Computers and Operations Research 105 (2019) 118–131

Contents lists available at ScienceDirect

Computers and Operations Research

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

Heuristics for vehicle routing problems: Sequence or set optimization?

Túlio A.M. Toffolo[a,b,*], Thibaut Vidal[c], Tony Wauters[a]

[a] Department of Computer Science, CODeS, KU Leuven, Belgium
[b] Department of Computing, Federal University of Ouro Preto, Brazil
[c] Computer Science Department, Pontifícia Universidade Católica do Rio de Janeiro, Brazil

A R T I C L E I N F O

Article history:
Received 19 January 2018
Revised 27 December 2018
Accepted 28 December 2018
Available online 29 December 2018

Keywords:
Decision-set decompositions
Metaheuristics
Dynamic programming
Integer programming
Large neighborhood search
Vehicle routing problem

A B S T R A C T

We investigate a structural decomposition for the capacitated vehicle routing problem (CVRP) based on vehicle-to-customer “assignment” and visits “sequencing” decision variables. We show that an heuristic search focused on assignment decisions with a systematic optimal choice of sequences (using Concorde TSP solver) during each move evaluation is promising but requires a prohibitive computational effort. We therefore introduce an intermediate search space, based on the dynamic programming procedure of Balas & Simonetti, which finds a good compromise between intensification and computational efficiency. A variety of speed-up techniques are proposed for a fast exploration: neighborhood reductions, dynamic move filters, memory structures, and concatenation techniques. Finally, a tunneling strategy is designed to *reshape* the search space as the algorithm progresses. The combination of these techniques within a classical local search, as well as in the unified hybrid genetic search (UHGS) leads to significant improvements of solution accuracy. New best solutions are found for surprisingly small instances with as few as 256 customers. These solutions had not been attained up to now with classic neighborhoods. Overall, this research permits to better evaluate the respective impact of sequence and assignment optimization, proposes new ways of combining the optimization of these two decision sets, and opens promising research perspectives for the CVRP and its variants.

1. Introduction

The Capacitated Vehicle Routing Problem (CVRP) is classically described as a combination of a Traveling Salesman Problem (TSP) with an additional capacity constraint which lends a Bin Packing (BP) substructure to the problem (Toth and Vigo, 2014). It can be seen as a Set Partitioning (SP) problem in which the cost of each set corresponds to the distance of the associated optimal TSP tour (Balinski and Quandt, 1964). These problem representations emphasize the two decision sets at play: customer-to-vehicle Assignments, and Sequencing choices for each route (Vidal et al., 2013b), a duality that has left long-standing impressions in the literature, from the early developments of route-first cluster-second (Beasley, 1983; Bodin and Berman, 1979) and cluster-first route-second constructive methods (Fisher and Jaikumar, 1981), all the way to the set-covering-based exact methods and matheuristics which are currently gaining popularity.

Examining the recent progress on metaheuristics for the CVRP, little has changed in recent years concerning intra-route neighborhood search: Relocate, Swap and 2-opt neighborhoods and their immediate generalizations are employed, and these neighborhoods alone are sufficient to guarantee that most solutions resulting from a local search contain TSP–optimal routes. This is generally because classical CVRP instances involve short routes with up to 15 or 20 visits. For such small problems, even simple neighborhood search methods for the TSP tend to produce optimal tours.

Based on this observation, a larger effort dedicated to TSP tour optimization, as a stand-alone neighborhood, is unlikely to result in further improvements. For this reason, it is very uncommon to observe the use of larger *intra-route* neighborhoods (e.g., 3-Opt or beyond) in recent state-of-the-art metaheuristics. Nevertheless, does this mean that Sequencing optimization should be abandoned in favor of more extensive search concerning Assignment choices? Certainly not. Indeed, even if local minima exhibit optimal TSP tours, inter-route moves frequently lead to TSP-suboptimal tours which are rejected due to their higher cost, but would be accepted otherwise if the tours were optimized. Such solution improvements would then not arise from separate Assignment or Sequencing optimizations, but from a careful combination of both.

Fig. 1 schematically represents the solution set of the CVRP, whose decision variables are split into Sequencing decisions (x-axis) and Assignment decisions (y-axis). The y-axis also represents

[*] Corresponding author at: Department of Computing, Federal University of Ouro Preto, Brazil.
E-mail address: tulio@toffolo.com.br (Túlio A.M. Toffolo).

https://doi.org/10.1016/j.cor.2018.12.023
0305-0548/© 2018 Published by Elsevier Ltd.

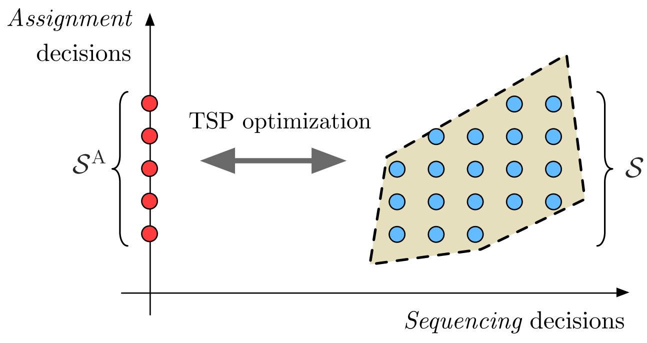


Fig. 1. Two alternative search spaces for the CVRP.

the solutions in terms of their Assignment decisions solely, ignoring Sequencing choices. These partial solutions can be viewed as a projection (Geoffrion, 1970) of the original solutions $\mathcal{S}$ on the space $\mathcal{S}^{\mathrm{A}}$ defined by a single decision subset (Assignment). Moreover, from a solution represented in terms of Assignment decisions, it is possible to find the best associated complete solution by solving each TSP associated with the routes.

With this picture in mind, it is tempting to conduct a search in the space $\mathcal{S}^{\mathrm{A}}$ rather than $\mathcal{S}$. After all, the size of $\mathcal{S}^{\mathrm{A}}$ is exponentially smaller than that of $\mathcal{S}$, the solutions of $\mathcal{S}^{\mathrm{A}}$ are in average of better quality, and the average size of a path in $\mathcal{S}^{\mathrm{A}}$ is smaller, such that fewer LS iterations are expected for convergence. However, the obvious drawback is that each move evaluation in $\mathcal{S}^{\mathrm{A}}$ requires solving one or several small TSPs to optimality, leading to a significant computational effort. Still, note that considerable progress has been made in the past 30 years with regard to the efficient solution of TSPs, and small problems with approximately 20 customers are solvable in a few milliseconds. Based on these observations, this article takes a fresh look at heuristic searches for the CVRP to answer two essential questions about the search space $\mathcal{S}^{\mathrm{A}}$:

1. Is it practical and worthwhile to search in the space $\mathcal{S}^{\mathrm{A}}$ rather than $\mathcal{S}$?
2. If searching in $\mathcal{S}^{\mathrm{A}}$ requires an excessive effort, can we define a search space which maintains most of the key properties of $\mathcal{S}^{\mathrm{A}}$ but can be more efficiently explored?

As will be demonstrated in Section 4, our experiments considering the Concorde solver led us to currently answer the first question negatively: even with non-trivial memory and speedup techniques (hashtables and move filters) the computational overhead related to the exact solution of TSPs by Concorde during each move evaluation, for a complete search in $\mathcal{S}^{\mathrm{A}}$, does not appear worth the gain in terms of solution quality.

By contrast, our answer to the second question is positive. Rather than requiring a complete exact solution of each TSP, the dynamic programming approach of Balas and Simonetti (2001), hereafter referred to as B&S, can be employed to perform a restricted route optimization *during move evaluations*. Given a range parameter k and an initial tour, the B&S algorithm finds, in $\mathcal{O}(k^2 2^{k-2} n)$ operations, the vertex sequence with minimum cost such that no vertex is displaced by more than k positions. This allows us to define a search space $\mathcal{S}^{\mathrm{B}}_k$ such that $\mathcal{S}^{\mathrm{B}}_0 = \mathcal{S}$ and $\lim_{k\to\infty} \mathcal{S}^{\mathrm{B}}_k = \mathcal{S}^{\mathrm{A}}$. Moreover, even for a fixed k, we propose *tunneling* techniques that exploit the memory of past solutions to dynamically reshape the search space, in such a way that $\mathcal{S}^{\mathrm{B}}_k$ converges towards $\mathcal{S}^{\mathrm{A}}$ as the search progresses.

To evaluate experimentally the potential of the new search spaces, we conduct experiments with a simple local search (LS), and with the unified hybrid genetic search (UHGS) of Vidal et al. (2012, 2014). The exploration of the search spaces $\mathcal{S}^{\mathrm{B}}_k$ for $k \in \{1, \dots, 3\}$ appears to lead to solutions of higher quality on the new instances from Uchoa et al. (2017). New best solutions were also found for surprisingly small instances with as few as 256 customers, highlighting the improvement potential of the proposed search techniques. More generally, this study allows to better evaluate the impact of Sequencing and Assignment optimization in heuristics, giving new ways of jointly optimizing these two decision sets, and leading to new state-of-the-art CVRP algorithms.

2. Related literature

This section reviews some key milestones concerning the management of Sequencing and Assignment decisions in vehicle routing heuristics, as well as decision-set decompositions.

Sequencing and Assignment decisions were first optimized separately in early constructive heuristics, giving rise to different families of methods. Route-first cluster-second algorithms (Beasley, 1983; Bodin and Berman, 1979) first produce a giant TSP tour, before subsequently assigning consecutive visits into separate trips to produce a complete solution. In cluster-first route-second methods (Fisher and Jaikumar, 1981), a clustering algorithm is employed to group customer visits into clusters, followed by TSP optimizations. Finally, *petal algorithms* (Foster and Ryan, 1976; Renaud et al., 1996) are based on an a-priori generation of candidate routes (petals), followed by the solution of a set partitioning problem.

In the development of local search and metaheuristic algorithms which ensued in the 1990s and thereafter, Assignment and Sequencing optimizations began to be better integrated. Classical moves such as Relocate, Swap, 2-opt* and their close variants allow to optimize both decision subsets. The associated neighborhood search methods form the basis of the vast majority of state-of-the-art algorithms. Petal algorithms have been naturally extended into hybrid metaheuristics: high-quality routes are extracted from local minima instead of being enumerated in advance, and the solution of the associated set partitioning problems fulfills the role of a large neighborhood search (see, e.g., Ahuja et al., 2002; Muter et al., 2010; Subramanian et al., 2013).

The variety of vehicle routing problem variants has also triggered studies concerning problem decompositions. Vidal et al. (2013b) established a review of the classical variants and their associated constraints, objectives, and decision sets, called *attributes*. The attributes were classified in relation to their impact on Sequencing, Assignment decisions, and Route evaluations in heuristics, leading to a structural problem decomposition which serves as a basis for the Unified Hybrid Genetic Search (UHGS) algorithm of Vidal et al. (2014) and allows to produce state-of-the-art results for dozens of VRP variants. Many problem attributes come jointly with new decision subsets, e.g., when optimizing vehicle routing with packing, timing or scheduling constraints (Goel and Vidal, 2014; Pollaris et al., 2015; Vidal et al., 2015b), visit choices (Bulhões et al., 2018; Vidal et al., 2016) or service-mode choices (Vidal, 2017; Vidal et al., 2015a).

Decision-set decompositions are employed throughout many of the aforementioned papers to perform a search in the space of Sequencing and Assignment choices and optimally determine the remaining decision variables during each route and move evaluation. In this paper, the decision-set decomposition does not result from supplementary problem attributes, but is instead used to define exponential-size polynomially-searchable neighborhoods and transform the search space. Exponential-size neighborhoods have a long history in the combinatorial optimization literature (Ahuja et al., 2002; Bompadre, 2012; Deineko and Woeginger, 2000). Most of these neighborhoods are based on shortest path or matching subproblems, as well as specific graph and distance matrix structures with which some NP-hard problems become tractable (consider, as examples, Halin graphs or Monge matrices). As a rule of thumb, larger neighborhoods and faster search procedures are generally desirable. There are, however, theoretical limitations to the size of polynomially-searchable neighborhoods. Gutin and Yeo (2003) proved that, for the TSP, no neighborhood of cardinality at least $(n-k)!$ for a given constant k can be searched unless NP $\subset$ P/poly.

The neighborhood of Balas and Simonetti (2001) is an exponential-size neighborhood for the TSP. Given an incumbent tour represented as a permutation σ and a value k, it contains all permutations π of σ such that π fulfills $\pi(1)=1$ and $\pi(i) \leq \pi(j)$ for all $i, j \in \{1,\ldots,n\}$ such that $i+k \leq j$. In other words, if i precedes j by more than k positions in σ, then $\pi(i)$ precedes $\pi(j)$. Setting $\pi(1)=1$ allows to fix the origin location (e.g., depot). This neighborhood contains $2^{\Theta(n)}$ solutions, and can be explored in $\mathcal{O}(k^2 2^{k-2} n)$ using dynamic programming. This is a linear time complexity when k is constant, and a polynomial complexity when $k = \mathcal{O}(\log n)$. Balas and Simonetti (2001) performed extensive experiments, and demonstrated that this dynamic programming procedure can be used as a stand-alone neighborhood to improve high quality local minima of the TSP and its immediate variants. Later, Gschwind and Drexl (2016); Irnich (2008), and Hintsch and Irnich (2018) employed this neighborhood to solve arc-routing problems with possible cluster constraints, and dial-a-ride problems. One common characteristic of these studies is that they employed B&S as a stand-alone neighborhood for route improvements. Only in one conference presentation (Irnich, 2013), the possibility of using the B&S neighborhood *in combination* with some classical CVRP moves has been highlighted, but the performance of such an approach remains largely unexplored.

We seek to go one step further. Rather than applying this tour optimization procedure as a stand-alone optimization technique or in combination with a single classical neighborhood, we investigate its *systematic use* in combination with every move of a classical CVRP local search. As discussed in the following, the methodological implications of such a redefinition of the search space are noteworthy.

3. Proposed methodology

We will describe the methodology as a local search on indirect solution representations, using a decoder. This algorithm can be readily extended into a wide range of vehicle routing metaheuristics, e.g., tabu search, iterated local search, or hybrid genetic algorithm (Gendreau and Potvin, 2010; Laporte et al., 2014). There is no widely accepted term, in the current heuristic literature, for referring to the elements which represent such indirect solutions. The evolutionary literature usually refers to a *genotype* to denote solution encodings (and *phenotype* for the solutions themselves), whereas the local-search based metaheuristic literature refers to *incomplete* or *indirect* solutions (which are converted into complete solutions via a *decoder* function). *Incomplete* lets us think that the representation is necessarily a subset of a complete solution, and unnecessarily restricts the application scope. To circumvent this issue, we henceforth employ the term *primitive solutions*. We first recall some basic definitions related to neighborhood search and indirect solution representations, then proceed with an analysis of alternative search spaces and the description of the proposed local search algorithm.

Definition 1 (Primitive solutions and search space)**.** We consider a combinatorial optimization problem of the form $\min_{x \in X} z(x)$, where X is the solution space, and z is an objective function to minimize. Let Y be the set of primitive solutions, and let the decoder f: $Y \rightarrow X$ be an injective application that transforms any $y \in Y$ into a complete solution $x \in X$. A neighborhood is defined as a mapping $\mathcal{N}: Y \rightarrow 2^Y$ that associates with each primitive solution y a set of neighbors $\mathcal{N}(y) \subset Y$. The graph induced by Y and $\mathcal{N}$ is referred to as the search space.

Definition 2 (TSP–optimal tour)**.** A tour σ is TSP–optimal if there exists no other permutation π of σ such that $\pi(1)=1$ with a shorter total distance.

Definition 3 (B^k–optimal tour)**.** A tour σ is B^k–optimal if there exists no other permutation π of σ with a shorter total distance such that $\pi(1)=1$ and $\pi(i) \leq \pi(j)$ for all $i, j \in \{1,\ldots,n\}$ with $i+k \leq j$. The parameter k is the range of the B&S neighborhood.

3.1. A choice of search space

In this section, we examine the search spaces associated with the set of all solutions ($\mathcal{S}$), of those with TSP–optimal tours ($\mathcal{S}^{\mathrm{A}}$), and of those with B^k–optimal tours ($\mathcal{S}^{\mathrm{B}}_k$) and discuss their relative merits.

Search space $\mathcal{S}$. Classical local search methods for the CVRP do not distinguish between primitive and complete solutions. In the usual search space $\mathcal{S}$, solution sets X and Y are equal and the decoder f is the identity function. The neighborhood $\mathcal{N}$ is based on the definition of one or several classes of moves. A move ϕ is a local modification that can be applied to a primitive solution y to generate a neighbor $\phi(y) \in \mathcal{N}(y)$. For each search space considered in this paper, we will eventually refer to several classes of moves, but to a single neighborhood only, which corresponds to the union of all primitive solutions attainable from y via one single move. Classical moves for the CVRP are based on relocations and exchanges of a bounded number of vertices, or replacements of a bounded number of edges. Most common neighborhoods have a quadratic cardinality ($|\mathcal{N}(y)| = \mathcal{O}(n^2)$ for all $y \in Y$). We refer to Vidal et al. (2013b) for a comprehensive survey on classical local searches for the CVRP.

Fig. 2 represents the search space $\mathcal{S}$ associated with Relocate moves only, for a small asymmetric CVRP instance with three customers. There are 13 possible solutions for this problem, each represented by a set of ordered customer visits. Solution '[1,2,3]', for example, employs one vehicle to visit customers 1, 2 and 3, while solution '[1,2,3]' employs three vehicles, one per customer. Each solution is represented by a node, positioned on the x-axis according to its quality (the more to the right, the better a solution is). The set of outgoing arcs of each solution points towards its neighbors. Moreover, solutions with identical customer-route assignments are grouped within dashed areas. Note that, for this instance size, it is always possible to reach the optimum solution from any starting point in two successive moves. In a local search that explores the neighborhood in random order and applies an improving move as soon as it is found, the worst case corresponds to five moves (when the initial solution is '[2,1,3]' or '[1,2,3]').

Search space $\mathcal{S}^{\mathrm{A}}$. As discussed earlier in this work, CVRP solutions can also be represented in terms of their Assignment decisions, excluding the Sequencing decisions in the representation and delegating the choices of the best visit sequences to the decoder. With such a paradigm, one can define a local search in the space Y of primitive solutions, where each $y \in Y$ represents an assignment of customer visits to vehicles in such a way that the capacity constraints are respected. The decoder f is based on an exact TSP solver, responsible for generating the best visit sequence originating and finishing at the depot for each subset of customers. In this sense, the image $f[Y] \subset X$ contains exclusively solutions with TSP–optimal tours.

The neighborhood used to explore the search space $\mathcal{S}^{\mathrm{A}}$ can remain similar to classical CVRP neighborhoods, based on relocations or exchanges of customers between subsets, or involve other families of moves specialized for assignment and partition problems.

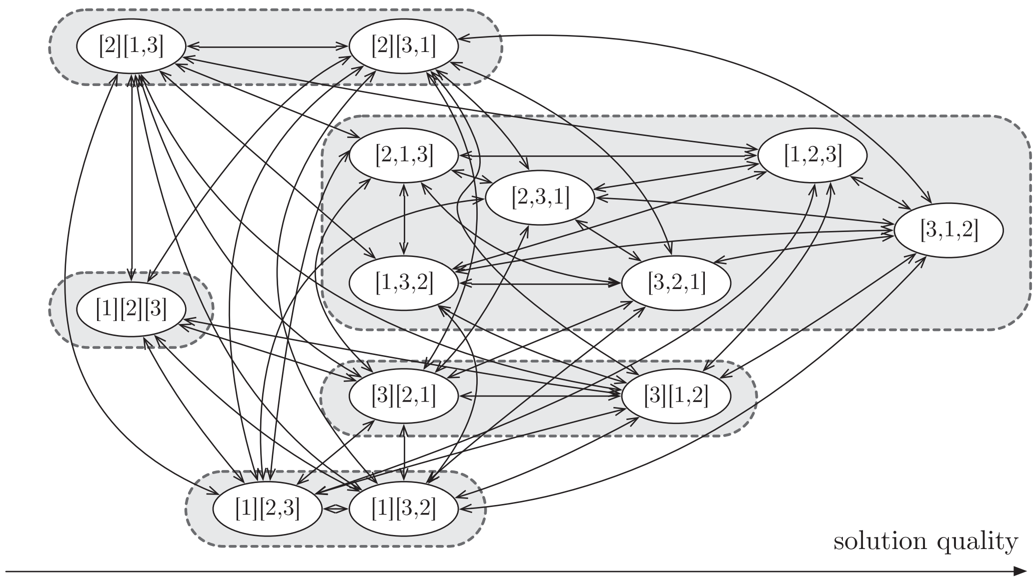


Fig. 2. Search space $\mathcal{S}$ for a small asymmetric CVRP instance.

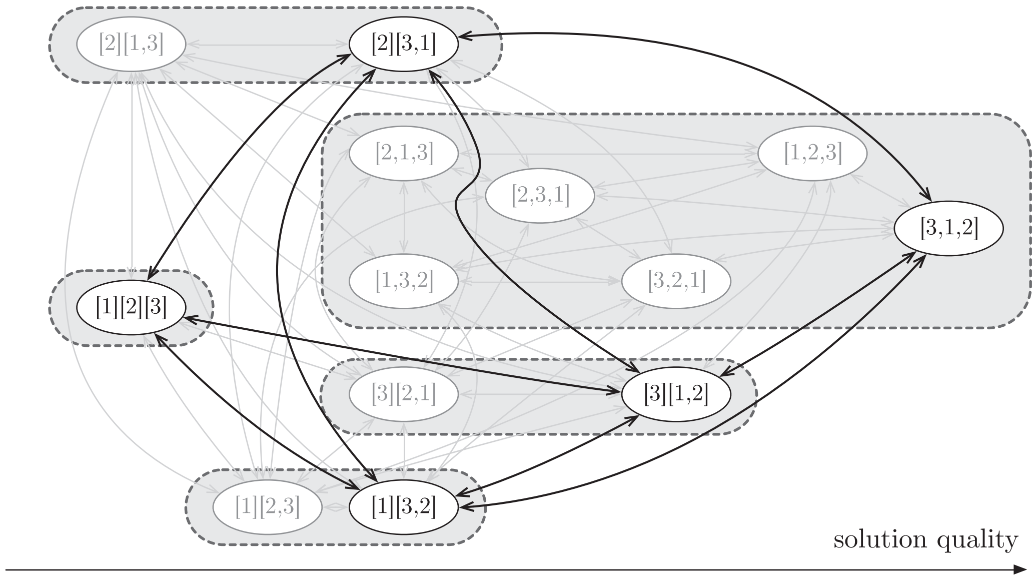


Fig. 3. Search space $\mathcal{S}^{\mathrm{A}}$ for a small asymmetric CVRP instance.

Fig. 3 represents the resulting search space with simple RELOCATE moves. Only TSP–optimal tours are explored and therefore the size of the search space reduces down to six primitive solutions. The other solutions and their connections are represented in light gray. Note, in our small example, that now at most three successive improving moves may be applied to attain the optimum from '[1,2,3]'.

Search space $\mathcal{S}^{\mathrm{A}}$ is smaller than $\mathcal{S}$, and our computational experiments (Section 4) demonstrate that a search in this space indeed leads to solutions with higher quality. However, each move evaluation in this space requires executing an algorithm with exponential worst-case time complexity, a TSP solver, in order to decode each primitive solution in the neighborhood for cost evaluation. Although research on the TSP has culminated in very efficient algorithms over the past thirty years, thousands (or millions) of small TSP instances should be solved during a local search in $\mathcal{S}^{\mathrm{A}}$, and thus the total computational effort dedicated towards decoding can grow prohibitively large. Moreover, bad behavior in a single case (e.g., due to an unusual long route with many customers or bad branching decisions) can be sufficient, without any other safeguard, to stall the entire algorithm.

Search space $\mathcal{S}^{\mathrm{B}}_k$. To circumvent the aforementioned issues, we study an alternative search space in which the set of primitive solutions Y is a subset of the complete solutions (with their ASSIGNMENT and SEQUENCING decisions), but where the decoder f is nontrivial, and consists of applying B&S multiple times to each route with a fixed range (k value) until the tours become B^k–optimal. With these assumptions, the image $f[Y]$ contains exclusively complete solutions with B^k–optimal tours. As such, the application of B&S can be viewed as a post-optimization step *during* classical CVRP move evaluations, opening the way for additional solution improvements. A careful analysis of the resulting search space gives even more significance to this approach, due to three properties:

Property 1. *From an initial solution containing a B^k–optimal tour, a local search in the space $\mathcal{S}^{\mathrm{B}}_k$ explores only B^k–optimal tours.*

Property 2. *For a fixed range k, each move evaluation and subsequent solution decoding is done in polynomial time as a function of n and the number of applications of* B&S.

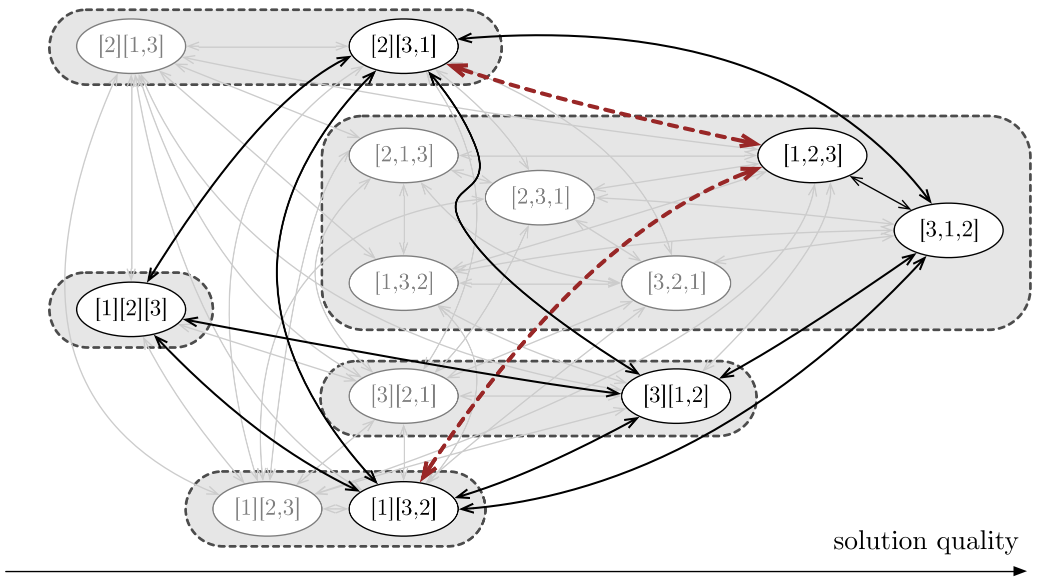


Fig. 4. Search space $\mathcal{S}_1^{\mathrm{B}}$ for a small asymmetric CVRP instance.

Property 3. *The search space* $\mathcal{S}_k^{\mathrm{B}}$ *is such that* $\mathcal{S}_0^{\mathrm{B}} = \mathcal{S}$ *and* $\mathcal{S}_{n-1}^{\mathrm{B}} = \mathcal{S}^{\mathrm{A}}$, *with n being the number of customers.*

These three properties are all fundamental for the methodology that follows. Property 1 demonstrates how space $\mathcal{S}_k^{\mathrm{B}}$ contains fewer solutions than $\mathcal{S}$ when $k > 0$, and that the overall quality of these solutions tends to be higher (since non-B^k-optimal tours are filtered out). Moreover, Property 2 gives some computational time guarantees: even if the computational effort grows quickly with the range k, the effort of the decoder is guaranteed to remain stable for all $y \in Y$ when k is constant, eliminating the possibility of a computational effort peak for specific TSP instances. Finally, Property 3 demonstrates how k balances the effort dedicated towards the optimization of the Assignment and Sequencing decision sets, and establishes $\mathcal{S}_k^{\mathrm{B}}$ as an intermediate search space generalizing $\mathcal{S}$ and $\mathcal{S}^{\mathrm{A}}$.

Fig. 4 illustrates the search space $\mathcal{S}_k^{\mathrm{B}}$ for the same example as previous figures with $k = 1$. It is an intermediate between the spaces depicted by Figs. 2 and 3, which correspond to $\mathcal{S}_0^{\mathrm{B}} = \mathcal{S}$ and $\mathcal{S}_2^{\mathrm{B}} = \mathcal{S}^{\mathrm{A}}$, respectively. Note how the solution [1,2,3] is now a neighbor of [2,3,1] and [1,3,2], as highlighted by the two dotted arcs.

Discussions and choice. In light of these observations, we have conducted computational experiments on search space $\mathcal{S}_k^{\mathrm{B}}$, using the dynamic programming algorithm of Balas and Simonetti (2001) to decode each solution, as well as on search space $\mathcal{S}^{\mathrm{A}}$ using the TSP solver Concorde (Applegate et al., 2006). Despite several speedup techniques (Section 3.2), the search in space $\mathcal{S}^{\mathrm{A}}$ remained inefficient throughout our current experiments, especially for instances with a large number of customers per route. We therefore decided to focus on search space $\mathcal{S}_k^{\mathrm{B}}$, and devised several speedup techniques to enable its efficient exploration.

3.2. Efficient local search

To efficiently explore space $\mathcal{S}_k^{\mathrm{B}}$, we developed a local search algorithm which exploits *static neighborhood reductions, dynamic move filters, efficient memory structures* and *concatenation techniques*. Most of these techniques seek to limit the search effort in $\mathcal{S}_k^{\mathrm{B}}$. This resulting method, displayed in Algorithm 1, can be easily integrated in state-of-the-art metaheuristics for vehicle routing problems.

Neighborhood reductions. First of all, as in the majority of recent local search based metaheuristics for the CVRP, the neighborhood $\mathcal{N}(x^t)$ of each incumbent solution x^t is limited to moves that involve close vertices (Algorithm 1, Line 3). In particular, we use the classical intra-route and inter-route Relocate and Swap moves, for single vertices or generalized to pairs of consecutive vertices, as well as the 2-Opt and 2-Opt* moves. The resulting neighborhood contains a quadratic number of moves. As detailed in Vidal et al. (2013a), and in a similar way as Johnson and McGeoch (1997) and Toth and Vigo (2003), the search can be restricted to a subset of these moves that reconnect at least one vertex i with a vertex j belonging to the Γ closest vertices of i. The neighborhood size becomes $\mathcal{O}(\Gamma n)$, enabling a significant speedup for large-scale problem instances.

Dynamic move filters. To further restrict the search to promising moves, each move ϕ's feasibility is evaluated in $\mathcal{O}(1)$, in terms of capacity constraints, being discarded if it leads to an infeasible solution. The total cost $z(\phi(x^t))$ of the solution generated by ϕ prior to its optimization by the B&S decoder is evaluated subsequently. This cost represents an upper bound for the final cost of the move in $\mathcal{S}_k^{\mathrm{B}}$ after the application of the decoder. The move evaluation is pursued only if the solution cost has increased by a factor $1+\psi$ or less due to its application, that is, only if Condition (1) is satisfied. Otherwise, the move is discarded.

$$z(\phi(x^t)) \leq (1+\psi) \times z(x^t) \tag{1}$$

Parameter ψ plays an important role in defining how many moves are evaluated. The higher the value of ψ, the less pruning is induced by Eq. (1). Contrastingly, when $\psi = 0$ only immediately improving neighbors are evaluated. Defining a good value for ψ is non-trivial, given that it is an instance-dependent parameter. Since a fixed value would not suit instances with different sizes and characteristics, we suggest to use an adaptive parameter. The principle consists in adjusting ψ to ensure a target range $[\xi^-, \xi^+]$ for the fraction of filtered moves. After each 1000 move evaluations, the fraction ξ of filtered moves is collected and whenever it falls outside of the desired range, ψ is updated. If this fraction is too large, then ψ is increased by a multiplicative factor α. Conversely, if ξ is insufficient, then the parameter ψ is decreased:

$$\psi = \begin{cases} \psi \times \alpha & \text{if } \xi \leq \xi^-, \\ \psi \;/\; \alpha & \text{if } \xi \geq \xi^+, \\ \psi & \text{otherwise.} \end{cases} \tag{2}$$

Algorithm 1: Efficient local search in the space $\mathcal{S}_k^{\mathrm{B}}$.

```
   Input: An initial complete solution $x^0$, an evaluation threshold $\psi$ and a granularity threshold $\Gamma$
 1 $t \leftarrow 0$
 2 repeat
       // Enumerating $\mathcal{O}(\Gamma n)$ moves - candidate lists based on vertex proximity
 3     for each move $\phi(x^t) \in \mathcal{N}(x^t)$ involving a vertex pair $(i, j)$, $j \in \Gamma(i)$
 4         The move $\phi$ modifies up to two routes of $x^t$. Let $z_{\text{BEFORE}}$ be the sum of the costs of these two routes, and let $(\sigma_1^1, \dots, \sigma_{b_1}^1)$
           and $(\sigma_1^2, \dots, \sigma_{b_2}^2)$ be the new routes in $\phi(x^t)$.

           // Filter infeasible moves with respect to capacity constraints in $\mathcal{O}(1)$:
 5         if $Q(\sigma_1^1 \oplus \cdots \oplus \sigma_{b_1}^1) > Q$ or $Q(\sigma_1^2 \oplus \cdots \oplus \sigma_{b_2}^2) > Q$ then
 6             continue.

           // Consider the cost of the classical CVRP move to filter non-promising solutions in $\mathcal{O}(1)$:
 7         if $z(x^t) + C(\sigma_1^1 \oplus \cdots \oplus \sigma_{b_1}^1) + C(\sigma_1^2 \oplus \cdots \oplus \sigma_{b_2}^2) - z_{\text{BEFORE}} > (1+\psi) \times z(x^t)$ then
 8             continue.

           // Decode the routes $\sigma^1$ and $\sigma^2$ to evaluate the move $\phi$ in $\mathcal{S}_k^{\mathrm{B}}$:
 9         $z_{\text{MOVE}} \leftarrow 0$
10         for each route $\sigma^i$ with $i \in \{1, 2\}$
               // Compute hash key in $\mathcal{O}(1)$ and check memory in $\mathcal{O}(1)$:
11             $(\bar{\sigma}^i, \bar{z}_i) \leftarrow$ LOOKUP$(H(\sigma_1^i \oplus \cdots \oplus \sigma_{b_i}^i))$
               // If not in memory, repeatedly apply B&S dynamic programming until the route becomes B$^k$-optimal:
12             if $(\bar{\sigma}^i, \bar{z}_i) =$ NOT FOUND then
13                 $(\bar{\sigma}^i, \bar{z}_i) \leftarrow$ BALAS-SIMONETTI$(\sigma_1^i \oplus \cdots \oplus \sigma_{b_i}^i)$
14                 STORE$((\bar{\sigma}^i, \bar{z}_i), H(\sigma_1^i \oplus \cdots \oplus \sigma_{b_i}^i))$
15             $z_{\text{MOVE}} \leftarrow z_{\text{MOVE}} + \bar{z}_i$

           // Filter non-improving moves:
16         if $z_{\text{MOVE}} \geq z_{\text{BEFORE}}$ then
17             continue.

           // At this stage, apply $\phi$ since it is an improving move in $\mathcal{S}_k^{\mathrm{B}}$:
18         Set $x^{t+1} = \phi(x)$ ; $t = t + 1$
19         Replace the routes $(\sigma^1,\sigma^2)$ by $(\bar{\sigma}^1,\bar{\sigma}^2)$ in $x^{t+1}$
20 until $x^t$ is a local minimum
21 return $x^t$
```

Global memory. The B&S algorithm, used as a decoder, requires a computational effort which grows linearly with the route size and exponentially with parameter k. It is thus essential to restrain the use of this procedure to a strict minimum and avoid decoding twice the same route over the course of the search. To that end, we rely on a global memory to store the routes that have been decoded, avoiding recalculations (Lines 12–14). We use a hashtable for this task, since it allows $\mathcal{O}(1)$ queries given the hash key associated with a route.

Two important aspects should be discussed. First, since the available memory space is finite, some strategy is necessary to limit the memory size in case of an excessive space consumption. To that end, we define an upper bound $\mathcal{M}^{\max}$ on the number of routes stored in the memory and eliminate half of the entries, those used with less frequency, whenever this limit is attained.

The second aspect to be discussed concerns the effort spent querying the memory. It is well known that local searches for the CVRP evaluate millions of moves, and that constant-time move evaluations are essential for a good performance. Capacity checks (Line 5) and simple distance computations (Line 7) can be easily achieved in $\mathcal{O}(1)$ using incremental move evaluations or concatenation techniques (Vidal et al., 2014). Moreover, querying the memory for a given route supposes the availability of a hash index which characterizes the associated sequence of visits, but a direct approach that sweeps through the route to compute this index already takes $\mathcal{O}(n)$ time. To avoid this bottleneck, we employ specific hash functions and calculation techniques based, again, on concatenations in $\mathcal{O}(1)$. These concepts are discussed in the following section.

3.3. Constant-time evaluations

We use the concatenation strategy of Vidal et al. (2014, 2015b) to perform efficient cost- and load-feasibility evaluations. This strategy exploits the fact that any route obtained from a classical move $\phi(x^t)$ on an incumbent solution x^t corresponds to a recombination of a bounded number of (customer and depot) visit sequences of x^t. As such, the new routes can be expressed as a concatenation of sequences $\sigma_1 \oplus \cdots \oplus \sigma_b$. We also extend this approach to enable $\mathcal{O}(1)$ computations of hash keys.

To efficiently evaluate the cost, load, and compute the hash keys, we perform a preliminary preprocessing on the $\mathcal{O}(n^2)$ subsequences of consecutive visits which compose the solution x^t. Four quantities are calculated: the total demand $Q(\sigma)$ of a sequence σ, its distance $C(\sigma)$, and its hash keys $H^p(\sigma)$ and $H^s(\sigma)$. For a sequence $\bar{\sigma} = [i]$ containing a single visit i with demand q_i, $Q(\bar{\sigma}) = q_i$, $C(\bar{\sigma}) = 0$, $H^p(\bar{\sigma}) = \rho \times i$ and $H^s(\bar{\sigma}) = \rho^i$, where ρ is a prime number. Moreover, Eqs. (3)–(5) extend these quantities, by induction, for any sequence of visits $\sigma_1 \oplus \sigma_2$ expressed as the concatenation of two sequences σ_1 and σ_2. In these equations, d_{ij} expresses the distance between visits i and j.

$$Q(\sigma^1 \oplus \sigma^2) = Q(\sigma^1) + Q(\sigma^2) \tag{3}$$

$$C(\sigma^1 \oplus \sigma^2) = C(\sigma^1) + d_{\sigma^1(|\sigma^1|),\sigma^2(1)} + C(\sigma^2) \tag{4}$$

$$H^p(\sigma^1 \oplus \sigma^2) = H^p(\sigma^1) + \rho^{|\sigma_1|} \times H^p(\sigma^2) \tag{5}$$

$$H^s(\sigma^1 \oplus \sigma^2) = H^s(\sigma^1) + H^s(\sigma^2). \tag{6}$$

As in Vidal et al. (2014), Eq. (3)–(6) are first employed iteratively, in lexicographic order, to obtain information concerning all sequences during the preprocessing phase. Afterwards, the same equations are used for move evaluations. Since any route obtained from a classical move corresponds to the concatenation of a bounded number of sequences, it is possible to obtain the associated load, distance, and hash keys by applying these equations a limited number of times. Then, the information on subsequences is updated every time an improving move is applied, a rare occurrence in comparison to the number of moves evaluated.

The two hash functions defined in Eq. (5) and (6) are employed together as a means of reducing chances of two distinct sequences having identical hashes. The function H^p is a multiplicative hash which depends on the visit permutation (Knuth, 1973). Note that, when implementing such a function, the values ρ^i must be precomputed and bounded (taking the rest of the integer division by a large number) to prevent overflow during multiplication. The second function H^s is an additive hash which only depends on the set of visited customers, and not on the visit sequence. These hash functions are easily recognized when reformulated as follows:

$$H^p(\sigma) = \sum_{i=1}^{|\sigma|} \rho^i \times \sigma_i \tag{7}$$

$$H^s(\sigma) = \sum_{i=1}^{|\sigma|} \rho^{\sigma_i}. \tag{8}$$

These functions fit well our purposes due to their inductive definition based on the concatenation operation. They are employed, along with the route distance and its number of visits, to verify a correct match in the memory in $\mathcal{O}(1)$ without a complete route comparison in $\mathcal{O}(n)$. To minimize the risk of two routes having identical hashes, we duplicated these hash functions with different values for ρ, leading to four hash values overall. In the first case, ρ is set to the smallest prime number greater than the number of customers. In the second case, ρ is set to 31 (multiplier of Kernighan and Ritchie, 1988). Despite this strategy, a tiny chance of false positives remains. However, no false positive was registered within our computational experiments considering multiple runs on 100 different instances.

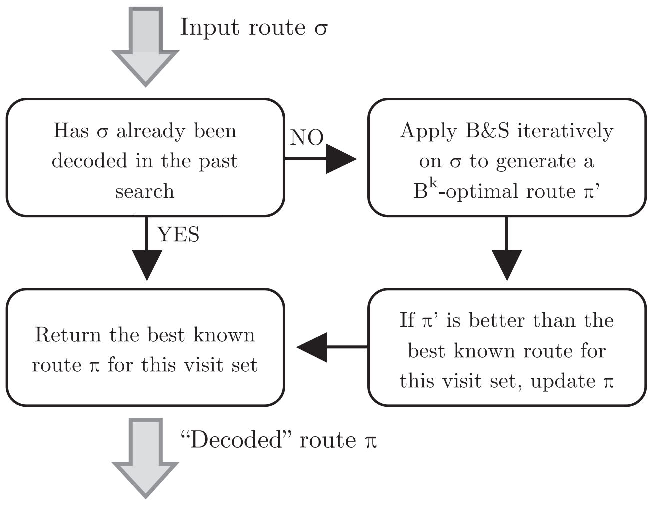

Input route σ
Has σ already been decoded in the past search
NO
Apply B&S iteratively on σ to generate a B^k-optimal route π'
YES
Return the best known route π for this visit set
If π' is better than the best known route for this visit set, update π
"Decoded" route π

Fig. 5. Tunneling strategy and solution decoding.

3.4. Reshaping the search space – tunneling strategy

Until now, the purpose of the global memory has been focused on saving computational effort by avoiding duplicate calls to the B&S algorithm. Yet, as shown in the following, this structure can be exploited to a larger extent to promote the discovery of good solutions.

Consider two input routes σ_1 and σ_2 indexed in the order of their appearance in the local search, and representing the same set of customer visits. The route σ_1 is first saved in the global memory along with its associated B^k–optimal route of cost z_1. Subsequently, the LS considers σ_2 without finding a match in the memory, triggering a new execution of the B&S algorithm and leading to a cost z_2. if $z_2 > z_1$, then the algorithm has failed to recognize that a better TSP tour has been found in prior search for the same customers.

To improve the behavior of the algorithm in such situations, we introduce a guidance mechanism called *tunneling*. Guidance techniques are a set of strategies which analyze and exploit the search history to direct the search towards promising or unexplored regions of the search space (Crainic and Toulouse, 2008). Our method works as follows: every route σ issued from a non-filtered LS move and absent from the memory is decoded by the B&S algorithm; yet, rather than directly returning the output of B&S, the algorithm finds and returns the best known permutation of the visits for this customer set, found over previous B&S executions. The goal of this strategy is to intensify the search around known high-quality tours without jeopardizing the discovery of better route configurations. It can be efficiently implemented with a refinement of the hashtable-based memory structure, by grouping the routes into different buckets according to their visit set, and using the additive hash function of Eq. (6) for $\mathcal{O}(1)$ queries. Fig. 5 summarizes this process.

This tunneling strategy has a significant impact on the search space. Initially, as the search starts, the algorithm explores the space $\mathcal{S}_k^{\mathrm{B}}$. Then, as the search progresses, the memory starts to be filled, and the algorithm re-introduces more and more frequently the best known routes in its solutions. In a hypothetical situation where all feasible routes have already been memorized (hypothetical due to the needed exponential memory size), the TSP–optimal routes would be systematically returned, and the algorithm behaves as if it was searching in $\mathcal{S}^{\mathrm{A}}$. With this limit case in mind, the tunneling strategy contributes to reshape the space $\mathcal{S}_k^{\mathrm{B}}$ into $\mathcal{S}^{\mathrm{A}}$ as the search progresses. Moreover, note that this strategy re-

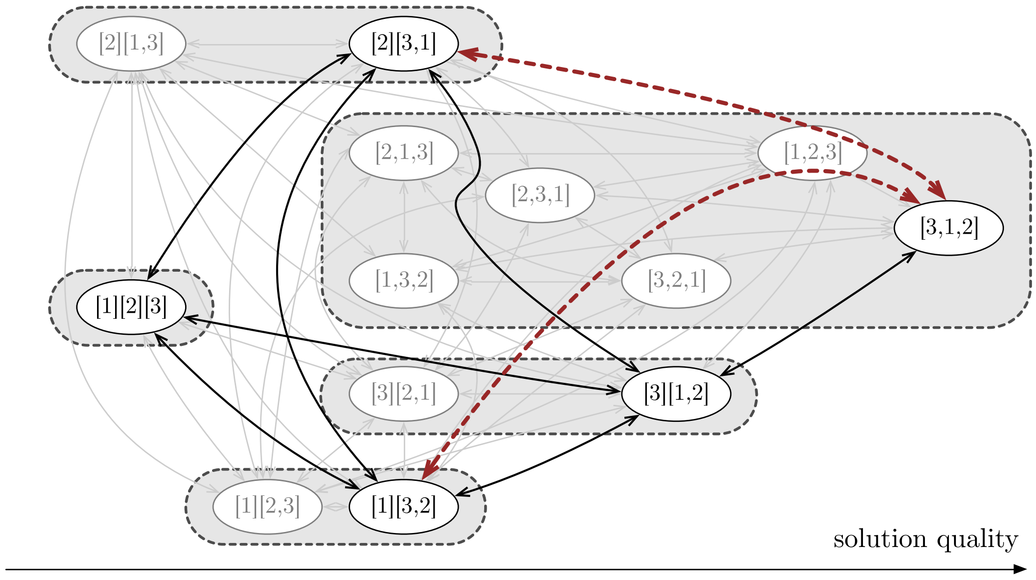


Fig. 6. Search space $\mathcal{S}_1^{\text{B}}$ with the tunneling search strategy, after discovering the route [3,1,2].

mains fully relevant for a classical neighborhood search, even without B&S decoder.

Consider the same example as depicted in Section 3.1 (Figs. 2–4). Suppose that the route [3,2,1] has been identified in the past search along with its associated B^k–optimal tour [3,1,2] and that the tunneling strategy is employed. Fig. 6 illustrates the resulting search space: all solutions that include '[1,2,3]' in their neighborhood now point towards solution '[3,1,2]' instead, which has identical customer-to-vehicle Assignments but a lower cost due to better Sequencing decisions. With only one route in memory, the resulting search space already becomes equivalent to $\mathcal{S}^{\text{A}}$.

4. Computational experiments

We conducted extensive computational analyses to measure the benefits of a search in $\mathcal{S}^{\text{A}}$ and $\mathcal{S}_k^{\text{B}}$, the impact of the tunneling strategy and move evaluation filters. For these tests, we considered the simple local search (LS) described in Section 3, as well as a more advanced metaheuristic, the UHGS of Vidal et al. (2012, 2014). In the LS, all moves are evaluated in random order and a "first-improvement" policy is used (any improving move is directly applied). UHGS is an evolutionary algorithm which iteratively generates new individuals by means of crossover and local search operations. To promote a diversified search, it relies on a "biased fitness" measure, which evaluates the quality of each solution based on its cost and similarity to other solutions in the population. UHGS was adapted by replacing its native local search by the proposed method. This method extension will be referred as UHGS-BS. Except this modification, all other parameters and procedures of UHGS-BS remain the same as in the original article: the neighborhood restriction parameter is maintained to $\Gamma = 20$, and the termination criterion is fixed to $It_{\text{MAX}} = 20,000$ consecutive iterations without improvement.

A single local search requires only a limited computational effort. Thus, the first analyses (Section 4.1) on the performance of the LS in $\mathcal{S}^{\text{A}}$ and $\mathcal{S}_k^{\text{B}}$ could be done for multiple values of the range parameter k. It was also possible to evaluate the impact of k without the interference of dynamic move filters and tunneling techniques. Since UHGS-BS performs a more extensive search with multiple local search runs, our analyses with this method (Section 4.2) are focused on a smaller set of values for the range parameter k, in the presence of dynamic move filters.

All algorithms were implemented in C++ and executed on a single thread of an Intel(R) Xeon(R) E5-2680v3 CPU. A RAM limit of 8 GB was imposed for each run. To solve the TSP problems when considering the space $\mathcal{S}^{\text{A}}$, we used the Concorde solver (Applegate et al., 2006). To conduct the experiments with the UHGS-BS, we used the code base made available at https://github.com/vidalthi/HGS-CARP, from Vidal (2017).

4.1. Preliminary experiments with a simple local search

In a first experiment, we tested the local search of Algorithm 1 on spaces $\mathcal{S}^{\text{A}}$ and $\mathcal{S}_k^{\text{B}}$ for $k \in \{0, \dots, 9\}$, to observe the growth of its computational time as a function of the parameter k, and identify a range of values over which the approach remains practical. As an initial solution, we used the result of the *savings* algorithm of Clarke and Wright (1964). We set $\psi = \infty$ in order to observe the results without the interference of move filters.

We considered the 100 recent benchmark instances of Uchoa et al. (2017), as these instances remain highly challenging for metaheuristics and cover a larger variety of instance size and characteristics: demand and customer distribution, depot location, and route length. For each instance, we ran the methods 20 times (using the same set of 20 random seeds for all instances and methods). The results are summarized in Fig. 7, in the form of boxplots. The leftmost graph represents the percentage gap in terms of solution quality, relative to that of the best known solution (BKS) collected from Uchoa et al. (2017): Gap $= 100 \times (z - z_{\text{BKS}})/z_{\text{BKS}}$, where z is the solution value of the method and z_{BKS} is the BKS value. The rightmost graph represents the CPU time of the method, using a logarithmic scale.

As illustrated by these experiments, the search in space $\mathcal{S}^{\text{A}}$ visibly leads to solutions of higher average quality, albeit in a CPU time largely greater than that of a classical local search in $\mathcal{S}$. The solution quality of a search in the space $\mathcal{S}_k^{\text{B}}$ consistently increases as k grows, along with the needed CPU time. When $k = 0$, the algorithm behaves as a classical local search in $\mathcal{S}$. When k is large, the method becomes more similar to a search in $\mathcal{S}^{\text{A}}$. A difference of solution quality can still be noticed between $\mathcal{S}_9^{\text{B}}$ and $\mathcal{S}^{\text{A}}$, due to some instances containing up to 25 deliveries per route.

Each value of the range parameter k establishes a trade-off between computational effort and solution quality. The computational time of the local search does not exceed two seconds when

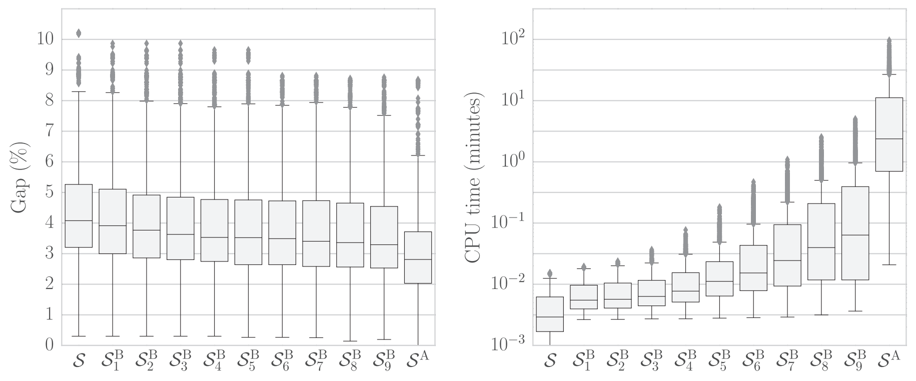
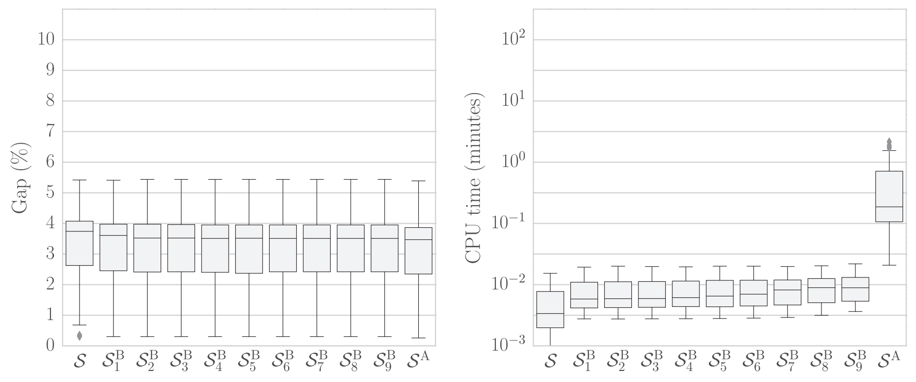


Fig. 7. Solution quality and CPU time of the LS, depending on the search space.

(a) Instances with average route cardinality in range [3.0, 4.55]:



(b) Instances with average route cardinality in range [16.47, 24.43]:

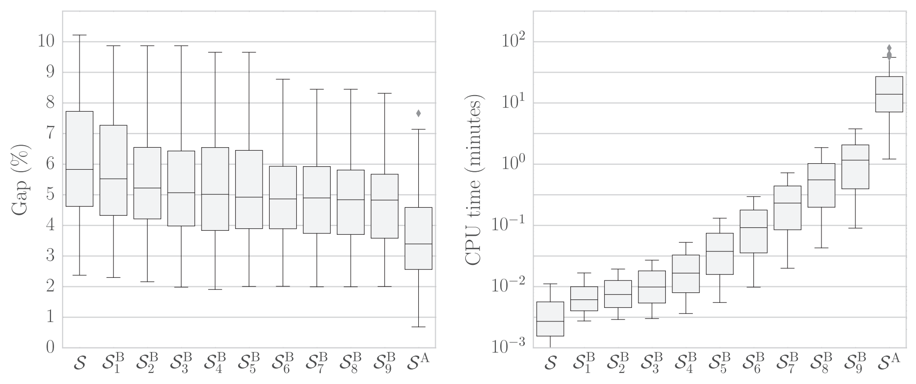


Fig. 8. Results (solution quality and runtime) of local search on different search spaces for instances with different route cardinalities.

exploring $\mathcal{S}_k^B$ with $k \leq 2$, but it culminates to 1000 seconds on the largest instances when exploring $\mathcal{S}^A$. Clearly, the additional CPU time required by Concorde is not compatible with the repeated application of local searches within state-of-the-art metaheuristic algorithms, such that we should concentrate our efforts on the exploration of the space $\mathcal{S}_k^B$ with moderate values of k.

Moreover, a careful analysis of these results on subsets of instances with a different average number of customers per route gives additional insights. This analysis is reported in Fig. 8, considering the 20 instances with smallest average route cardinality, in the range [3.0,4.55], and the 20 instances with largest average route cardinality, in [16.47,24.43].

As illustrated in Fig. 8, the benefits of a search in $\mathcal{S}_k^B$ or $\mathcal{S}^A$ are small for instances with a small number of customer visits per route. In particular, all runs with $k \geq 4$ lead to a similar solution quality and CPU time. This is due to the fact that B&S does an exact TSP optimization when k is greater or equal to the route cardinality. In contrast, the benefits in terms of solution quality are larger on instances with a high number of customer visits per vehicle. We observe a significant improvement of the solutions when k varies from 0 to 4, from an average gap of 6.15% down to 5.25%. Subsequently, as k increases beyond 4, the rate of improvement is smaller. Increasing k up to the maximum route size would still be beneficial, but impracticable in terms of CPU time.

4.2. Experiments with UHGS-BS – range parameter, move filters and tunneling

As viewed in the previous section, a local search in the space $\mathcal{S}_k^B$ can lead to solutions of better quality than a search in $\mathcal{S}$, at the expense of a higher computational effort. Still, even if solution improvements were observed for simple local searches, it is an open question whether the inclusion of these extended search

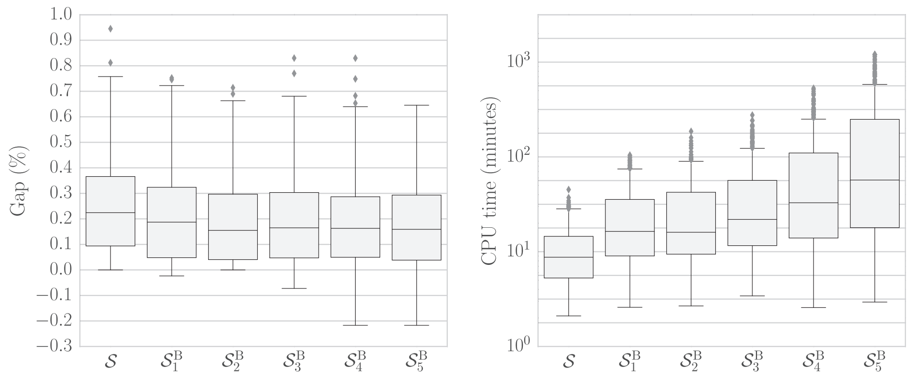


Fig. 9. Results (solution quality and runtime) of UHGS-BS for different cache strategies and k values.

procedures into state-of-the-art metaheuristics translates back into significant quality improvements.

This section now analyses the performance of the UHGS-BS metaheuristic, considering $k \in \{1, \ldots, 5\}$ in combination with dynamic move filters. The move filters are designed to eliminate a large fraction of complete move evaluations, such that a larger computational effort can be spent in the evaluation of each remaining move without significantly impacting the overall CPU time of the method.

After preliminary experiments, we observed that setting $k = 2$ and $[\xi^-, \xi^+] = [90\%, 95\%]$ establishes a good compromise between solution quality and computational effort. This configuration, without tunneling strategy, was used as baseline for the experiments of this section. We then modified each parameter and design choice, in turn, to investigate its impact. This experimentation was restricted to a subset of 40 instances with a number of customer visits $n \in \{195, 411\}$, as these instances require limited CPU time and remain challenging for state-of-the-art metaheuristics. For each instance, ten runs have been conducted with different seeds.

Impact of the range parameter. Fig. 9 compares the performance, in terms of average percentage gap and CPU time, of the classical UHGS with that of its extended version searching in $\mathcal{S}_k^B$, for $k \in \{1, \ldots, 5\}$. As in previous figures, the results are presented in the form of boxplots.

As expected, the gaps obtained with the variants of UHGS-BS are much smaller than those of simple local searches, due to the better exploration capabilities of the method. Average gaps range from 0.25% when exploring the classical search space $\mathcal{S}$, to 0.18% when exploring $\mathcal{S}_5^B$. As highlighted by pairwise Wilcoxon tests, statistically significant differences exist between the results of $\mathcal{S}$, $\mathcal{S}_1^B$ and $\mathcal{S}_2^B$ (p-values < 0.05). A *decreasing returns* effect can also be observed; the difference of quality between the solutions obtained by $\mathcal{S}$ and $\mathcal{S}_1^B$ is larger than between $\mathcal{S}_1^B$ by $\mathcal{S}_2^B$, which is turn is larger than between $\mathcal{S}_2^B$ by $\mathcal{S}_3^B$, and so on. The configuration $\mathcal{S}_2^B$, in particular, achieves a good trade-off between quality and search effort. As demonstrated by the outliers with negative gap in the figure, this configuration has led to new best known solutions for surprisingly small instances with 256 and 294 customers, on which hundreds of test runs had been conducted in the past. This is an indication that the search in $\mathcal{S}_k^B$ has the potential to lead to structurally different solutions, which were not attained with more classical searches. The main challenge therefore is to harness this ability without sacrificing too much computational effort.

Impact of the dynamic move filter. Fig. 10 investigates the impact of different target intervals $[\xi^-, \xi^+]$ (desired quantity of filtered moves – Section 3.2) for the dynamic move filters. The range parameter remains fixed to $k = 2$. It also indicates the results obtained when filtering *all* non-improving moves ($\psi = 0$), which is equivalent to using B&S only as a post-optimization procedure, after the discovery of each improving move.

These experiments demonstrate that move filters have a large incidence on the solution quality. These filters are, however, essential to maintain a low computational effort. In particular, filtering all non-improving moves prior to the evaluation of B&S ($\psi = 0$) leads to an average gap of 0.26%, compared to 0.21% when setting $[\xi^-, \xi^+] = [90\%, 95\%]$ as a target for the dynamic move filter and evaluating the non-filtered moves in combination with B&S. This validates an important hypothesis explored in this article: many moves that are usually discarded in regular local search methods and metaheuristics can lead to improved solutions when applied in combination with a route optimization procedure. Naturally, this capability goes along with an increased CPU time. Nonetheless, by an adequate calibration of the move filters, the total CPU time dedicated to B&S can be restricted enough to not intervene as a bottleneck. This is visible by the results of configuration $[\xi^-, \xi^+] = [90\%, 95\%]$, which on average used no more than twice the time of $\psi = 0$. For the remainder of these analyses, we selected this configuration, which establishes a good balance between the exploitation of the capabilities of B&S and the computational effort.

Impact of the tunneling strategy. Finally, Table 1 compares the performance of UHGS-BS without and with the tunneling strategy. The range parameter has been set to $k = 2$, and $[\xi^-, \xi^+] = [90\%, 95\%]$. The leftmost group of columns reports the instance name, number of customers n average number of visits per route in the BKS for each instance. Then, the next columns present, for each approach, the average solution value over ten runs, best solution value, average CPU time, percentage of routes which have been successfully queried from the global memory without a re-evaluation, number of executions of the B&S optimization procedure, total number of iterations.

The tunneling strategy leads to an average Gap(%) of 0.15%, compared to 0.17% without tunneling. Although this is only a difference of 0.02%, reducing the gap indeed becomes harder as the solution quality approaches known BKS and optimal solutions. This improvement in solution quality also comes with a reduction of the overall CPU time, as the tunneling stimulates a faster convergence towards local minima and allows a better management of the global memory, by storing at most one route per customer set. As a consequence, the chances of successful queries in the memory is sensibly higher (81.5% compared to 80.2% on average), thereby reducing on average the number of B&S executions as well as the total number of iterations. Based on these observations, tunneling is beneficial without any other visible counterpart. We will use this mechanism for the final tests on the complete set of benchmark instances, in the next section.

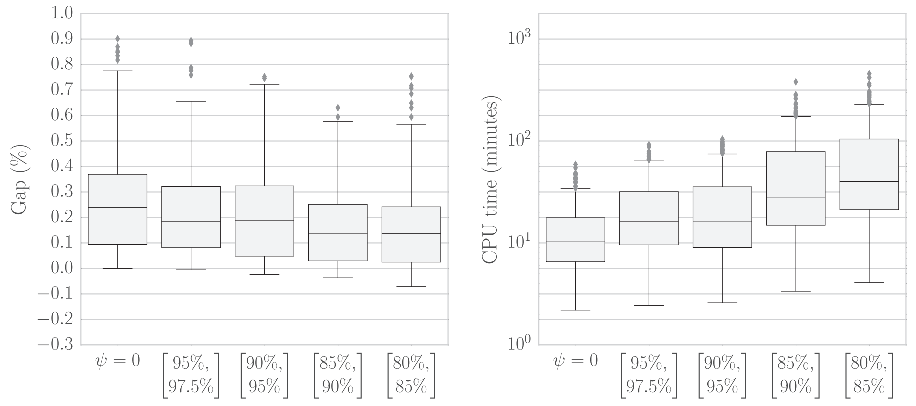


Fig. 10. Results (solution quality and runtime) of UHGS-BS for different $[\xi^-, \xi^+]$ values.

Table 1
Impact of the tunneling strategy for 40 medium-size instances.

#	Instance	Without tunneling strategy						With tunneling strategy					
		Time	Avg.	Best	Cache	B&S	Iters	Time	Avg.	Best	Cache	B&S	Iters
21	X-n195-k51	3.8	44,289.5	**44,225**	60.6%	1.4E+08	34,577	5.2	**44,272.6**	**44,225**	60.2%	1.8E+08	41,713
22	X-n200-k36	5.3	**58,618.5**	**58,578**	97.8%	1.1E+07	43,664	6.2	58,625.7	**58,578**	97.8%	1.0E+07	41,788
23	X-n204-k19	5.8	19,569.0	**19,565**	91.0%	2.7E+07	31,315	6.6	**19,568.5**	**19,565**	92.6%	2.2E+07	31,118
24	X-n209-k16	15.5	**30,673.1**	**30,656**	77.6%	1.1E+08	35,319	17.8	30,676.8	**30,656**	81.0%	9.1E+07	36,174
25	X-n214-k11	34.8	10,873.2	**10,856**	70.5%	2.1E+08	48,054	32.3	**10,872.0**	**10,856**	76.2%	1.5E+08	46,133
26	X-n219-k73	12.9	**117,602.7**	**117,595**	3.7%	5.0E+08	30,949	11.0	117,603.8	**117,595**	3.6%	5.0E+08	30,773
27	X-n223-k34	6.2	40,490.3	**40,437**	95.4%	2.3E+07	38,698	7.7	**40,489.0**	**40,437**	95.7%	2.1E+07	39,268
28	X-n228-k23	14.5	25,784.1	**25,743**	87.9%	8.5E+07	48,303	14.6	**25,779.5**	**25,743**	88.6%	6.9E+07	42,435
29	X-n233-k16	10.9	**19,305.3**	**19,230**	87.1%	4.7E+07	31,274	9.5	19,309.0	**19,230**	90.4%	3.2E+07	29,652
30	X-n237-k14	16.6	27,053.8	27,050	75.9%	9.7E+07	26,927	17.5	**27,047.2**	**27,042**	81.1%	8.1E+07	30,294
31	X-n242-k48	8.7	82,922.0	**82,792**	94.3%	4.4E+07	49,734	12.9	**82,919.6**	**82,792**	93.6%	6.6E+07	62,591
32	X-n247-k47	14.6	**37,380.7**	**37,278**	49.2%	6.1E+08	63,084	13.7	37,393.1	**37,281**	49.1%	5.1E+08	52,872
33	X-n251-k28	12.7	**38,767.4**	**38,699**	90.8%	6.4E+07	48,107	11.4	38,793.5	**38,699**	91.2%	4.5E+07	35,265
34	X-n256-k16	8.8	18,880.0	18,880	85.9%	4.3E+07	22,991	9.4	**18,875.9**	**18,839**	89.6%	3.3E+07	24,930
35	X-n261-k13	43.0	26,628.7	**26,579**	71.9%	2.3E+08	46,952	38.9	**26,620.5**	**26,586**	75.1%	1.7E+08	39,770
36	X-n266-k58	14.3	**75,703.6**	**75,558**	98.9%	1.5E+07	69,013	17.8	75,737.9	**75,646**	98.9%	1.5E+07	72,435
37	X-n270-k35	6.6	**35,310.5**	**35,303**	96.8%	1.5E+07	34,796	8.5	35,318.5	**35,303**	97.1%	1.4E+07	35,654
38	X-n275-k28	11.7	21,256.9	**21,245**	89.0%	5.9E+07	36,470	12.8	**21,255.0**	**21,245**	91.0%	4.4E+07	34,488
39	X-n280-k17	69.1	33,614.0	33,593	66.7%	4.1E+08	53,528	86.3	**33,600.8**	**33,584**	70.1%	4.3E+08	63,405
40	X-n284-k15	82.2	20,306.0	20,255	70.0%	4.3E+08	73,605	67.4	**20,283.3**	**20,238**	73.9%	3.0E+08	61,772
41	X-n289-k60	18.3	95,501.4	95,395	69.6%	4.5E+08	75,465	20.7	**95,475.6**	**95,245**	69.5%	4.4E+08	72,940
42	X-n294-k50	9.3	**47,272.9**	47,240	96.8%	2.6E+07	47,162	11.6	47,289.7	**47,239**	96.3%	3.3E+07	53,125
43	X-n298-k31	7.1	34,282.8	**34,231**	94.6%	2.3E+07	27,625	8.1	**34,280.8**	**34,231**	95.2%	2.0E+07	26,203
44	X-n303-k21	22.3	21,839.1	**21,744**	86.7%	1.0E+08	43,974	21.1	**21,833.3**	**21,744**	88.9%	7.9E+07	39,415
45	X-n308-k13	73.6	**25,893.2**	**25,864**	68.5%	3.3E+08	48,194	63.6	25,911.8	**25,866**	71.8%	2.6E+08	42,804
46	X-n313-k71	15.8	**94,270.5**	**94,169**	57.8%	4.7E+08	54,350	16.8	94,289.7	**94,192**	57.6%	4.9E+08	58,766
47	X-n317-k53	29.7	**78,389.3**	**78,372**	97.5%	3.3E+07	60,402	31.6	78,405.0	**78,380**	97.7%	3.2E+07	65,092
48	X-n322-k28	10.9	29,918.8	29,880	93.3%	3.7E+07	36,519	17.4	**29,894.6**	**29,834**	93.9%	4.3E+07	48,316
49	X-n327-k20	31.2	27,618.2	**27,560**	81.3%	1.5E+08	44,348	35.5	**27,591.7**	**27,560**	83.4%	1.5E+08	47,634
50	X-n331-k15	71.6	31,138.6	**31,103**	65.2%	3.7E+08	47,298	63.2	**31,128.3**	**31,103**	67.3%	3.0E+08	41,354
51	X-n336-k84	34.3	**139,522.2**	**139,303**	39.0%	1.4E+09	86,713	40.9	139,572.7	**139,339**	39.7%	1.4E+09	89,964
52	X-n344-k43	11.7	42,161.9	42,086	97.1%	2.3E+07	42,768	22.7	**42,136.7**	**42,066**	97.4%	3.0E+07	65,989
53	X-n351-k40	26.1	26,010.1	25,958	94.6%	7.9E+07	72,726	32.8	**25,991.2**	**25,947**	94.7%	8.3E+07	76,997
54	X-n359-k29	65.3	**51,676.6**	51,608	84.9%	3.1E+08	84,963	56.1	51,684.5	**51,569**	85.9%	2.4E+08	67,011
55	X-n367-k17	92.0	**22,870.1**	**22,814**	72.2%	4.0E+08	60,550	66.8	22,894.6	**22,814**	71.1%	2.8E+08	39,572
56	X-n376-k94	61.3	**147,736.8**	**147,718**	99.2%	1.6E+07	64,604	53.1	147,738.1	**147,718**	99.2%	1.4E+07	61,378
57	X-n384-k52	26.0	66,254.7	**66,095**	97.7%	4.2E+07	77,380	29.1	**66,184.1**	**66,133**	97.6%	3.8E+07	68,679
58	X-n393-k38	22.7	38,361.0	38,269	92.7%	7.2E+07	47,398	26.9	**38,305.5**	**38,260**	93.7%	6.5E+07	48,965
59	X-n401-k29	79.6	66,357.8	66,269	83.9%	3.4E+08	86,787	69.7	**66,347.6**	**66,225**	86.1%	2.7E+08	78,713
60	X-n411-k19	110.9	19,753.4	19,725	72.8%	4.9E+08	65,729	74.8	**19,748.3**	**19,719**	77.9%	2.9E+08	51,034
	Average values:	30.4	0.17%	0.02%	80.2%	2.1E+08	51,058	**29.3**	**0.15%**	**0.01%**	81.5%	1.8E+08	49,912

4.3. Comparison with recent state-of-the-art algorithms

Finally, this section reports detailed results of UHGS-BS, using the baseline configuration and the tunneling strategy, on the complete set of 100 instances proposed by Uchoa et al. (2017). The results of UHGS-BS are compared to that of the current state-of-the-art algorithms: the hybrid Iterated Local Search (ILS) proposed by Subramanian et al. (2013), and the original UHGS of Vidal et al. (2012, 2014), which were executed 50 times for each instance. To keep the total computational effort within reasonable limits, UHGS-BS was executed 10 times for each instance. The maximum number of consecutive iterations without improvements was set to $It_{\text{MAX}} = 50,000$ to evaluate UHGS-BS in the same conditions as UHGS on this set of instances (Uchoa et al., 2017). A hard run-time limit of 24 hours was imposed for each run. Based on the single-thread Passmark benchmark (Software, 2018), our CPU is ap-

Table 2
Results for the instances with up to 331 customers from Uchoa et al. (2017).

#	Instance	ILS			UHGS			UHGS-BS			
		Time	Average	Best	Time	Average	Best	Time†	Average	Best	
1	X-n101-k25	0.1	**27,591.0**	**27,591**	1.4	**27,591.0**	**27,591**	3.1	**27,591.0**	**27,591**	
2	X-n106-k14	2.0	26,375.9	**26,362**	4.0	26,381.8	26,378	23.3	**26,374.3**	**26,362**	
3	X-n110-k13	0.2	**14,971.0**	**14,971**	1.6	**14,971.0**	**14,971**	5.2	**14,971.0**	**14,971**	
4	X-n115-k10	0.2	**12,747.0**	**12,747**	1.8	**12,747.0**	**12,747**	8.5	**12,747.0**	**12,747**	
5	X-n120-k6	1.7	13,337.6	**13,332**	2.3	**13,332.0**	**13,332**	51.0	**13,332.0**	**13,332**	
6	X-n125-k30	1.4	55,673.8	**55,539**	2.7	55,542.1	**55,539**	8.1	**55,540.0**	**55,539**	
7	X-n129-k18	1.9	28,998.0	28,948	2.7	28,948.5	**28,940**	11.2	**28,940.0**	**28,940**	
8	X-n134-k13	2.1	10,947.4	**10,916**	3.3	10,934.9	**10,916**	26.9	**10,916.0**	**10,916**	
9	X-n139-k10	1.6	13,603.1	**13,590**	2.3	**13,590.0**	**13,590**	11.8	**13,590.0**	**13,590**	
10	X-n143-k7	1.6	15,745.2	15,726	3.1	15,700.2	**15,700**	44.1	**15,700.0**	**15,700**	
11	X-n148-k46	0.8	43,452.1	**43,448**	3.2	**43,448.0**	**43,448**	6.7	**43,448.0**	**43,448**	
12	X-n153-k22	0.5	21,400.0	21,340	5.5	21,226.3	**21,220**	20.2	**21,225.6**	21,225	
13	X-n157-k13	0.8	**16,876.0**	**16,876**	3.2	**16,876.0**	**16,876**	37.8	**16,876.0**	**16,876**	
14	X-n162-k11	0.5	14,160.1	**14,138**	3.3	14,141.3	**14,138**	16.2	**14,138.0**	**14,138**	
15	X-n167-k10	0.9	20,608.7	20,562	3.7	20,563.2	**20,557**	48.0	**20,557.0**	**20,557**	
16	X-n172-k51	0.6	45,616.1	**45,607**	3.8	**45,607.0**	**45,607**	7.5	**45,607.0**	**45,607**	
17	X-n176-k26	1.1	48,249.8	48,140	7.6	47,957.2	**47,812**	21.5	**47,830.7**	**47,812**	
18	X-n181-k23	1.6	25,571.5	**25,569**	6.3	25,591.1	**25,569**	18.5	**25,569.4**	**25,569**	
19	X-n186-k15	1.7	24,186.0	**24,145**	5.9	24,147.2	**24,145**	26.9	**24,145.0**	**24,145**	
20	X-n190-k8	2.1	17,143.1	17,085	12.1	16,987.9	**16,980**	215.4	**16,985.3**	**16,980**	
21	X-n195-k51	0.9	**44,234.3**	**44,225**	6.1	44,244.1	**44,225**	12.4	44,283.8	**44,225**	
22	X-n200-k36	7.5	58,697.2	58,626	8.0	58,626.4	**58,578**	15.9	**58,615.1**	**58,578**	
23	X-n204-k19	1.1	19,625.2	19,570	5.4	19,571.5	**19,565**	20.9	**19,567.0**	**19,565**	
24	X-n209-k16	3.8	30,765.4	30,667	8.6	30,680.4	**30,656**	47.5	**30,671.3**	**30,656**	
25	X-n214-k11	2.3	11,126.9	10,985	10.2	10,877.4	**10,856**	69.6	**10,872.1**	**10,856**	
26	X-n219-k73	0.9	**117,595.0**	**117,595**	7.7	117,604.9	**117,595**	24.2	117,600.5	**117,595**	
27	X-n223-k34	8.5	40,533.5	40,471	8.3	40,499.0	**40,437**	24.7	**40,478.4**	**40,437**	
28	X-n228-k23	2.4	25,795.8	25,743	9.8	25,779.3	**25,742**	38.6	**25,768.0**	25,743	
29	X-n233-k16	3.0	19,336.7	19,266	6.8	19,288.4	**19,230**	45.3	**19,276.5**	**19,230**	
30	X-n237-k14	3.5	27,078.8	**27,042**	8.9	27,067.3	**27,042**	42.9	**27,048.8**	**27,042**	
31	X-n242-k48	17.8	**82,874.2**	82,774	12.4	82,948.7	82,804	24.1	82,920.9	**82,751**	
32	X-n247-k47	2.1	37,507.2	37,289	20.4	**37,284.4**	**37,274**	36.8	37,388.9	**37,274**	
33	X-n251-k28	10.8	38,840.0	38,727	11.7	38,796.4	38,699	26.9	**38,778.7**	**38,684**	
34	X-n256-k16	2.0	18,883.9	18,880	6.5	18,880.0	18,880	30.6	**18,867.7**	**18,839**	⊛
35	X-n261-k13	6.7	26,869.0	26,706	12.7	26,629.6	**26,558**	64.7	**26,618.1**	**26,558**	
36	X-n266-k58	10.0	**75,563.3**	**75,478**	21.4	75,759.3	75,517	39.7	75,710.7	**75,478**	
37	X-n270-k35	9.1	35,363.4	35,324	11.3	35,367.2	**35,303**	25.2	**35,314.6**	**35,303**	
38	X-n275-k28	3.6	21,256.0	**21,245**	12.0	21,280.6	**21,245**	30.1	**21,255.0**	**21,245**	
39	X-n280-k17	9.6	33,769.4	33,624	19.1	33,605.8	33,505	181.1	**33,587.9**	**33,503**	
40	X-n284-k15	8.6	20,448.5	20,295	19.9	20,286.4	**20,227**	130.0	**20,282.1**	20,228	
41	X-n289-k60	16.1	95,450.6	95,315	21.3	95,469.5	95,244	55.5	**95,447.2**	**95,211**	
42	X-n294-k50	12.4	**47,254.7**	47,190	14.7	47,259.0	47,171	35.9	47,272.7	**47,161**	⊛
43	X-n298-k31	6.9	34,356.0	34,239	10.9	34,292.1	**34,231**	27.5	**34,276.3**	**34,231**	
44	X-n303-k21	14.2	21,895.8	21,812	17.3	21,850.9	21,748	64.3	**21,811.2**	**21,744**	
45	X-n308-k13	9.5	26,101.1	25,901	15.3	**25,895.4**	**25,859**	150.0	25,897.3	25,861	
46	X-n313-k71	17.5	94,297.3	94,192	22.4	**94,265.2**	94,093	40.8	94,280.4	**94,045**	
47	X-n317-k53	8.6	**78,356.0**	**78,355**	22.4	78,387.8	**78,355**	66.9	78,385.3	**78,355**	
48	X-n322-k28	14.7	29,991.3	29,877	15.2	29,956.1	29,870	36.8	**29,892.5**	**29,834**	⊛
49	X-n327-k20	19.1	27,812.4	27,599	18.2	27,628.2	27,564	91.4	**27,590.8**	**27,532**	⊛
50	X-n331-k15	15.7	31,235.5	31,105	24.4	31,159.6	**31,103**	135.9	**31,126.7**	**31,103**	
	Average gap:		0.37%	0.13%		0.14%	0.02%		0.10%	0.00%	

† Scaled by a factor 1.33 to account for CPU differences. ⊛ improvement over the best solution listed at http://vrp.atd-lab.inf.puc-rio.br/

proximately 33% faster than the Intel i7-3960X 3.30 GHz used by Uchoa et al. (2017). Therefore, the time values reported for our method were multiplied by 1.33 in the tables.

Tables 2 and 3 report the results obtained with UHGS-BS, in comparison with UHGS and ILS. The columns present the average CPU time in minutes, the average solution value and the best solution value for all approaches. The best results are highlighted in boldface in the table, with ⊛ indicating an improvement over the solutions collected at http://vrp.atd-lab.inf.puc-rio.br/.

These tables highlight how the local search applied on $\mathcal{S}_2^{\mathrm{B}}$ resulted in several improvements over the state-of-the-art methods considered, with an average gap of 0.10% and 0.24% from the best solutions on the medium and large instances, respectively, in comparison with 0.14% and 0.30% for a classical search in $\mathcal{S}$. This improvement in solution quality comes at the price of an overall twofold increase of CPU time. Another notable observation of these experiments is that the search in $\mathcal{S}_2^{\mathrm{B}}$ finds solutions which are structurally different. Indeed, we obtained some new best known solutions for unexpectedly small instances, with 256, 294, 322 and 327 customers (marked with a ⊛). This is not a coincidence, given that the solutions listed at http://vrp.atd-lab.inf.puc-rio.br/ already originate from various previous articles and methods, over a large cumulated amount of test runs and parameter settings. For the particular case of the instance X-n256-k16, one interesting characteristic was observed: only 16 vehicles are used, with a total capacity usage of 99.6%.

5. Conclusions and future work

In this article, we investigated decision-set decompositions for the classic CVRP. Our experiments show that decomposing the problem into Assignment and Sequencing decisions, and conduct-

Table 3
Results for the instances with more than 331 customers from Uchoa et al. (2017).

#	Instance	ILS			UHGS			UHGS-BS			
		Time	Average	Best	Time	Average	Best	Time†	Average	Best	
51	X-n336-k84	21.4	139,461.0	**139,197**	38.0	139,534.9	139,210	87.7	**139,460.1**	139,303	
52	X-n344-k43	22.6	42,284.0	42,146	21.7	42,208.8	42,099	52.8	**42,156.1**	**42,056**	⊛
53	X-n351-k40	25.2	26,150.3	26,021	33.7	26,014.0	25,946	68.5	**25,981.8**	**25,938**	
54	X-n359-k29	48.9	52,076.5	51,706	34.9	51,721.7	**51,509**	148.9	**51,640.7**	51,555	
55	X-n367-k17	13.1	23,003.2	22,902	22.0	**22,838.4**	**22,814**	156.1	22,876.2	**22,814**	
56	X-n376-k94	7.1	**147,713.0**	**147,713**	28.3	147,750.2	147,717	93.5	147,740.5	147,714	
57	X-n384-k52	34.5	66,372.5	66,116	40.2	66,270.2	66,081	75.6	**66,170.3**	**65,997**	
58	X-n393-k38	20.8	38,457.4	38,298	28.7	38,374.9	38,269	65.6	**38,309.3**	**38,260**	⊛
59	X-n401-k29	60.4	66,715.1	66,453	49.5	66,365.4	66,243	146.6	**66,359.0**	**66,212**	
60	X-n411-k19	23.8	19,954.9	19,792	34.7	19,743.8	**19,718**	167.6	**19,736.7**	19,721	
61	X-n420-k130	22.2	**107,838.0**	**107,798**	53.2	107,924.1	**107,798**	116.6	107,913.7	**107,798**	
62	X-n429-k61	38.2	65,746.6	65,563	41.5	**65,648.5**	65,501	87.3	65,661.6	**65,470**	
63	X-n439-k37	39.6	36,441.6	**36,395**	34.6	36,451.1	**36,395**	76.0	**36,410.1**	**36,395**	
64	X-n449-k29	59.9	56,204.9	55,761	64.9	55,553.1	55,378	176.4	**55,432.7**	**55,330**	
65	X-n459-k26	60.6	24,462.4	24,209	42.8	24,272.6	24,181	123.5	**24,226.0**	**24,145**	⊛
66	X-n469-k138	36.3	**222,182.0**	**221,909**	86.7	222,617.1	222,070	189.3	222,427.5	222,235	
67	X-n480-k70	50.4	89,871.2	89,694	67.0	89,760.1	89,535	97.3	**89,744.7**	**89,513**	
68	X-n491-k59	52.2	67,226.7	66,965	71.9	66,898.0	66,633	108.9	**66,794.1**	**66,607**	
69	X-n502-k39	80.8	69,346.8	69,284	63.6	69,328.8	69,253	236.3	**69,277.1**	**69,247**	
70	X-n513-k21	35.0	24,434.0	24,332	33.1	24,296.6	**24,201**	132.2	**24,256.2**	**24,201**	
71	X-n524-k137	27.3	155,005.0	**154,709**	80.7	**154,979.5**	154,774	275.8	155,038.1	154,787	
72	X-n536-k96	62.1	95,700.7	95,524	107.5	**95,330.6**	95,122	192.2	95,335.4	**95,112**	
73	X-n548-k50	64.0	**86,874.1**	**86,710**	84.2	86,998.5	86,822	181.7	86,881.0	86,778	
74	X-n561-k42	68.9	43,131.3	42,952	60.6	42,866.4	42,756	102.6	**42,860.0**	**42,733**	
75	X-n573-k30	112.0	51,173.0	51,092	188.2	50,915.1	**50,780**	1040.5	**50,876.9**	50,801	
76	X-n586-k159	78.5	190,919.0	190,612	175.3	190,838.0	190,543	311.6	**190,752.4**	**190,442**	
77	X-n599-k92	73.0	109,384.0	109,056	125.9	109,064.2	108,813	222.0	**108,993.3**	**108,576**	
78	X-n613-k62	74.8	60,444.2	60,229	117.3	59,960.0	59,778	137.8	**59,859.7**	**59,654**	
79	X-n627-k43	162.7	62,905.6	62,783	239.7	62,524.1	62,366	722.3	**62,442.9**	**62,254**	
80	X-n641-k35	140.4	64,606.1	64,462	158.8	64,192.0	**63,839**	404.9	**64,105.6**	63,859	
81	X-n655-k131	47.2	**106,782.0**	**106,780**	150.5	106,899.1	106,829	336.8	106,855.6	106,804	
82	X-n670-k126	61.2	147,676.0	147,045	264.1	**147,222.7**	**146,705**	356.1	147,663.9	147,163	
83	X-n685-k75	73.9	68,988.2	68,646	156.7	68,654.1	**68,425**	235.4	**68,596.0**	68,496	
84	X-n701-k44	210.1	83,042.2	82,888	253.2	82,487.4	82,293	489.4	**82,409.2**	**82,174**	
85	X-n716-k35	225.8	44,171.6	44,021	264.3	43,641.4	43,525	581.5	**43,599.9**	**43,498**	
86	X-n733-k159	111.6	137,045.0	136,832	244.5	**136,587.6**	**136,366**	444.5	136,607.4	136,424	
87	X-n749-k98	127.2	78,275.9	77,952	313.9	77,864.9	77,715	410.1	**77,862.8**	**77,605**	
88	X-n766-k71	242.1	115,738.0	115,443	383.0	115,147.9	**114,683**	439.6	**115,115.9**	114,812	
89	X-n783-k48	235.5	73,722.9	73,447	269.7	73,009.6	72,781	467.1	**72,892.4**	**72,738**	
90	X-n801-k40	432.6	74,005.7	73,830	289.2	73,731.0	73,587	563.9	**73,651.6**	**73,466**	
91	X-n819-k171	148.9	159,425.0	159,164	374.3	158,899.3	158,611	898.5	**158,849.0**	**158,592**	
92	X-n837-k142	173.2	195,027.0	194,804	463.4	**194,476.5**	**194,266**	844.4	194,504.0	194,356	
93	X-n856-k95	153.7	89,277.6	89,060	288.4	89,238.7	89,118	418.4	**89,220.0**	**89,020**	
94	X-n876-k59	409.3	100,417.0	100,177	495.4	99,884.1	99,715	722.3	**99,780.3**	**99,610**	
95	X-n895-k37	410.2	54,958.5	54,713	321.9	54,439.8	**54,172**	665.2	**54,407.4**	54,254	
96	X-n916-k207	226.1	330,948.0	330,639	560.8	330,198.3	**329,836**	1439.7	**330,153.2**	329,866	
97	X-n936-k151	202.5	134,530.0	133,592	531.5	**133,512.9**	**133,140**	1359.6	133,729.3	133,376	
98	X-n957-k87	311.2	85,936.6	85,697	432.9	85,822.6	85,672	409.5	**85,681.5**	**85,555**	
99	X-n979-k58	687.2	120,253.0	119,994	554.0	**119,502.1**	119,194	1234.7	119,527.7	**119,188**	
100	X-n1001-k43	792.8	73,985.4	73,776	549.0	72,956.0	72,742	1267.3	**72,903.3**	**72,629**	
	Average gap:		0.74%	0.42%		0.30%	0.06%		0.24%	0.03%	

† Scaled by a factor 1.33 to account for CPU differences. ⊛ improvement over the best solution listed at http://vrp.atd-lab.inf.puc-rio.br/

ing local search in the Assignment space ($\mathcal{S}^{\mathrm{A}}$) generates consistently better results than heuristically searching in the complete search space $\mathcal{S}$. When doing so, each solution is systematically *decoded* by finding optimal routes (Sequencing decisions) with the Concorde TSP solver. However, the extra CPU dedicated to solution decoding is prohibitively high to employ this technique within state-of-the-art metaheuristics. To circumvent this issue, the B&S neighborhood was employed to define an intermediate search space ($\mathcal{S}_k^{\mathrm{B}}$) which can be more efficiently searched while retaining a high solution quality.

Different techniques were proposed and evaluated for efficiently searching in space $\mathcal{S}_k^{\mathrm{B}}$: *neighborhood reduction, dynamic move filters, concatenation techniques* and *efficient memory structures*. Moreover, tunneling techniques were employed to reshape $\mathcal{S}_k^{\mathrm{B}}$ into a search space more similar to $\mathcal{S}^{\mathrm{A}}$ as the search progresses. The combination of these techniques within the UHGS solver resulted in an significant improvement of solution accuracy. Multiple instances from the literature had their best known solution improved, and new state-of-the-art results were obtained for the CVRP.

This improvement of search space, however, still results in some extra computational effort. Therefore, many research perspectives are open about how to fully exploit a larger search space such as $\mathcal{S}_k^{\mathrm{B}}$ without any time compromise. One possibility could be to employ the search in $\mathcal{S}_k^{\mathrm{B}}$ only in exceptional circumstances, for a selected subset of promising solutions (e.g., each new best incumbent solutions during the search). Other open research possibilities relate to the search space $\mathcal{S}^{\mathrm{A}}$. Indeed, even with efficient memory structures and neighborhood reduction techniques, using Concorde for each solution evaluation remains impracticable. This effort could be mitigated if good and fast lower bounds were proposed for the cost of the routes, therefore permitting to filter a large proportion of moves as in Vidal (2017). Concorde is also not optimized to handle millions of small cardinality routes issued from a local search, such that a dedicated (heuristic or exact) TSP solution procedure exploiting information from the current incumbent solution represents another promising research line. Finally, the proposed approach can be naturally evaluated for other variants of the CVRP, in the presence of different types of attributes that have an impact on the Assignment and Sequencing decision classes. These are all promising perspectives for future research at the crossroads of dynamic programming, integer programming, and metaheuristics.

Acknowledgments

The authors thank the referees for their detailed comments, which significantly contributed to improving the quality of the paper. This research has been funded by the Belgian Science Policy Office (BELSPO) in the Interuniversity Attraction Pole COMEX (http://comex.ulb.ac.be), the National Council for Scientific and Technological Development (CNPQ – grant number 308498/2015-1) and FAPERJ in Brazil (grant number E-26/203.310/2016). The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Hercules Foundation and the Flemish Government – department EWI. This support is gratefully acknowledged. In addition, we would like to thank Luke Connolly for providing editorial consultation.

References

Ahuja, R., Ergun, O., Orlin, J., Punnen, A., 2002. A survey of very large-scale neighborhood search techniques. Discrete Appl. Math. 123 (1–3), 75–102.

Applegate, D., Bixby, R., Chvatal, V., Cook, W., 2006. CONCORDE TSP solver. http://www.math.uwaterloo.ca/tsp/concorde.html. Accessed: 2017-10-17.

Balas, E., Simonetti, N., 2001. Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study. INFORMS J. Comput. 13 (1), 56–75.

Balinski, M., Quandt, R., 1964. On an integer program for a delivery problem. Oper. Res. 12 (2), 300–304.

Beasley, J., 1983. Route first-cluster second methods for vehicle routing. Omega (Westport) 11 (4), 403–408.

Bodin, L., Berman, L., 1979. Routing and scheduling of school buses by computer. Transp. Sci. 13 (2), 113.

Bompadre, A., 2012. Exponential lower bounds on the complexity of a class of dynamic programs for combinatorial optimization problems. Algorithmica 62, 659–700.

Bulhões, T., Hà, M., Martinelli, R., Vidal, T., 2018. The vehicle routing problem with service level constraints. Eur. J. Oper. Res. 265 (2), 544–558.

Clarke, G., Wright, J.W., 1964. Scheduling of vehicles from a central depot to a number of delivery points. Oper. Res. 12 (4), 568–581.

Crainic, T., Toulouse, M., 2008. Explicit and emergent cooperation schemes for search algorithms. In: Maniezzo, V., Battiti, R., Watson, J.P. (Eds.), Learning and Intelligent Optimization, Volume 5315 of LNCS. Springer-Verlag, Berlin, Heidelberg, pp. 95–109.

Deineko, V., Woeginger, G., 2000. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. Math. Program 87 (3), 519–542.

Fisher, M., Jaikumar, R., 1981. A generalized assignment heuristic for vehicle routing. Networks 11 (2), 109–124.

Foster, B., Ryan, D., 1976. An integer programming approach to the vehicle scheduling problem. Oper. Res. Q. 27 (2), 367–384.

Gendreau, M., Potvin, J.Y., 2010. Handbook of Metaheuristics, vol. 146. Springer.

Geoffrion, A., 1970. Elements of large-scale mathematical programming: part i: concepts. Manag. Sci. 16 (11), 652–675.

Goel, A., Vidal, T., 2014. Hours of service regulations in road freight transport: an optimization-based international assessment. Transp. Sci. 48 (3), 391–412.

Gschwind, T., Drexl, M., 2016. Adaptive large neighborhood search with a constant-time feasibility test for the dial-a-ride problem. In: Working Papers 1624. Gutenberg School of Management and Economics, Johannes Gutenberg-Universität Mainz.

Gutin, G., Yeo, A., 2003. Upper bounds on ATSP neighborhood size. Discrete Appl. Math. 129 (2–3), 533–538.

Hintsch, T., Irnich, S., 2018. Large multiple neighborhood search for the clustered vehicle-routing problem. Eur. J. Oper. Res. 270 (1), 118–131.

Irnich, S., 2008. Solution of real-world postman problems. Eur. J. Oper. Res. 190 (1), 52–67.

Irnich, S., 2013. Efficient local search for the CARP with combined exponential and Classical Neighborhood. VEROLOG. Southampton, U.K..

Johnson, D., McGeoch, L., 1997. The traveling salesman problem: a case study in local optimization. In: Aarts, E., Lenstra, J. (Eds.), Local search in Combinatorial Optimization. University Press, Princeton, NJ, pp. 215–310.

Kernighan, B., Ritchie, D., 1988. The C Programming Language, Volume 78, 2 ed. Inc., Prentice-Hall.

Knuth, D., 1973. The Art of Computer Programming, Vol. 3: Sorting and Searching. Addison-Wesley.

Laporte, G., Ropke, S., Vidal, T., 2014. Heuristics for the Vehicle Routing Problem. In: Toth, P., Vigo, D. (Eds.), Vehicle Routing: Problems, Methods, and Applications, chapter 4, pp. 87–116. Society for Industrial and Applied Mathematics

Muter, I., Birbil, S., Sahin, G., 2010. Combination of metaheuristic and exact algorithms for solving set covering-type optimization problems. INFORMS J. Comput. 22 (4), 603–619.

Pollaris, H., Braekers, K., Caris, A., Janssens, G.K., Limbourg, S., 2015. Vehicle routing problems with loading constraints: state-of-the-art and future directions. OR Spectrum 37 (2), 297–330.

Renaud, J., Boctor, F., Laporte, G., 1996. An improved petal heuristic for the vehicle routeing problem. J. Operat. Res. Soc. 47 (2), 329–336.

Software, P., 2018. CPU benchmarks, https://www.cpubenchmark.net/.

Subramanian, A., Uchoa, E., Ochi, L.S., 2013. A hybrid algorithm for a class of vehicle routing problems. Comput. Operat. Res. 40 (10), 2519–2531.

Toth, P., Vigo, D., 2003. The granular tabu search and its application to the vehicle-routing problem. INFORMS J. Comput. 15 (4), 333–346.

Toth, P., Vigo, D., 2014. Vehicle routing: Problems, methods, and applications, 2nd Soc. Ind. Appl. Math., Philadelphia, PA.

Uchoa, E., Pecin, D., Pessoa, A., Poggi, M., Subramanian, A., Vidal, T., 2017. New benchmark instances for the capacitated vehicle routing problem. Eur. J. Oper. Res. 257 (3), 845–858.

Vidal, T., 2017. Node, edge, arc routing and turn penalties: multiple problems – one neighborhood extension. Oper. Res. 65 (4), 992–1010.

Vidal, T., Battarra, M., Subramanian, A., Erdogan, G., 2015a. Hybrid metaheuristics for the clustered vehicle routing problem. Comput. Operat. Res. 58 (1), 87–99.

Vidal, T., Crainic, T., Gendreau, M., Lahrichi, N., Rei, W., 2012. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Oper. Res. 60 (3), 611–624.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2013a. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Comput. Operat. Res. 40 (1), 475–489.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2013b. Heuristics for multi-attribute vehicle routing problems: a survey and synthesis. Eur. J. Oper. Res. 231 (1), 1–21.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2014. A unified solution framework for multi-attribute vehicle routing problems. Eur. J. Oper. Res. 234 (3), 658–673.

Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2015b. Timing problems and algorithms: time decisions for sequences of activities. Networks 65 (2), 102–128.

Vidal, T., Maculan, N., Ochi, L., Penna, P., 2016. Large neighborhoods with implicit customer selection for vehicle routing problems with profits. Transp. Sci. 50 (2), 720–734.