500$, $A = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, $N = 0.35$, $LocalIter = 3850$, $Tenure = 5$, and $E = 20$, as a result of experimental analysis. Because of our methodology's randomized features, each instance was run five times with different random seeds. This number of replicates responds to the number reported in previous works to provide a fair comparison.

Figure 3 presents 95% confidence intervals for the optimality gaps and computation times of the proposed variants of our algorithm. The results presented in Fig. 3 aggregate all constructive procedures for each displayed variant. The best objective values are achieved by the GRASP\PR and GRASP+Filtering\PR approaches, with average gaps of 0.12% and 0.22%, respectively. This result confirms the capabilities of the Path Relinking strategy to improve the solutions obtained within the GRASP procedure. However, in terms of computation time, the GRASP+Filtering\PR outperforms the GRASP\PR by an average of 31.53 seconds, suggesting that the Filtering strategy can improve performance while preserving the quality of the objective values. Following the results in Fig. 3, we select the GRASP+Filtering\PR variant of our algorithm to perform comparisons with previous works, as it balances solution quality and computational efficiency.

The total number of iterations $TotalIter$ has a significant impact on computational time and solution quality. We compare the chosen GRASP+Filtering\PR variant of our methodology for different numbers of total iterations ($TotalIter$). Figure 4 shows 95% confidence intervals for the GRASP algorithm with Path Relinking and Filtering, considering 2500, 4500, and 9000 total iterations. When increasing iterations from 2500 to 4500, the quality of solutions improves by an average of 29% without increasing computation times. This occurs because when using fewer iterations, the training process for selecting coefficients α_i (i.e., for tuning greediness) may not have sufficient information. Because of such reduced training, there is no convergence in the selection of adequate coefficients α_i. As a result, obtained solutions are highly heterogeneous, making the node-by-node comparisons of the Path Relinking procedure computationally expensive.

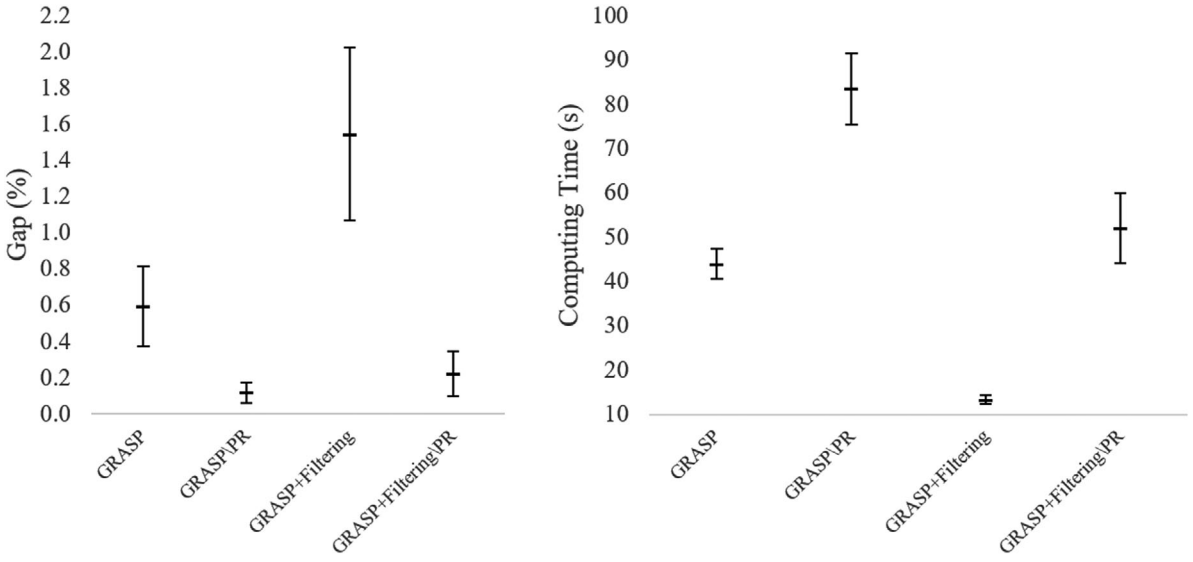

Fig. 3. Ninety-five percent confidence interval for optimality gaps and computational time (in seconds) between variations of the proposed approach.

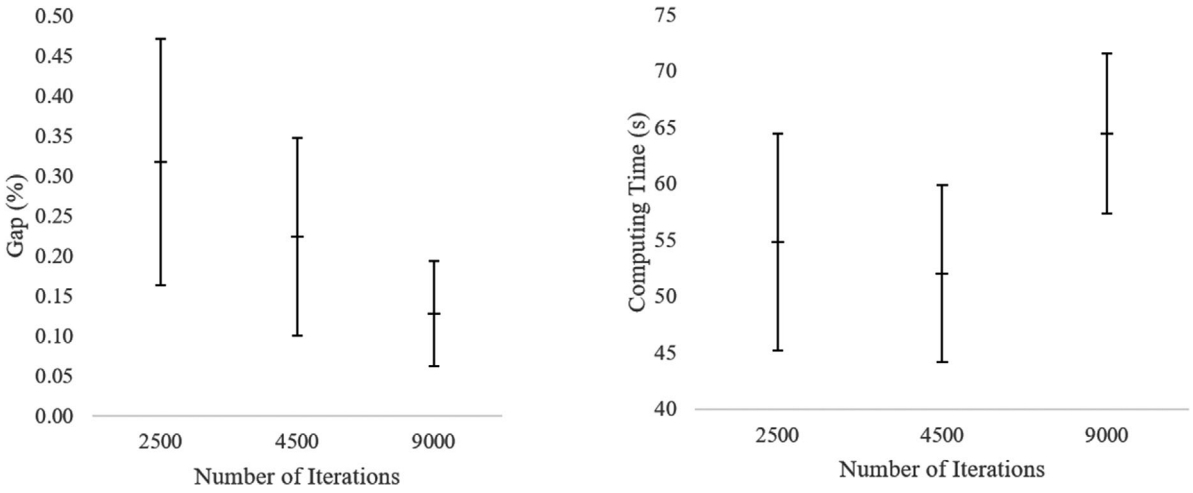

Fig. 4. Ninety-five percent confidence interval for optimality gaps and computational time (in seconds) for the GRASP algorithm with filtering and path-relinking strategies considering different number of total iterations.

Conversely, when increasing iterations from 4500 to 9000, the quality of solutions improves by 43% on average, but computational time is increased by 12.42 seconds. Based on these results, we choose the GRASP+Filtering\PR with 4500 total iterations for the comparisons with other authors (Section 4.2) due to its adequate balance between solution quality and computation time.

Finally, in Fig. 5, we compare the optimality gaps and computational times obtained for the three proposed constructive procedures. The *purchase-first* and *route-first* procedures show similar

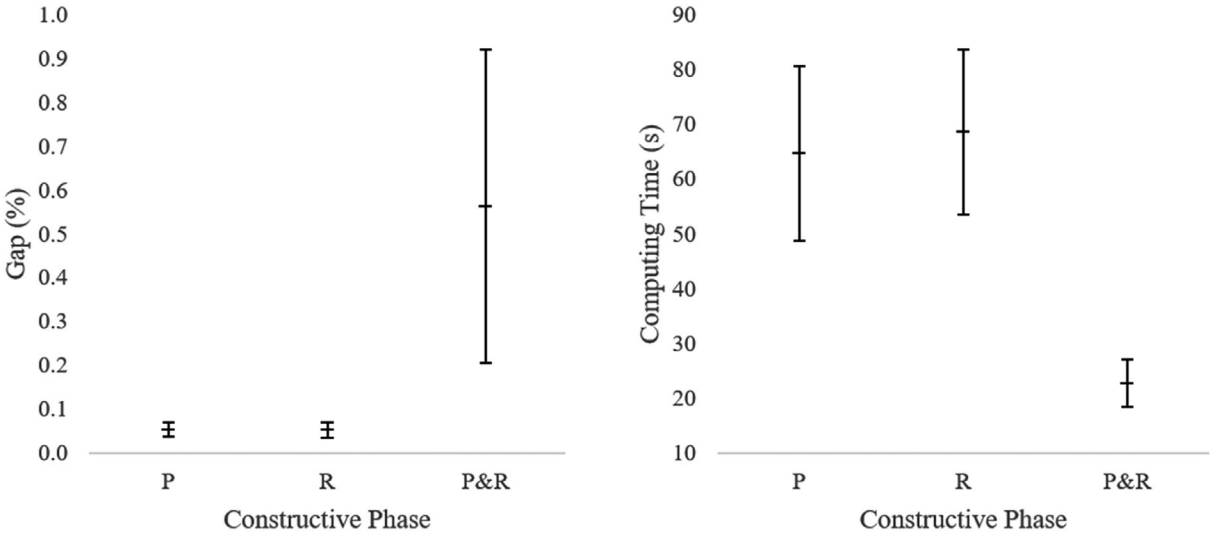

Fig. 5. Ninety-five percent confidence interval for the optimality gaps and the calculation time (in seconds) of the different variations in the constructive phase of the proposed approach.

results in terms of gap and computation time, and both significantly outperform the third constructive procedure (P&R), which randomly alternate the two mentioned procedures. Consequently, the purchase-focused and route-focused procedures will be taken into account for comparisons with other authors (Section 4.2). The main reason for the poor performance of the randomly alternate approach (P&R) is that the number of iterations is split between the purchase-first and route-first procedures. As a result, solutions do not get to improve as much as in the pure purchase- or route-based procedures. Experimental results indicate that the approach *P&R* produces homogeneous elite sets, thus, leading to short computational times due to a small number of combination steps in the Path Relinking procedure. Furthermore, the training of α coefficients (i.e., reactive sizes of RCLs) is also affected negatively by the split number of iterations. Hence, an adequate level of greediness may not be obtained for the reactive feature of the GRASP. The good performance of *P&R* in terms of computation time may be explained by the fact that, as solutions tend not to improve as much, many solutions end up being filtered, thus, avoiding local search computations often.

The computational complexity of the proposed algorithm is given by the number of markets ($|M|$) and products ($|K|$), along with parameters $TotalIter$, $LocalIter$, and E, which limit the number of iterations at each of the stages of the algorithm, as discussed in Section 3. Let us consider two cases. First, if $|M| >> |K|$, the complexity for the construction and improvement phases would be $O(TotalIter * LocalIter * |M|^2)$ while the path relinking strategy would have complexity $O(E * LocalIter * |M|^3)$. Thus, the complexity of the GRASP+Filtering\PR depends on the values of $TotalIter$ and $E * |M|$. For example, in cases in which $E * |M| >> TotalIter$, the computational time is dominated by the Path Relinking strategy, whereas in the opposite case most time is consumed in the construction and improvement stages. The second case, in which $|K| >> |M|$, leads to a complexity $O(TotalIter * LocalIter * |M| * |K|)$ for the construction and improvement

stages and a complexity O($E * LocalIter * |M|^2 * |K|$) for the Path Relinking. As before, the overall complexity of the algorithm depends on $TotalIter$ and $E * |M|$.

4.2. Benchmarking against the best previous works

4.2.1. Comparison for the asymmetric TPP

We propose a comparison of our methodology against the one proposed by Bernardino and Paias (2018), as they provide the latest reported results for the Class 6 TPP instances. We compare the two best-performing variants of our algorithm with the three solution approaches proposed in Bernardino and Paias (2018), namely, their purchase-focused approach (MH-P), their route-focused approach (MH-R), and their combined approach (MH-C). Although our general solution scheme is similar to the one in Bernardino and Paias (2018), a key difference lies in the constructive phase. Our constructive phase specializes in building initial solutions that guarantee feasibility and pursue a reasonably good objective based on knowledge of the problem's structure. That has a significant impact, as local search iterations and improvement strategies focus their effort (mostly) on high-quality solutions. In contrast, the genetic approach in Bernardino and Paias (2018) uses a population in which individuals' quality is not controlled. Furthermore, the adaptive and filtering features within the GRASP and the subsequent path relinking strategy improve the performance significantly, as evidenced by the results below.

In this section, we present results for instances with $|M| \in \{50, 75, 100, 200, 300\}$ and $|K| \in \{50, 100, 200\}$, which correspond to the 75 instances for which Bernardino and Paias (2018) report solutions. For the remaining 780 instances of Class 6, we make our results available on http://opentechs.co/projects or by request. The proposed algorithms were executed five times, providing the same number of replicates used in Bernardino and Paias (2018). The quality of solutions is measured in terms of their optimality gap, as shown in Equation (1). See Table 1 for details about the instances and technical specifications of the comparisons.

Table 2 presents a comparison between our results and the ones reported in Bernardino and Paias (2018) in terms of optimality gaps and computational time. In the table, instances are grouped by their size (number of markets and products). The reported values correspond to averages of all instances of the indicated sizes, including the five replicates considered for each. The two algorithms we propose are superior in terms of solution quality, achieving an average optimality gap of 0.05%, in contrast to values of 0.58% and above by the MH-P, MH-C, and MH-R strategies. Bernardino and Paias (2018) reports that the MH-R algorithm obtains optimal solutions for 10 out of the 75 reported instances, while the MH-P and MH-C approaches only obtain optimal solutions in eight instances, achieving optimality in 13 out of the 75 instances in total. In contrast, each of the two variants of our methodology obtains optimal solutions in 49 out of the 75 instances, achieving optimality in 53 out of the 75 instances. The solution approaches proposed in this paper and those proposed by Bernardino and Paias (2018) exhibit reasonable results in terms of computational time, but the reported times are not directly comparable due to the differences in programming language and computing architectures.

Figure 6 shows 95% confidence intervals for the average gap of solutions according to the number of markets and the number of products for the set of instances. Results do not evidence sig-

Table 2
Comparison of optimality gaps and computational time (in seconds) between the proposed approach and the work of Bernardino and Paias (2018)

		Gap (%)					Computing Time (s)				
		Bernardino et al. (2018)			GRASP + Filtering/PR		Bernardino et al. (2018)			GRASP + Filtering/PR	
		MH-P	MH-R	MH-C	P	R	MH-P	MH-R	MH-C	P	R
|**M**|	50	0.23	0.19	0.21	**0.03**	0.04	122	60	94	33	39
	75	0.37	0.35	0.36	**0.04**	**0.04**	157	81	118	48	58
	100	0.37	0.33	0.36	**0.03**	0.04	180	95	135	50	62
	200	0.89	0.89	0.90	**0.06**	**0.06**	296	142	201	71	71
	300	1.16	1.12	1.14	0.11	**0.09**	417	183	268	122	113
|**K**|	50	0.91	0.87	0.91	**0.01**	**0.01**	120	57	78	19	21
	100	0.49	0.46	0.46	**0.03**	**0.03**	218	108	150	42	44
	200	0.41	0.40	0.41	**0.12**	**0.12**	365	172	262	133	141
Average		0.60	0.58	0.60	**0.05**	**0.05**	234	112	163	65	69

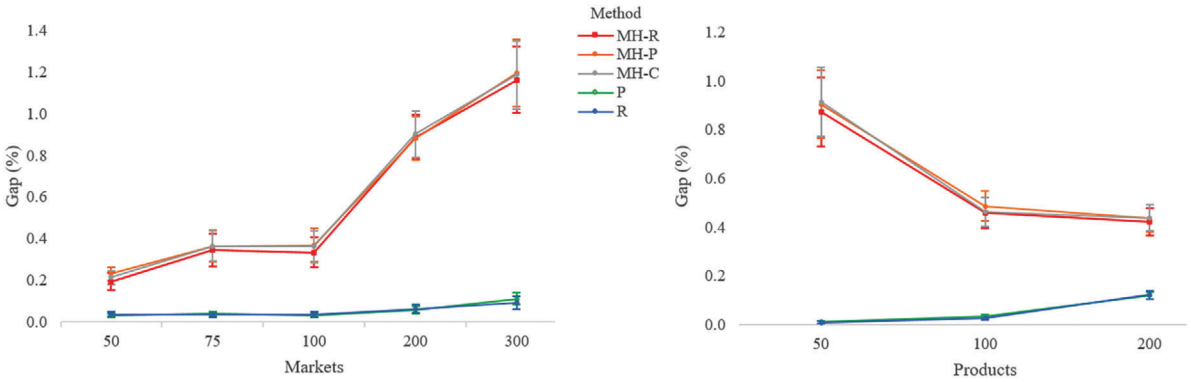

Fig. 6. Ninety-five percent confidence interval for optimality gaps for the Bernardino and Paias (2018) approaches (MH-R, MH-P, MH-C) and for the proposed GRASP algorithm with filtering and path-relinking approaches (Purchase - P, Route - R).

nificant differences between the three variants of the methodology proposed by Bernardino and Paias (2018). The average gap for the MH-R, MH-P, and MH-C algorithms remains stable (within a 0.19% and 0.37% range) for instances with fewer than 100 markets, and increases considerably (to 0.88% to 1.2%) when increasing to 200 and 300 markets. In contrast, the algorithms proposed in this work show a stable optimality gap (in the 0.03–0.11% range) when increasing the number of markets. Regarding variations with respect to the number of products, the MH-R, MH-P, and MH-C algorithms (Bernardino and Paias, 2018) achieve their worst performance for instances with 50 products (with gaps between 0.87% and 0.92%) but obtain improvements to gaps slightly below 0.5% when increasing the number of products. In contrast, the algorithms proposed in this work show an average gap between 0.01% and 0.12%, which represents a significant difference with respect to the results in Bernardino and Paias (2018).

Table 3
Comparison of gaps between the proposed approach and the latest reported results (Goerler et al., 2013) for the Class 1 instances that have been solved to optimality

		Goerler et al. (2013)	GRASP + Filtering/PR			
			P		R	
|M|	|K|	Gap %	Gap %	T (s)	Gap %	T (s)
33	50	**0.00**	**0.00**	20	**0.00**	20
33	100	0.82	**0.00**	39	**0.00**	42
33	150	1.43	**0.00**	68	**0.00**	70
33	200	1.96	**0.00**	92	**0.00**	97
33	250	2.67	**0.21**	128	0.22	131
Average		1.37	**0.04**	69	**0.04**	72

4.2.2. Comparison for the symmetric traveler purchaser problem

In order to test the performance of the proposed algorithms beyond the Class 6 asymmetric TPP instances, we propose additional experiments using the Class 1 and Class 3 instances introduced above for the symmetric TPP. As before, each instance will be run five times, matching the number of replicates in previous works. See Table 1 for details about the instances and technical specifications of the comparisons.

Class 1 instances. In Class 1 instances, the number of markets is fixed to 33 and there are five instances for each size of the product set $|K| = 50, 100, 150, 200, 250, 300, 350, 400, 450, 500$. Laporte et al. (2003) solved instances of up to 250 products to optimality using a branch-and-cut approach; these instances are referred to as closed Class 1 instances, in opposition to the open ones, for which optimal values are not reported.

In this section, we compare our results against those reported in the following works: first, the Late Acceptance Hill Climbing algorithm by Goerler et al. (2013); and second, the genetic algorithm methodology in Bernardino and Paias (2018). Both works provide new BKS for open Class 1 instances. The quality of our solutions in closed instances will be measured in terms of optimality gap based on Equation (1), whereas in open instances, we compare the best solution obtained with our methodology against the best-known results to date.

Table 3 shows the average gap for the two variants of the proposed methodology (considering five replicates per instance) along with the best result reported by Goerler et al. (2013) (with only one execution per instance) for instances below 250 products with known optimal solutions. The two variants of our GRASP-based methodology (i.e., with P and R constructive procedures) exhibit similar performance, with 0% gaps for all instances, except for those with 250 products, in which the P and R procedures achieve gaps of 0.21% and 0,22%, respectively. The two proposed methodologies outperform the results in Goerler et al. (2013) with an average difference of 1.33% in the optimality gap and with superior results in all instances with more than 50 products. The algorithm in Goerler et al. (2013) achieves optimality in 5 out of the 25 instances, whereas each of our algorithms obtains 19 optimal solutions.

Table 4 shows the best solution obtained by Goerler et al. (2013), by Bernardino and Paias (2018), and by each variant of the methodology proposed in this work for instances with more

Table 4
Comparison of the objective function value between the proposed approach and the latest reported results (Goerler et al., 2013; Bernardino and Paias, 2018) for the Class 1 instances that had not been solved to optimality. Values in bold correspond to BKSs and values with asterisks correspond to optimal solutions

			Objective function value			
			Goerler	Bernardino	GRASP + Filtering/PR	
|M|	|K|	Id	et al. (2013)	et al. (2018)	P	R
33	300	1	14,346	14,206	**13,883***	**13,883***
33	300	2	14,710	14,270	**13,954***	**13,954***
33	300	3	14,354	14,207	**13,995**	14,020
33	300	4	14,189	14,118	**13,765***	**13,765***
33	300	5	14,329	14,407	**14,281**	**14,281**
33	350	1	15,901	15,872	**15,306***	**15,306***
33	350	2	14,665	14,689	**14,324***	**14,324***
33	350	3	16,108	16,195	**15,858***	**15,858***
33	350	4	16,023	15,777	**15,550***	**15,550***
33	350	5	16,118	15,966	15,816	**15,810**
33	400	1	17,790	17,467	17,097	**17,084***
33	400	2	16,832	16,336	16,158	**16,116***
33	400	3	17,335	17,168	**16,903**	**16,903**
33	400	4	17,313	17,526	**17,149***	**17,149***
33	400	5	17,825	17,675	**17,162***	**17,162***
33	450	1	18,140	18,493	**17,860**	**17,860**
33	450	2	18,097	17,564	**17,329***	**17,329***
33	450	3	18,555	18,762	**18,203***	**18,203***
33	450	4	17,819	17,700	17,530	**17,524**
33	450	5	18,795	18,931	**18,424***	18,428
33	500	1	19,821	19,979	19,273	**19,270***
33	500	2	19,604	19,721	**19,310***	**19,310***
33	500	3	19,774	19,814	19,399	**19,376***
33	500	4	19,863	19,591	**19,300***	**19,300***
33	500	5	19,497	19,573	**18,956***	**18,956***
Total			427,803	426,007	168,312	**114,826**

than 250 products without known optimal solutions (i.e., open instances). The results in Goerler et al. (2013) correspond to one run per instance, whereas those in Bernardino and Paias (2018) and our methodology correspond to five runs per instance. The two GRASP-based algorithms, with purchase (P) and routing (R) constructive procedures, achieve new BKS for all instances. As there are no optimal results reported for these instances, we ran our MIP implementation of the TPP as explained above to verify optimality and compute gaps. The solutions denoted with an asterisk in Table 4 correspond to optimal objectives. Our algorithm achieves 19 optimal solutions out of the 25 open instances.

Class 3 instances. Class 3 instances, proposed by Riera-Ledesma and Salazar-González (2005), also correspond to the symmetric version of the TPP and are commonly known as

Table 5
Comparison of gaps between the proposed approach and the latest reported results (Riera-Ledesma and Salazar-González, 2006; Bontoux and Feillet, 2008; Goldbarg et al., 2009) for the Class 3 instances that have been solved to optimality

		Riera-Ledesma et al. (2005)		Bontoux et al. (2008)		Goldbarg et al. (2009)		GRASP + Filtering/PR			
								P		R	
		Gap (%)	T (s)	Gap (%)	T (s)	Gap (%)	T (s)	Gap (%)	T (s)	Gap (%)	T (s)
|M|	50	0.07	3	**0.00**	2	**0.00**	1	0.27	35	0.68	32
	100	0.14	10	**0.00**	20	**0.00**	1	1.49	56	1.16	51
	150	**0.03**	14	0.08	172	**0.03**	2	2.83	67	3.04	73
	200	0.32	19	0.02	232	**0.00**	3	2.26	83	2.49	75
	250	0.06	25	0.01	154	**0.00**	3	2.72	106	5.03	109
|K|	50	0.07	5	**0.00**	37	**0.00**	1	1.07	27	2.06	28
	100	0.24	13	0.05	154	**0.00**	2	0.64	54	0.55	45
	150	0.1	20	**0.00**	96	**0.00**	2	3.68	74	4.09	80
	200	0.08	21	**0.03**	165	**0.03**	3	2.26	121	3.22	120
Average		0.12	14	0.02	115	**0.01**	2	1.91	69	2.48	68

symmetric Euclidean instances. The size of instances is given by $|M| \in [50 - 350]$ and $|K| \in [50 - 200]$, with five instances per combination of M and K, leading to a total of 140 instances. Several authors have solved this set of instances: first, Riera-Ledesma and Salazar-González (2006) implement a local search algorithm; second, Bontoux and Feillet (2008) implement an Ant colony Optimization algorithm; and Goldbarg et al. (2009) implement a Transgenetic algorithm.

In this class, the closed instances (i.e., those with known optimal) correspond to 89 out of the 140 instances. For the 51 remaining instances, the BKS are found in the works of Bontoux and Feillet (2008) and Goerler et al. (2013). We compare the quality of our solutions based on Equation (1) for the closed instances, and report the best solution found with our methodology for the open instances.

Table 5 shows average results for the optimality gap and the computation time of each methodology for closed instances. The average values are grouped by instance size. All methodologies perform five replicates per instance, except for Riera-Ledesma and Salazar-González (2006), which performs one run per instance. The methodology proposed by Goldbarg et al. (2009) outperforms all methodologies with an average gap of 0.01%.

Table 6 shows the best solution obtained by each of the methodologies for open instances. The methodology proposed in this paper obtains solutions that match 34 of the BKS already reported in the literature and reports a new BKS (instance $350 - 150 - 1$).

5. Conclusions

We presented a GRASP-based methodology for the TPP, which achieves significantly better performance than published results for the benchmark instances proposed by Singh and van Oudheusden (1997). The proposed methodology consists of three constructive procedures, two local search operators, and three strategies to enhance the overall methodology. The constructive procedures

Table 6
Objective function value and computational times (seconds) for the proposed approach and the latest reported results (Bontoux and Feillet, 2008; Goldbarg et al., 2009) for the Class 3 instances that have not been solved to optimality. Values in bold correspond to BKSs and values with asterisks correspond to optimal solutions

			Bontoux et al. (2008)		Goldbarg et al. (2009)		GRASP + Filtering/PR	
$\|M\|$	$\|K\|$	Id	Sol	T (s)	Sol	T (s)	Sol	T (s)
200	150	4	**2419**	1217	**2419**	24	**2419**	52
200	200	4	**2344**	527	**2344**	99	2403	78
250	100	1	**1301**	34	**1301**	143	**1301**	40
250	100	4	**1673**	10	**1673**	4	**1673**	29
250	100	5	**1641**	550	**1641**	2	**1641**	58
250	150	4	**1836**	45	**1836**	2	**1836**	77
250	150	5	**1531**	21	**1531**	151	**1531**	87
250	200	2	**2785**	1138	2786	246	2838	75
250	200	3	**1924**	282	**1924**	16	1947	269
250	200	4	**2116**	84	**2116**	3	**2116**	63
250	200	5	**1797**	930	**1797**	39	1851	179
300	50	1	**1477**	160	**1477**	2	1485	33
300	50	2	**813**	116	**813**	1	879	39
300	50	3	**1117**	20	**1117**	1	**1117**	17
300	50	4	**1176**	2	**1176**	1	**1176**	20
300	50	5	1257	276	**1256**	2	**1256**	22
300	100	1	**1035**	56	**1035**	2	**1035**	37
300	100	2	**1179**	617	1180	4	**1179**	29
300	100	3	**1498**	103	**1498**	2	**1498**	32
300	100	4	**1749**	312	**1749**	37	1770	122
300	100	5	**1774**	3	**1774**	2	**1774**	33
300	150	1	**1457**	757	**1457**	99	**1457**	187
300	150	2	**1656**	483	**1656**	3	**1656**	118
300	150	3	2485	663	**2484**	6	**2484**	77
300	150	4	**1801**	96	**1801**	8	**1801**	53
300	150	5	**1816**	309	**1816**	41	**1816**	87
300	200	1	1815	488	**1803**	575	1814	214
300	200	2	1791	1919	**1790**	628	1839	250
300	200	3	2442	2852	**2437**	184	2443	487
300	200	4	**1815**	2947	**1815**	114	1960	121
300	200	5	2022	1578	**2014**	605	2022	220
350	50	1	**723**	46	**723**	2	**723**	15
350	50	2	**736**	26	**736**	13	**736**	37
350	50	3	**942**	6	**942**	2	**942**	29
350	50	4	**805**	379	**805**	5	917	24
350	50	5	**1125**	26	1225	2	1225	23
350	100	1	**1317**	1699	**1317**	230	**1317**	90
350	100	2	**962**	156	**962**	2	**962**	43
350	100	3	**796**	840	**796**	2	**796**	59
350	100	4	**1059**	14	**1059**	9	**1059**	39
350	100	5	**1566**	465	**1566**	42	**1566**	58
350	150	1	1457	1986	1459	320	**1455**	142

Continued

Table 6
Continued

			Bontoux et al. (2008)		Goldbarg et al. (2009)		GRASP + Filtering/PR	
$\|M\|$	$\|K\|$	Id	Sol	T (s)	Sol	T (s)	Sol	T (s)
350	150	2	**1315**	159	**1315**	16	**1315**	140
350	150	3	**2553**	258	2558	598	**2553**	102
350	150	4	**1239**	596	**1239**	3	**1239**	87
350	150	5	**2288**	9	**2288**	229	**2288**	115
350	200	1	1503	1033	**1498**	25	1511	485
350	200	2	1374	3085	**1369**	56	**1369**	369
350	200	3	**1873**	369	**1873**	59	**1873**	444
350	200	4	1385	122	**1356**	33	1359	318
350	200	5	**2336**	2386	**2336**	205	**2336**	131
Total			**80,896**	32,254	80,938	**4899**	81,558	5950

consist in: (i) focusing on purchase decisions first; (ii) focusing on routing decisions first; (iii) randomly choosing either of the two previous procedures. The local search operators insert and remove nodes in the tour, analogous to Add and Drop moves in the literature. The enhancement strategies are a reactive GRASP that adjusts the size of the restricted candidate list as a function of performance; a filtering process that avoids the application of local search operations on poorly performing candidate solutions; and a Path-Relinking strategy that combines elite solutions from the GRASP seeking to obtain better performing solutions.

Computational experiments on the set of instances (Singh and van Oudheusden, 1997) suggest that the Path-Relinking strategy is significantly effective at improving GRASP solutions' objective function, but requires considerable computational effort. The filtering strategy does not show competitive performance in terms of the objective function by itself. However, it proves to be useful when combined with the Path-Relinking strategy, as it helps to improve the efficiency of the algorithm with the minor detriment of the objective function (with respect to applying Path-Relinking alone). The purchase-first constructive procedure shows competitive computational times and consistently leads to smaller optimality gaps than other constructive procedures, achieving results within 1% of optimal or best-known solutions within 4500 iterations. The third constructive procedure (randomly alternate between purchasing and routing decisions) is the fastest but shows the least competitive results in terms of the objective function. Additional experiments were performed on Class 1 and Class 3 instances, which correspond to the symmetric TPP. The proposed methodology shows competitive performance on the Class 1 instances, obtaining BKS and optimal solutions in a majority of instances. However, in Class 3 instance, only one new BKS is achieved, although many solutions match existing BKS.

Ongoing research is devoted to the incorporation of stochasticity into key TPP parameters such as travel time and prices. The inclusion of other TPP variants (e.g., the restricted and dynamic cases) is also part of ongoing work, as well as the integration of optimization with data analysis on information systems, seeking to make our algorithms more attractive for practical applications.

References

Almeida, C.P., Gonçalves, R.A., Goldbarg, E.F., Goldbarg, M.C., Delgado, M.R., 2012. An experimental analysis of evolutionary heuristics for the biobjective traveling purchaser problem. *Annals of Operations Research* 199, 1, 305–341.

Barbalho, H., Rosseti, I., Martins, S.L., Plastino, A., 2013. A hybrid data mining grasp with path-relinking. *Computers & Operations Research* 40, 12, 3159–3173.

Bernardino, R., Paias, A., 2018. Metaheuristics based on decision hierarchies for the traveling purchaser problem. *International Transactions in Operational Research* 25, 4, 1269–1295.

Boctor, F.F., Laporte, G., Renaud, J., 2003. Heuristics for the traveling purchaser problem. *Computers & Operations Research* 30, 4, 491–504.

Bontoux, B., Feillet, D., 2008. Ant colony optimization for the traveling purchaser problem. *Computers & Operations Research* 35, 2, 628–637.

Boudia, M., Louly, M.A.O., Prins, C., 2007. A reactive GRASP and path relinking for a combined production–distribution problem. *Computers & Operations Research* 34, 11, 3402–3419.

Burstall, R.M., 1966. A heuristic method for a job-scheduling problem. *Journal of the Operational Research Society* 17, 3, 291–304.

Cambazard, H., Penz, B., 2012. A constraint programming approach for the traveling purchaser problem. In Milano, M. (ed.) *International Conference on Principles and Practice of Constraint Programming*. Springer, Berlin, pp. 735–749.

Campos, V., Martí, R., Sánchez-Oro, J., Duarte, A., 2014. Grasp with path relinking for the orienteering problem. *Journal of the Operational Research Society* 65, 12, 1800–1813.

El-Dean, R.A.H.Z., 2008. A tabu search approach for solving the travelling purchase problem. Proceedings of the International Conference on Informatics and System, INFOS2008, Citeseer, pp. 24–30.

Festa, P., Resende, M.G., 2013. Hybridizations of grasp with path-relinking. In Talbi, E.G. (ed.) *Hybrid Metaheuristics*. Springer, Berlin, pp. 135–155.

Glover, F., 1997. Tabu search and adaptive memory programming—advances, applications and challenges. In Barr, R.S., Helgason, R.V., Kennington, J.L. (eds) *Interfaces in Computer Science and Operations Research*. Springer, Berlin, pp. 1–75.

Goerler, A., Schulte, F., Voß, S., 2013. An application of late acceptance hill-climbing to the traveling purchaser problem. In Pacino, D., Voß, S., Jensen, R.M. (eds) *International Conference on Computational Logistics*. Springer, Berlin, pp. 173–183.

Goldbarg, M.C., Bagi, L.B., Goldbarg, E.F.G., 2009. Transgenetic algorithm for the traveling purchaser problem. *European Journal of Operational Research* 199, 1, 36–45.

Golden, B., Levy, L., Dahl, R., 1981. Two generalizations of the traveling salesman problem. *Omega* 9, 4, 439–441.

Gouveia, L., Paias, A., Voß, S., 2011. Models for a traveling purchaser problem with additional side-constraints. *Computers & Operations Research* 38, 2, 550–558.

Ho, S.C., Szeto, W.Y., 2016. Grasp with path relinking for the selective pickup and delivery problem. *Expert Systems with Applications* 51, 14–25.

Laporte, G., Riera-Ledesma, J., Salazar-González, J.J., 2003. A branch-and-cut algorithm for the undirected traveling purchaser problem. *Operations Research* 51, 6, 940–951.

Lin, S., Kernighan, B.W., 1973. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research* 21, 2, 498–516.

Manerba, D., Mansini, R., Riera-Ledesma, J., 2017. The traveling purchaser problem and its variants. *European Journal of Operational Research* 259, 1, 1–18.

Mansini, R., Pelizzari, M., Saccomandi, R., 2005. An effective tabu search algorithm for the capacitated traveling purchaser problem. Technical report TR2005-10-49. DEA, University of Brescia.

Martínez, D.A., Alvarez-Valdes, R., Parreño, F., 2015. A grasp algorithm for the container loading problem with multi-drop constraints. *Pesquisa Operacional* 35, 1, 1–24.

Marzo, R.G., Ribeiro, C.C., 2020. A grasp with path-relinking and restarts heuristic for the prize-collecting generalized minimum spanning tree problem. *International Transactions in Operational Research* 27, 3, 1419–1446.

Miller, C.E., Tucker, A.W., Zemlin, R.A., 1960. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)* 7, 4, 326–329.

Morán-Mirabal, L., González-Velarde, J., Resende, M.G., 2014. Randomized heuristics for the family traveling salesperson problem. *International Transactions in Operational Research* 21, 1, 41–57.

Ochi, L.S., Drummond, L.M., Figueiredo, R.M., 1997. Design and implementation of a parallel genetic algorithm for the travelling purchaser problem. Proceedings of the 1997 ACM Symposium on Applied Computing, pp. 257–262.

Ochi, L.S., Silva, M.B., Drummond, L., 2001. Metaheuristics based on GRASP and VNS for solving traveling purchaser problem. 4th Metaheuristics International Conference, pp. 489–494.

Ong, H.L., 1982. Approximate algorithms for the travelling purchaser problem. *Operations Research Letters* 1, 5, 201–205.

Pearn, W.L., 1991. On the traveling purchaser problem. Technical Report, Department of Industrial Engineering and Management, National Chiao Tung University, pp. 91–01.

Ramesh, T., 1981. Travelling purchaser problem. *Journal of the Operational Research* 18, 78–91.

Resendel, M.G., Ribeiro, C.C., 2005. Grasp with path-relinking: recent advances and applications. In Ibaraki, T., Nonobe, K., Yagiura, M. (eds) *Metaheuristics: Progress as Real Problem Solvers*. Springer, Berlin, pp. 29–63.

Ribeiro, C.C., Resende, M.G., 2012. Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics* 18, 2, 193–214.

Riera-Ledesma, J., Salazar-González, J.J., 2005. A heuristic approach for the travelling purchaser problem. *European Journal of Operational Research* 162, 1, 142–152.

Riera-Ledesma, J., Salazar-González, J.J., 2006. Solving the asymmetric traveling purchaser problem. *Annals of Operations Research* 144, 1, 83–97.

Singh, K.N., van Oudheusden, D.L., 1997. A branch and bound algorithm for the traveling purchaser problem. *European Journal of Operational Research* 97, 3, 571–579.

Teeninga, A., Volgenant, A., 2004. Improved heuristics for the traveling purchaser problem. *Computers & Operations Research* 31, 1, 139–150.

Voß, S., 1996a. Add and drop-procedures for the traveling purchaser problem. *Methods of Operations Research* 53, 317–318.

Voß, S., 1996b. Dynamic tabu search strategies for the traveling purchaser problem. *Annals of Operations Research* 63, 2, 253–275.