Appendix A: Algorithmic Components Analysis: Additional Material

This section contains additional material related to the Algorithmic Components Analysis Section.

A.1. Initial Solution Definition

Consider the scenario depicted in Figure 1, showing a single run for instance X-n936-k136. This example illustrates the evolution of the route minimization procedure with $\Delta_{RM} = 1000$ on the left and of the core optimization procedure with $\Delta_{CO} = 5000$ on the right. Both procedures ran for about three seconds and were applied to the same starting solution generated by the construction phase. By moving into the infeasible space, the route minimization procedure is very effective in quickly improving and compacting trivially bad initial solutions. However, since its structure is specifically designed to reduce the number of routes, the improvements vanish after a few hundred iterations.

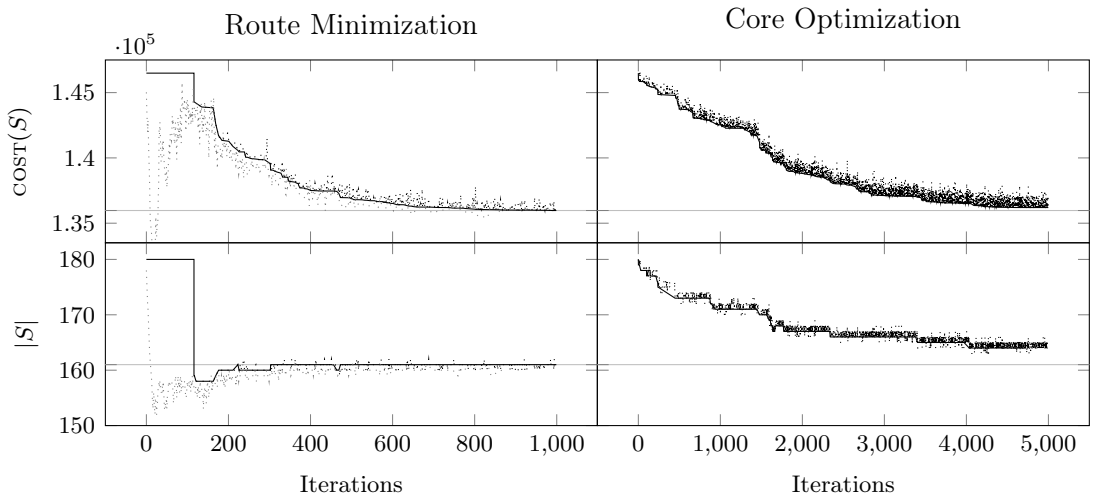


Figure 1 **Example of improvement procedures evolution when applied to the same initial solution for instance X-n936-k151. The continuous line shows the cost (top) and number of routes (bottom) associated with the best solution (in terms of cost) found up to that iteration; dots represents the current search trajectory. Gray and black dots are associated with infeasible and feasible solutions, respectively.**

A.2. Local Search

Figure 2 shows, for each local search operator applied to a shaken solution, the gap improvement when successfully applied, the application time in 10^{-6} seconds, and the success ratio computed as the number of improving applications over the total number of attempts. Similarly, Figure 3 shows the effect of EJCH$(\cdot)$ when applied to solutions that are already a local optimum for the first HRVND tier.

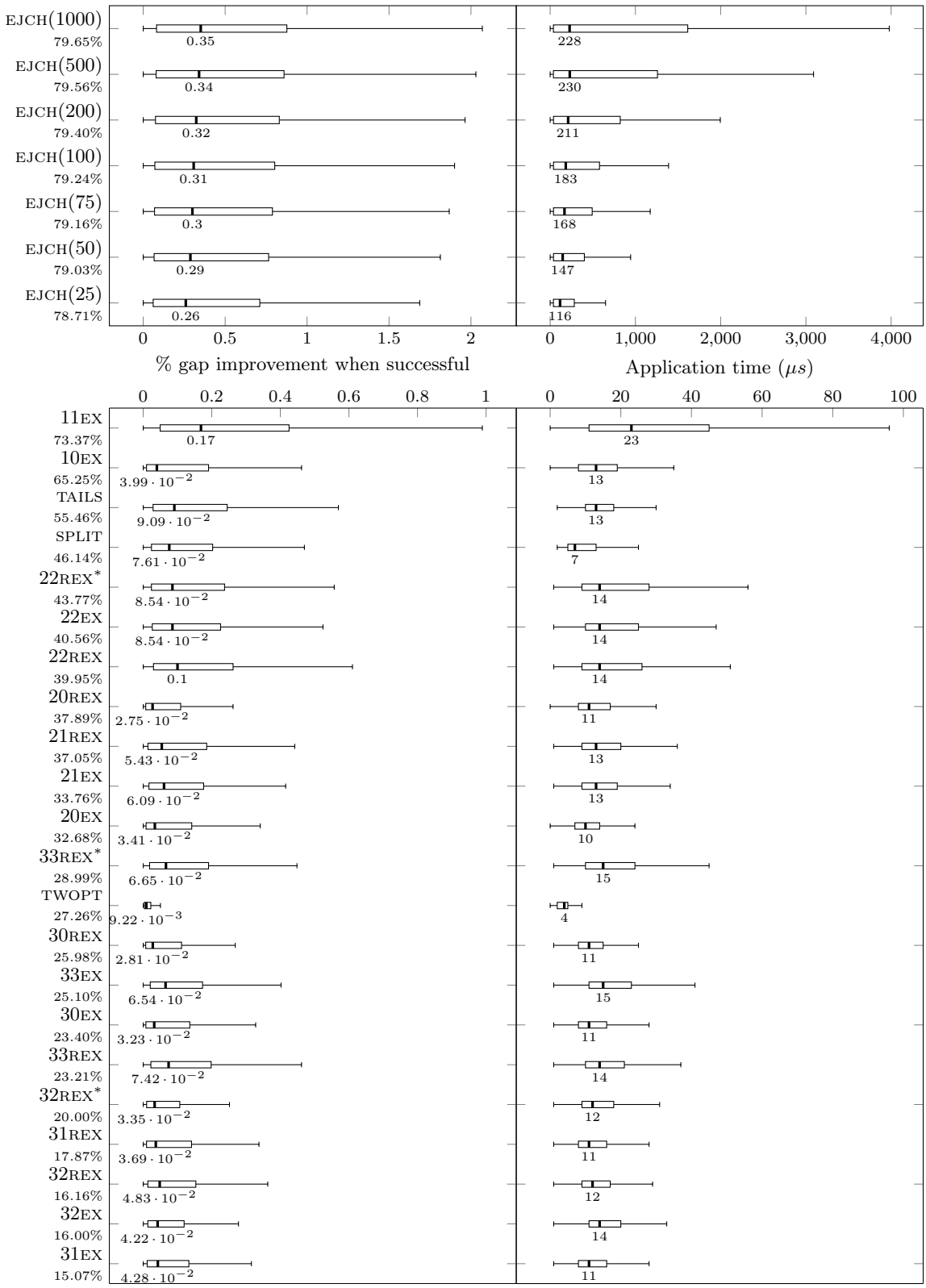


Figure 2 **Statistics for local search operators when applied to shaken solutions. For each operator, we report the expected gap improvement when successfully applied (left), the total exploration time (right), and the success ratio (below each operator's name). The median value is shown below each boxplot.**

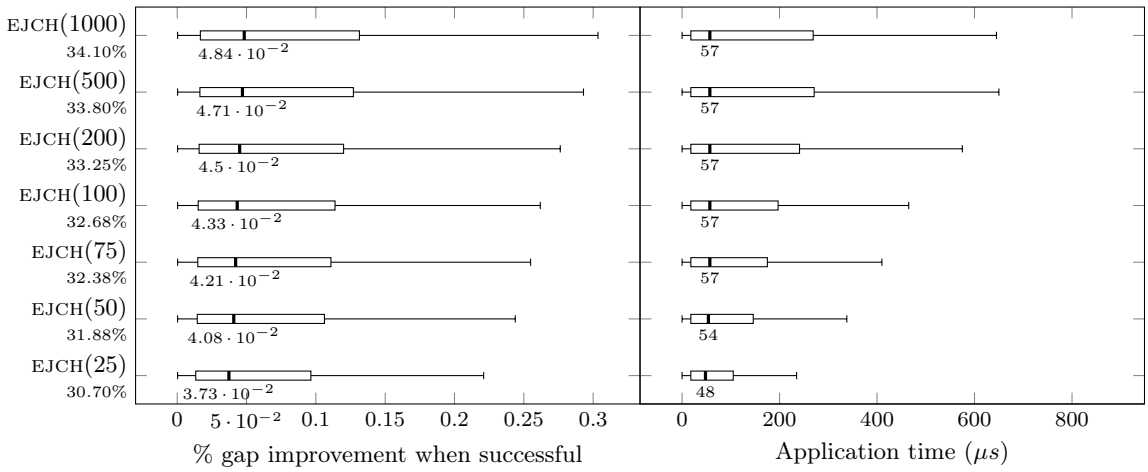


Figure 3 **Statistics for local search operators when applied to HRVND first tier local optima. For each operator, we report the expected gap improvement when successfully applied (left), the total exploration time (right), and the success ratio (below each operator's name). The median value is shown below each boxplot.**

A.3. Extreme Runs

Table 1 shows the results we obtained by setting $\Delta_{CO} = 10^7$.

A.4. Computations with Limited Memory Footprint

Random access memory (RAM) is relatively cheap nowadays, and large amounts are easily supported, even by low-cost laptops. Time, on the other hand, is much more valuable; new chips are moving to massive parallelization rather than an increase in their working frequency. Solution methods, however, seldom make use of parallel processing to solve a single instance, even though this approach might be a very interesting, yet challenging, research direction for very large-scale instances. We can thus state that the real bottleneck is probably not the amount of available RAM, but the computing time used to solve an instance. Indeed, the largest instance we considered, F2 from the $\mathbb{B}$ dataset, with thirty thousand vertices, can be easily processed on a laptop with 16GB of RAM. During the main core optimization procedure, only about 66% of the available RAM would be used. Despite the fact that RAM is not currently a limiting factor, in this section we investigate the behavior of FILO when the cost matrix, which is one of the most RAM-consuming data structures we use, is not stored but, instead, arc costs are computed on demand. In this case, the computing time increases by about 52% (i.e., from 1.72 minutes to 2.60 minutes). As an example, without storing the cost matrix, the F2 instance's RAM requirements drop from 10.56 GB to 3.68GB. This may allow the algorithm to be applicable to instances even larger than the one we considered. An alternative, more sophisticated, method proposed in Arnold, Gendreau, and Sörensen (2019), consists of storing a number of arcs connecting close vertices that are supposed to be used more frequently than others, in a hashmap.

Table 1 Long computations on large-sized $\mathbb{X}$ instances.

ID	BKS	Best	Avg	Worst	t
X-n502-k39	69226	0.00	0.01	0.04	262.65
X-n513-k21	24201	0.00	0.07	0.16	222.27
X-n524-k153	154593	0.01	0.14	0.27	146.99
X-n536-k96	94868	0.50	0.60	0.71	161.76
X-n548-k50	86700	0.01	0.02	0.09	207.85
X-n561-k42	42717	0.08	0.20	0.28	138.73
X-n573-k30	50673	0.12	0.22	0.26	205.74
X-n586-k159	190316	0.20	0.25	0.31	181.93
X-n599-k92	108451	0.15	0.22	0.33	189.87
X-n613-k62	59545	0.12	0.20	0.39	121.34
X-n627-k43	62173	0.03	0.12	0.32	198.31
X-n641-k35	63705	0.05	0.11	0.18	206.51
X-n655-k131	106780	0.00	0.02	0.03	378.73
X-n670-k130	146332	0.50	0.64	0.90	136.33
X-n685-k75	68225	0.17	0.34	0.52	142.94
X-n701-k44	81923	0.03	0.11	0.35	163.87
X-n716-k35	43387	0.09	0.16	0.29	176.98
X-n733-k159	136190	0.06	0.13	0.20	135.04
X-n749-k98	77314	0.15	0.25	0.38	141.90
X-n766-k71	114456	0.16	0.27	0.34	156.24
X-n783-k48	72394	0.10	0.17	0.26	191.01
X-n801-k40	73331	-0.03	0.09	0.15	188.16
X-n819-k171	158121	0.39	0.44	0.53	141.35
X-n837-k142	193737	0.12	0.21	0.26	191.22
X-n856-k95	88990	0.01	0.07	0.13	188.52
X-n876-k59	99303	0.11	0.16	0.24	176.58
X-n895-k37	53928	-0.04	0.13	0.31	189.72
X-n916-k207	329179	0.19	0.23	0.31	200.72
X-n936-k151	132812	0.16	0.26	0.38	127.16
X-n957-k87	85469	-0.00	0.06	0.11	195.38
X-n979-k58	118988	0.05	0.16	0.24	244.49
X-n1001-k43	72369	0.06	0.17	0.27	169.54
Mean		0.11	0.19	0.30	183.74

New best solutions: (X-n801-k40, 73311);(X-n895-k37, 53906);(X-n957-k87, 85467).

Appendix B: Move Generators and Static Move Descriptors

Efficiently managing move generators is crucial for any local search-based algorithm making use of GNs. In addition, flexibility might be required when experimenting different sparsification rules and composition of them, as we did in the analysis proposed in Section 4.4. In the following sections, we first discuss a flexible but still efficient implementation of move generators for the symmetric CVRP, supporting the union of different sparsification rules and a vertex-wise dynamic management. Then, we show how it can be used to efficiently implement SMD-based local search operators.

B.1. Move Generators: Storage and Management

Given a set R of sparsification rules, the complete set of move generators T is defined by the union of a number of move generator sets T_r, each one defined by a sparsification rule $r \in R$; that is, $T = \bigcup_{r \in R} T_r$. Each set T_r may be filtered according to a sparsification vector $\boldsymbol{\gamma} = (\gamma_0, \gamma_1, \ldots, \gamma_N)$ defining, for each vertex $i \in V$, a percentage $\gamma_i \in [0,1]$ of move generators in T_r to be considered as active. More precisely, the dynamic set of move generators $T_r^{\boldsymbol{\gamma}}$ filtered according to $\boldsymbol{\gamma}$ is defined as $T_r^{\boldsymbol{\gamma}} = \bigcup_{i \in V} \{(i,j),(j,i) : j \in V \wedge \textsc{Condition}(i,j,\gamma_i)\}$, where $\textsc{Condition}(i,j,\gamma_i)$ is a criterion that determines whether (i,j) and (j,i) are active based on the value of γ_i.

In the following, we describe a possible implementation of the above defined general framework for move generators. An illustrative example, representing the implementation of a set of move generators T defined by the union of two sparsification rules r_0 and r_1, is shown in Figure 4.

A list of move generators $L(T)$ is built by considering unique move generators defined by the different sparsification rules $r \in R$. By denoting with $L(T)_\ell$ the move generator $(i,j)_\ell$ indexed by ℓ in $L(T)$, we structured the list $L(T)$ so as to satisfy:

1. $L(T)_\ell = (i,j)_\ell \wedge L(T)_{\ell+1} = (j,i)_{\ell+1}$ for each even index ℓ;
2. $L(T) \not\ni (i,i), \forall i \in V$.

Condition 1 asks that both (i,j) and (j,i) are considered. This ensures the evaluation of a consistent set of moves when exploring asymmetric neighborhoods. Moreover, (i,j) and (j,i) are stored contiguously into $L(T)$ so that given a move generator indexed by ℓ, its reversed counterpart can be efficiently retrieved, when necessary. Finally, Condition 2 discards self-moves which are typically not used in local search procedures.

A number of lists $L(T_r, v)$, one for each vertex $v \in V$, is associated with each sparsification rule $r \in R$. Those lists identify a portion of $L(T)$ consisting of move generators $(i,j) \in T_r$. In particular, each list $L(T_r, v)$ keeps track of the even indices ℓ of move generators $(i,j)_\ell$ such that $v = i$ or $v = j$. Because of Condition 1 on $L(T)$, it is not necessary to store the index of the counterpart of (i,j) that can be found accessing $L(T)_{\ell+1}$.

In addition, each list $L(T_r, i)$, along with indices ℓ, stores an inclusion percentage value $p_{ri\ell}$ used to define whether move generators $(i,j)_\ell$ and $(j,i)_{\ell+1}$ are active according to the current sparsification factor γ_i. More precisely, the above defined $\textsc{Condition}(i,j,\gamma_i)$ is implemented as $\textsc{Condition}(i,j,\gamma_i) = p_{ri\ell} \leq \gamma_i$ where ℓ is the even index pointing to move generator $L(T)_\ell$ having i and j as endpoints.

Depending on how the inclusion percentage values are defined we can model the classical or the vertex-wise management of the dynamic move generators for a sparsification rule $r \in R$. In the classical management $\sum_{i \in V} \sum_{\ell \in L(T_r,i)} p_{ri\ell} = 1$ whereas in the vertex-wise management $\sum_{\ell \in L(T_r,i)} p_{ri\ell} = 1$ for each $i \in V$.

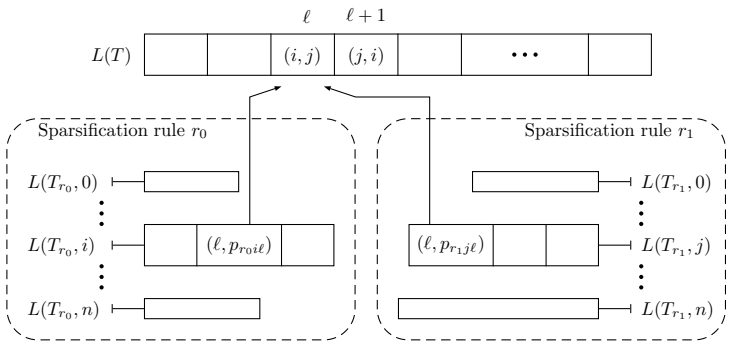


Figure 4 **Implementation of the list $L(T)$ of move generators defined by two sparsification rules r_0 and r_1. Two move generators (i,j) and (j,i) indexed ℓ and $\ell+1$ respectively, are explicitly shown. As can be seen, those move generators are defined by both sparsification rules and the associated entry in the lists point to the same shared entry in $L(T)$. The inclusion percentages $p_{r_0i\ell}$ and $p_{r_1j\ell}$ might however cause those move generators to be active at different times according to the value of γ_i and γ_j.**

The complete dynamic set of move generators can thus be addressed by iterating over each sparsification rule $r \in R$, each vertex $i \in V$, each list $L(T_r, i)$, and considering move generators $(i,j)_\ell$ and $(j,i)_{\ell+1}$ such that $p_{ri\ell} \leq \gamma_i$.

Note that by storing indices instead of move generators, different sparsification rules identifying the same subset of move generators would point to the same entry of $L(T)$.

In our implementation, the Sparsification Rule r_0 described in Section 2.2.2 is implemented by defining $p_{r_0v\ell} = n/|L(T_{r_0}, v)|$, where n is the index of move generator ℓ in a list of move generators $(i,j) \in L(T_{r_0}, i)$ sorted in increasing c_{ij} cost. The additional Sparsification Rule r_1 employed in the analysis of Section 4.4 defines instead $p_{r_1v\ell} = n/|T_{r_1}|$, where n is the index of move generator ℓ in a list of move generators $(i,j) \in L(T_{r_1})$ sorted in increasing c_{ij} cost.

B.2. An Abstract SMD-based Local Search Operator

The general structure of all SMD-based local search operators we used is shown in Algorithm 1. Note that, as mentioned in Section 2.2.3, we use the terms SMD and move generator interchangeably.

First, during a *pre-processing* step, additional computation useful for the actual neighborhood exploration may be executed. As an example, TAILS and SPLIT benefit from pre-computing route cumulative loads.

A heap data structure $\mathcal{H}$ is initialized with the currently active move generators according to the sparsification vector $\boldsymbol{\gamma}$, and such that at least one of the endpoints belongs to the set $\bar{V}_S$ of cached vertices for the solution S under examination. In particular, those move generators, denoted by $T^\gamma(S)$, can be easily retrieved by using the data structures defined in Section B.1 and, more specifically, $T^\gamma(S) = \bigcup_{r \in R, i \in \bar{V}_S} \{(i,j)_\ell : p_{ri\ell} \leq \gamma_i, \ell \in L(T_r, i)\}$. When dealing with local search operators defining asymmetric neighborhoods, both $(i,j)_\ell$ and $(j,i)_{\ell+1}$ have to be included. Given condition 1 on list $L(T)$ defined in Section B.1, this can be done by setting $T^\gamma(S) = T^\gamma(S) \cup \bigcup_{r \in R, i \in \bar{V}_S} \{(j,i)_{\ell+1} : p_{ri\ell} \leq \gamma_i, \ell \in L(T_r, i)\}$.

For each move generator $(i,j) \in T^\gamma(S)$, the δ-tag $\delta(i,j)$, identifying the effect of the application of the move induced by (i,j) on S, is computed. Every (i,j) inducing an improving move (i.e., $\delta(i,j) < 0$) is inserted into

the heap data structure $\mathcal{H}$. By only inserting improving move generators, the heap computational complexity is kept at its minimum.

The heap $\mathcal{H}$ is linearly scanned until a feasible move is found. If such a move cannot be found, then S is considered to be a local optimum with respect to the local search operator under examination. Note that by heuristically restricting the initialization stage to only consider move generators involving vertices in $\bar{V}_S$ some improvement may be overlooked, but, as shown by the experiments in Section 4.5, this does not significantly affect the final solution quality.

Once a move generator (i, j) inducing a feasible and improving change to S is found, a list A of operator-dependant affected vertices is assembled. The application of (i, j) during the execute stage will change some of the δ-tag of move generators involving vertices in A.

The update stage recomputes the δ-tag for active move generators $U_{(i,j)} = \bigcup_{r \in R, i \in A} \{(v_1, v_2)_\ell : \ell \in L(T_r, i) \wedge (v_1 = i \vee v_2 = i) \wedge p_{ri\ell} \leq \gamma_i\}$. In case of an asymmetric neighborhood, the updates are extended to $U_{(i,j)} = U_{(i,j)} \cup \bigcup_{r \in R, i \in A} \{(v_2, v_1)_{\ell+1} : \ell \in L(T_r, i) \wedge (v_1 = i \vee v_2 = i) \wedge p_{ri\ell} \leq \gamma_i\}$. For each move generator requiring an update $(v_1, v_2) \in U_{(i,j)}$, the following cases are possible:

- (v_1, v_2) is removed from the heap $\mathcal{H}$ if $(v_1, v_2) \in \mathcal{H}$ and $\delta(v_1, v_2) \geq 0$;
- (v_1, v_2) is inserted into the heap $\mathcal{H}$ if $(v_1, v_2) \notin \mathcal{H}$ and $\delta(v_1, v_2) < 0$;
- the heap property is checked and possibly restored if $(v_1, v_2) \in \mathcal{H}$ and $\delta(v_1, v_2) < 0$;
- finally, (v_1, v_2) is ignored if $(v_1, v_2) \notin \mathcal{H}$ and $\delta(v_1, v_2) \geq 0$.

Since different sparsification rules may refer to the same move generators, a timestamp associated with each move generator (i, j) can be used to avoid evaluating it more than once, both in the initialization and in the update stages. Note that also when using a single sparsification rule r, a double evaluation may occur when $i, j \in A$ and $T_\ell = (i, j)_\ell$ is such that $\ell \in L(T_r, i) \wedge p_{ri\ell} < \gamma_i$ and $\ell \in L(T_r, j) \wedge p_{rj\ell} < \gamma_j$.

Algorithm 1 Abstract SMD-Based Local Search Operator

```
1: procedure APPLY(S, T^γ)
2:     PREPROCESS(S)
3:     H ← INITIALIZATION(S, T^γ)
4:     n ← 0
5:     while n < len(H) do
6:         (i, j) ← PEEK(H, n)
7:         n ← n + 1
8:         if ¬ISFEASIBLE(S, (i, j)) then continue
9:         A ← AFFECTEDVERTICES(S, (i, j))
10:        S ← EXECUTE(S, (i, j))
11:        UPDATE(H, T^γ, A)
12:        n ← 0
13:    end while
14: end procedure
```

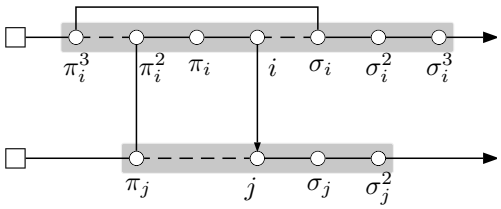


Figure 5 **A 30ex application induced by move generator (i, j) relocating path $(\pi_i^2 - i)$ between π_j and j. Some SMDs involving vertices in the gray area require an update to their δ-tag after the move execution.**

B.2.0.1. *Restricted Update for Asymmetric Neighborhoods.* The δ-tag update of move generators involving a vertex $i \in A$ for a local search operator defining an asymmetric neighborhood may sometimes be restricted from $\{(i, v_2), (v_1, i) : v_1, v_2 \in V\} \cap T^\gamma$ to only one between $\{(i, v_2) : v_2 \in V\} \cap T^\gamma$ and $\{(v_1, i) : v_1 \in V\} \cap T^\gamma$. As an example, consider the 30EX application shown in Figure 5. The set of affected vertices is $A = \{\pi_i^3, \pi_i^2, \pi_i, i, \sigma_i, \sigma_i^2, \sigma_i^3, \pi_j, j, \sigma_j, \sigma_j^2\}$. However, not all move generators $\{(i, j), (j, i) : i \in A, j \in V\} \cap T^\gamma$ have to be updated. For example, considering vertex π_i^3, move generators $(\pi_i^3, j), j \in V$ require an update since the successor of π_i^3 changes after the move execution, however, move generators $(j, \pi_i^3), j \in V$ do not, in fact, the predecessor of π_i^3 remains the same. For operator 30EX (and ignoring the current sparsification level), the total number of move generators requiring an update after a 30EX application can be reduced of approximately 36%.

Finally, we refer to Section B.3 of the Appendix for the operator-dependant definitions of the feasibility check, the assembly of the list A of vertices, the restricted update and the execution stage.

B.2.1. Dynamic Vertex-wise Move Generators for SMD-based Local Search Operators. Vertex-wise management of move generators requires a little extra care for the SMD update stage to be correctly performed. To highlight this, consider a scenario in which a vertex j is currently marked as cached at the beginning of a neighborhood exploration for a solution S; that is, $j \in \bar{V}_S$. An illustrative example is shown in Figure 6. The figure represents a portion of an instance with six customers together with a number of move generators depicted as lines ending with little circles. In particular, for a move generator (v_1, v_2), a full circle near to v_1 means that the move generator is currently active in v_1, i.e., $p_{rv_1\ell} \leq \gamma_{v_1}$ where ℓ is the index of $(v_1, v_2)_\ell$ in $L(T)$, while an empty circle represents the opposite scenario, i.e., $p_{rv_1\ell} > \gamma_{v_1}$. In the

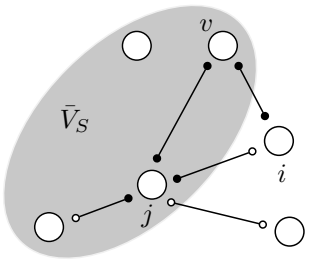


Figure 6 **A portion of an instance containing six customers (circles) and five move generators (lines). A move generator $(v_1, v_2)_\ell$ active in v_1, i.e., $p_{rv_1\ell} \leq \gamma_{v_1}$, is represented with a full circle near to v_1, whereas an empty circle denotes the opposite, i.e., $p_{rv_1\ell} > \gamma_{v_1}$. As an example (i, j) is active in j but not in i. The direction of move generators is not shown because not relevant.**

example, (i, j) is active in j but not in i. Finally, the grayed area identifies the set $\bar{V}_S$ of currently cached vertices for S.

During the SMD initialization stage, move generators involving vertices in $\bar{V}_S$ are considered to be inserted into the heap data structure $\mathcal{H}$. Suppose that, because inducing an improving move and currently active in j, move generator (i, j) along with with other improving move generators including (i, v) are inserted into $\mathcal{H}$. During the SMD search stage, move generator (i, v) is found feasible before the evaluation of (i, j). The move induced by (i, v) is then applied and a number of affected vertices A may be such that $i \in A$ but $j \notin A$. Note that i shares (i, j) with j but (i, j) is not currently active in i because of the sparsification factor γ_i. The SMD update stage requires that, from each vertex $v \in A$ active move generators are updated. *Should* (i, j) *still be updated even if not active in* i*?* The answer is yes. Being (i, j) active in j, it may be, as described in this scenario, already into the heap $\mathcal{H}$ and possibly be extracted during future search stages. In fact, being i among the affected vertices A as a result of the application of (i, v), the δ-tag of any move generator involving a vertex $v \in A$ requires an update, and hence (i, j) does. On the other hand, if (i, j) were not active in both j and i, updating it after (i, v) was not required because it could have never been inserted into $\mathcal{H}$.

The dynamic management of vertex-wise move generators may thus require the update of move generators that are not active in one of the affected vertices but only active in the other endpoint that may not be in the list of vertices affected by the move application. A possible implementation uses two additional flags per move generator (i, j) storing whether it is active in i and/or in j. During the SMD update stage the flags are checked to identify whether an update is required.

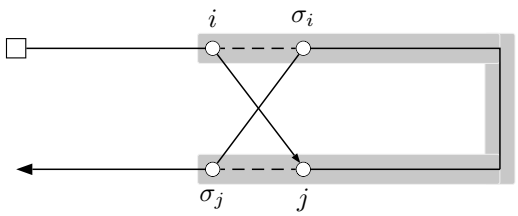
Finally, note that this approach works correctly under the assumption that the sparsification vector $\boldsymbol{\gamma}$ is not changed during a neighborhood exploration.

B.3. SMD Implementation Details of Local Search Operators

In the following, we provide a detailed description of the implementation for the local search operators used in FILO. In particular, we detail the operator-dependant procedures to be defined when applying a move induced by a move generator (i, j) within an SMD-based operator. To better accomplish this, we introduce few additional notation elements. In particular, we denote by π_i^ℓ and σ_i^ℓ the ℓ^{th}- predecessor and successor of vertex $i \in V$, respectively, in the solution under examination. We omit the apex when $\ell = 1$. Moreover, we identify with q_i^{up} and q_i^{from} the cumulative load up to and from any customer $i \in V_c$ included, i.e., $q_i^{up} = q_i + q_{\pi_i} + q_{\pi_i^2} + \ldots + q_0$ and $q_i^{from} = q_i + q_{\sigma_i} + q_{\sigma_i^2} + \ldots + q_0$. In the following paragraphs, a figure is shown for each local search operator highlighting the move induced by a move generator (i, j) and the set of vertices affected by its application (grayed area). Note that the move generator direction does not necessarily reflect the crossing direction in the resulting route. Finally, as defined in Section 2.2, path is used to refer to a contiguous sequence of vertices belonging to a route, and head and tail are used to denote a path belonging respectively to the initial and final part of a route.

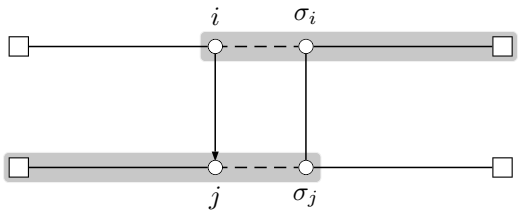
B.3.1. TWOPT.

Replace a path of vertices with its reverse.



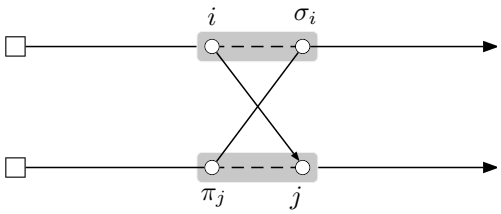
- Type: symmetric.
- Pre-processing: none.
- Cost computation: $\delta_{ij} = -c_{i\sigma_i} + c_{ij} - c_{j\sigma_j} + c_{\sigma_i\sigma_j}$.
- Feasibility Check: $r_i = r_j$.
- Update List: all vertices between i and σ_j, for which successors and predecessors change after the move application.
- Execution: reverse the path between σ_i and j included.

B.3.2. SPLIT. Replace the tail of route r_i with the reversed head of route r_j and replace the head of route r_j with the reversed tail of route r_i.



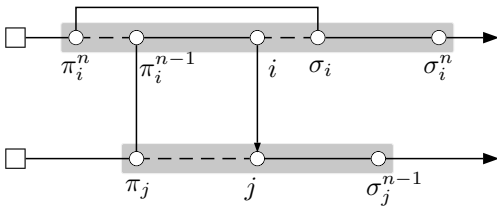
- Type: symmetric.
- Pre-processing: compute q_i^{up} and q_i^{from} for any customer $i \in V_c$.
- Cost Computation: $\delta_{ij} = -c_{i\sigma_i} + c_{ij} - c_{j\sigma_j} + c_{\sigma_i\sigma_j}$.
- Feasibility Check: $(r_i \neq r_j) \wedge (q_i^{up} + q_j^{up} \leq Q) \wedge (q_{\sigma_j}^{from} + q_{\sigma_i}^{from} \leq Q)$.
- Update List: all vertices from i to the depot and from the depot to σ_j, for which successors and predecessors change after the move application.
- Execution: replace (i, σ_i) and (j, σ_j) with (i, j) and (σ_i, σ_j) and reverse paths from depot to j and from σ_i to depot. Update the cumulative load q_v^{up} and q_v^{from} for customers v belonging to r_i and r_j, if not empty.

B.3.3. TAILS. Swap the tails of two different routes.



- Type: asymmetric.
- Pre-processing: compute q_i^{up} and q_i^{from} for any customer $i \in V_c$.
- Cost Computation: $\delta_{ij} = -c_{i\sigma_i} + c_{ij} - c_{j\pi_j} + c_{\pi_j\sigma_i}$.
- Feasibility Check: $(r_i \neq r_j) \wedge (q_i^{up} + q_j^{from} \leq Q) \wedge (q_{\pi_j}^{up} + q_{\sigma_i}^{from} \leq Q)$.
- Update List: i, σ_i, j and π_j.
  The update can be restricted to move generators $\{(i,v),(v,\sigma_i),(v,j),(\pi_j,v) : v \in V\} \cap T^{\gamma}$.
- Execution: replace (i,σ_i) and (π_j,j) with (i,j) and (π_j,σ_i). Update the cumulative load q_v^{up} and q_v^{from} for customers v belonging to r_i and r_j, if not empty.

B.3.4. *n*0EX with $n \geq 1$. Relocate a path of n vertices within the same or into a different route.

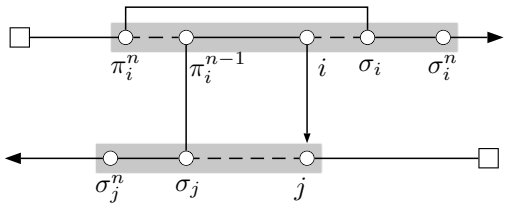


- Type: asymmetric.
- Pre-processing: none.
- Cost Computation: $\delta_{ij} = -c_{\pi_i^n \pi_i^{n-1}} - c_{i\sigma_i} - c_{j\pi_j} + c_{\pi_i^n \sigma_i} + c_{\pi_j \pi_i^{n-1}} + c_{ij}$.
- Feasibility Check: logical disjunction of
  — $(r_i = r_j) \wedge \bigwedge_{\ell=1}^{n-1} (j \neq \pi_i^{\ell}) \wedge (j \neq \sigma_i)$.
  When $r_i = r_j$ and $\bigvee_{\ell=1}^{n-1} j = \pi_i^{\ell}$, the path to relocate overlaps with the destination position.
  When $r_i = r_j$ and $j = \sigma_i$, the move does nothing but the δ_{ij} computation is not correct.
  — $(r_i \neq r_j) \wedge (i \neq 0) \wedge \bigwedge_{\ell=1}^{n-1} (\pi_i^{\ell} \neq 0) \wedge (q_{r_j} + q_i + \sum_{\ell=1}^{n-1} q_{\pi_i^{\ell}} \leq Q)$.
  The condition makes sure the depot is not relocated and the capacity constraint of the target route is respected.
- Update List: $\bigcup_{\ell=1}^{n} \pi_i^{\ell} \cup i \cup \bigcup_{\ell=1}^{n} \sigma_i^{\ell} \cup \pi_j \cup j \cup \bigcup_{\ell=1}^{n-1} \sigma_j^{\ell}$.
  The update can be restricted to move generators

— $\{(\pi_i^n, v), (\pi_i^{n-1}, v), (v, \pi_i^{n-1}) : v \in V\} \cap T^{\gamma}$;
— $\{(\pi_i^{\ell}, v) : \ell = n-2, \ldots, 1 \text{ and } v \in V\} \cap T^{\gamma}$;
— $\{(i, v) : v \in V\} \cap T^{\gamma}$;
— if $n = 1$ include $\{(v, i) : v \in V\} \cap T^{\gamma}$;
— $\{(\sigma_i, v), (v, \sigma_i) : v \in V\} \cap T^{\gamma}$;
— $\{(\sigma_i^{\ell}, v) : \ell = 2, \ldots, n \text{ and } v \in V\} \cap T^{\gamma}$;
— $\{(\pi_j, v) : v \in V\} \cap T^{\gamma}$;
— $\{(j, v), (v, j) : v \in V\} \cap T^{\gamma}$;
— $\{(\sigma_j^{\ell}, v) : \ell = 1, \ldots, n-1 \text{ and } v \in V\} \cap T^{\gamma}$.

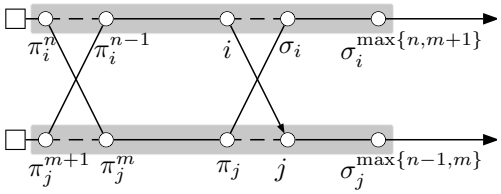
- Execution: replace $(\pi_i^n, \pi_i^{n-1}), (i, \sigma_i)$ and (j, π_j) with $(\pi_i^n, \sigma_i), (\pi_i^{n-1}, \pi_j)$ and (i, j).

B.3.5. *n*0REX with $n \geq 2$. Relocate a reversed path of n vertices within the same or into a different route.



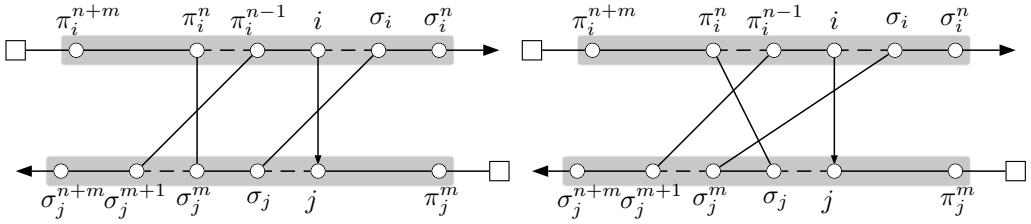
- Type: asymmetric.
- Pre-processing: none.
- Cost Computation: $\delta_{ij} = -c_{\pi_i^n \pi_i^{n-1}} - c_{i\sigma_i} - c_{j\sigma_j} + c_{\pi_i^n \sigma_i} + c_{\sigma_j \pi_i^{n-1}} + c_{ij}$.
- Feasibility Check: logical disjunction of
  — $(r_i = r_j) \wedge \bigwedge_{\ell=1}^{n} (j \neq \pi_i^{\ell})$.
  When $r_i = r_j$ and $\bigvee_{\ell=1}^{n-1} j = \pi_i^{\ell}$, the path to relocate overlaps with the destination position.
  When $r_i = r_j$ and $j = \pi_i^n$, the move could be reduced to a TWOPT induced by move generator (j, i) but would require a special handling in this context.
  — $(r_i \neq r_j) \wedge (i \neq 0) \wedge \bigwedge_{\ell=1}^{n-1} (\pi_i^{\ell} \neq 0) \wedge (q_{r_j} + q_i + \sum_{\ell=1}^{n-1} q_{\pi_i^{\ell}} \leq Q)$.
  The condition makes sure the depot is not relocated and the capacity constraint of the target route is respected.
- Update List: $\bigcup_{\ell=1}^{n} \pi_i^{\ell} \cup i \cup \bigcup_{\ell=1}^{n} \sigma_i^{\ell} \cup j \cup \bigcup_{\ell=1}^{n} \sigma_j^{\ell}$.
  The update can be restricted to move generators
  — $\{(\pi_i^{\ell}, v), (\pi_i^{\ell}, v), (i, v), (v, i) : \ell = n, \ldots, 1 \text{ and } v \in V\} \cap T^{\gamma}$;
  — $\{(\sigma_i^{\ell}, v), (\sigma_j^{\ell}, v) : \ell = 1, \ldots, n \text{ and } v \in V\} \cap T^{\gamma}$;
  — $\{(j, v), (v, j) : v \in V\} \cap T^{\gamma}$.
- Execution: replace $(\pi_i^n, \pi_i^{n-1}), (i, \sigma_i)$ and (j, σ_j) with $(\pi_i^n, \sigma_i), (\pi_i^{n-1}, \sigma_j)$ and (i, j).

B.3.6. nmEX with $1 \leq m \leq n$. Swap a path of n vertices with a path of m vertices within the same or between different routes.



- Type: asymmetric.
- Pre-processing: none.
- Cost Computation: $\delta_{ij} = -c_{\pi_i^n \pi_i^{n-1}} - c_{i\sigma_i} - c_{\pi_j^{m+1} \pi_j^m} - c_{j\pi_j} + c_{\pi_i^n \pi_j^m} + c_{\pi_j \sigma_i} + c_{\pi_j^{m+1} \pi_i^{n-1}} + c_{ij}$.
- Feasibility Check: logical disjunction of
  - $(r_i = r_j) \wedge \bigwedge_{\ell=1}^{n-1} (j \neq \pi_i^\ell) \wedge \bigwedge_{\ell=1}^{m+1} (j \neq \sigma_i^\ell)$.
    When $r_i = r_j$ and $\bigvee_{\ell=1}^{m} j = \sigma_i^\ell \vee \bigvee_{\ell=1}^{n-2} j = \pi_i^\ell$, the paths overlap.
    When $r_i = r_j$ and $j = \sigma_i^{m+1}$, the move could be reduced to a m0EX induced by move generator (σ_i, π_i^{n-1}) or to a n0EX induced by move generator (i, j) but would require a special handling in this context.
    When $r_i = r_j$ and $j = \pi_i^{n-1}$, vertex j is at the same time part of the path to move and destination of the movement.
  - $(r_i \neq r_j) \wedge (i \neq 0) \wedge \bigwedge_{\ell=1}^{n-1} (\pi_i^\ell \neq 0) \wedge \bigwedge_{\ell=1}^{m} (\pi_j^\ell \neq 0) \wedge (q_{r_j} + q_i + \sum_{\ell=1}^{n-1} q_{\pi_i^\ell} - \sum_{\ell=1}^{m} q_{\pi_j^\ell} \leq Q) \wedge (q_{r_i} - q_i - \sum_{\ell=1}^{n-1} q_{\pi_i^\ell} + \sum_{\ell=1}^{m} q_{\pi_j^\ell} \leq Q)$.
    The condition makes sure the depot is not relocated and the capacity constraints are not violated.
- Update List: $\bigcup_{\ell=1}^{n} \pi_i^\ell \cup i \cup \bigcup_{\ell=1}^{\max\{n,m+1\}} \sigma_i^\ell \cup \bigcup_{\ell=1}^{m+1} \pi_j^\ell \cup j \cup \bigcup_{\ell=1}^{\max\{n-1,m\}} \sigma_j^\ell$.
  The update can be restricted to move generators
  - $\{(\pi_i^n, v) : v \in V\} \cap T^\gamma$;
  - $\{(\pi_i^\ell, v) : \ell = n-1, \ldots, 1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(v, \pi_i^\ell) : \ell = n-1, \ldots, n-1-m \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(i, v) : v \in V\} \cap T^\gamma$;
  - if $n - m < 2$ include $\{(v, i) : v \in V\} \cap T^\gamma$;
  - $\{(\sigma_i, v), (v, \sigma_i) : v \in V\} \cap T^\gamma$;
  - $\{(\sigma_i^\ell, v) : \ell = 2, \ldots, n \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(v, \sigma_i^\ell) : \ell = 2, \ldots, m+1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(\pi_j^{m+1}, v) : v \in V\} \cap T^\gamma$;
  - $\{(\pi_j^\ell, v), (v, \pi_j^\ell) : \ell = m, \ldots, 1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(j, v), (v, j) : v \in V\} \cap T^\gamma$
  - $\{(\sigma_j^\ell, v) : \ell = 1, \ldots, n-1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(v, \sigma_j^\ell) : \ell = 1, \ldots, m \text{ and } v \in V\} \cap T^\gamma$.
- Execution: replace (π_i^n, π_i^{n-1}), (i, σ_i), (π_j^{m+1}, π_j^m) and (j, π_j) with (π_i^n, π_j^m), $(\pi_i^{n-1}, \pi_j^{m+1})$, (π_j, σ_i) and (i, j).

B.3.7. nmREX (and nmREX*) with $1 \leq m \leq n$. Swap a reversed path of n vertices with a path of m vertices within the same or between different routes (nmREX). If $m \geq 2$, we consider an additional variant that also reverses the path of m vertices (nmREX*).

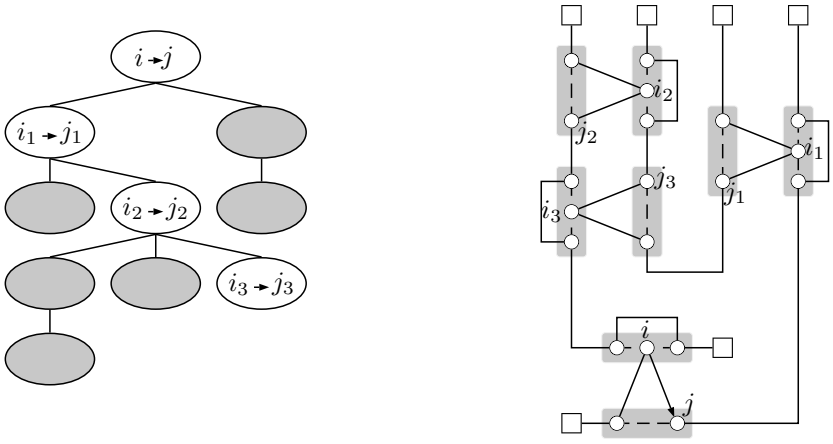


- Type: asymmetric.
- Pre-processing: none.
- Cost Computation:
  - nmREX*: $\delta_{ij} = -c_{\pi_i^n \pi_i^{n-1}} - c_{i\sigma_i} - c_{j\sigma_j} - c_{\sigma_j^m \sigma_j^{m+1}} + c_{\pi_i^n \sigma_j^m} + c_{\pi_i^{n-1} \sigma_j^{m+1}} + c_{\sigma_j \sigma_i} + c_{ij}$.
  - nmREX: $\delta_{ij} = -c_{\pi_i^n \pi_i^{n-1}} - c_{i\sigma_i} - c_{j\sigma_j} - c_{\sigma_j^m \sigma_j^{m+1}} + c_{\pi_i^n \sigma_j} + c_{\pi_i^{n-1} \sigma_j^{m+1}} + c_{\sigma_j^m \sigma_i} + c_{ij}$.
- Feasibility Check: logical disjunction of
  - $(r_i = r_j) \wedge \bigwedge_{\ell=1}^{n+m} (j \neq \pi_i^\ell)$.
    When $r_i = r_j$ and $\bigvee_{\ell=1}^{n+m-1} j = \sigma_i^\ell$, the paths overlap.
    When $r_i = r_j$ and $j = \sigma_i^{m+n}$
    - nmREX*: the move could be reduced to a TWOPT induced by move generator (j, i);
    - nmREX: the move could be reduced to a $n0$REX induced by move generator (i, j);

    but, in both cases, this would require a special handling.
  - $(r_i \neq r_j) \wedge (i \neq 0) \wedge \bigwedge_{\ell=1}^{n-1} (\pi_i^\ell \neq 0) \wedge \bigwedge_{\ell=1}^{m} (\sigma_j^\ell \neq 0) \wedge (q_{r_j} + q_i + \sum_{\ell=1}^{n-1} q_{\pi_i^\ell} - \sum_{\ell=1}^{m} q_{\sigma_j^\ell} \leq Q) \wedge (q_{r_i} - q_i - \sum_{\ell=1}^{n-1} q_{\pi_i^\ell} + \sum_{\ell=1}^{m} q_{\sigma_j^\ell} \leq Q)$.
    The condition makes sure the depot is not relocated and the capacity constraints are not violated.
- Update List: $\bigcup_{\ell=1}^{n+m} \pi_i^\ell \cup i \cup \bigcup_{\ell=1}^{n} \sigma_i^\ell \cup \bigcup_{\ell=1}^{m} \pi_j^\ell \cup j \cup \bigcup_{\ell=1}^{n+m} \sigma_j^\ell$.
  The update can be restricted to move generators
  - $\{(v, \pi_i^\ell) : \ell = n+m, \ldots, n+1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(\pi_i^\ell, v), (v, \pi_i^\ell) : \ell = n, \ldots, 1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(i, v), (v, i) : v \in V\} \cap T^\gamma$;
  - $\{(\sigma_i^\ell, v) : \ell = 1, \ldots, n \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(\sigma_j^\ell, v) : \ell = n+m, \ldots, m+1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(\sigma_j^\ell, v), (v, \sigma_j^\ell) : \ell = m, \ldots, 1 \text{ and } v \in V\} \cap T^\gamma$;
  - $\{(j, v), (v, j) : v \in V\} \cap T^\gamma$;
  - $\{(v, \pi_j^\ell) : \ell = 1, \ldots, m \text{ and } v \in V\} \cap T^\gamma$.
- Execution:
  - nmREX*: Replace $(\pi_i^n, \pi_i^{n-1}), (i, \sigma_i), (j, \sigma_j)$ and $(\sigma_j^m, \sigma_j^{m+1})$ with $(\pi_i^n, \sigma_j^m), (\pi_i^{n-1}, \sigma_j^{m+1}), (\sigma_j, \sigma_i)$ and (i, j). Reverse the paths between $\pi^n - 1$ and i and the path between σ_j and σ_j^m.

— nmREX: Replace $(\pi_i^n, \pi_i^{n-1}), (i, \sigma_i), (j, \sigma_j)$ and $(\sigma_j^m, \sigma_j^{m+1})$ with $(\pi_i^n, \sigma_j), (\pi_i^{n-1}, \sigma_j^{m+1}), (\sigma_j^m, \sigma_i)$ and (i, j). Reverse the path between $\pi^n - 1$ and i.

B.3.8. **EJCH.** Perform a sequence of 10EX applications.



- Type: asymmetric.
- Pre-processing: none.
- Cost Computation: same as for 10EX operator.
- Feasibility Check: a tree of nodes associated with 10EX moves is generated by using (i, j) as tree root. A path from the tree root (i, j) to any other node in the tree is a sequence of 10EX moves. Every sequence s stores a number of state variables defining the effect of its application on the current solution. The goal of the feasibility check is to find a sequence whose application generates a feasible and improving solution. In particular, each sequence s contains
  — the modified loads q_r^s for each route r affected by 10EX moves of s;
  — the change in the objective function δ_s due to the application of 10EX moves of s;
  — a set $\mathcal{F}_i^s$ storing customers that cannot be relocated because already involved in previous relocate moves within s. More precisely, the successors or predecessors of customers in $\mathcal{F}_i^s$ have changed in previous 10EX moves of s but the current structure of the solution does not reflect these changes. Note, in fact, that during the feasibility check we are only simulating the 10EX effects to find a feasible and improving sequence without really changing the solution;
  — finally, a set $\mathcal{F}_j^s$ storing customers that cannot be the target of a relocate move because already involved in previous relocate moves within s. More precisely, the predecessors of customers in $\mathcal{F}_j^s$ have changed in previous 10EX moves of s but, as described above, the current structure of the solution does not reflect these changes.

  Note that a different handling without $\mathcal{F}_i^s$ and $\mathcal{F}_j^s$ would require, for each tree node associated with a sequence s, to keep a copy of the solution.

A sequence s of 10EX moves generating a cost change δ_s in the objective function, ending with a move (i_n, j_n) that relocates customer $i_n \in V_c$ from route r_{i_n} to route r_{j_n} before vertex $j_n \in V$, is extended by scanning all customers i_{n+1} belonging to $r_{j_n} = r_{i_{n+1}}$ that satisfy the following joint conditions:

— $q^s_{r_{i_{n+1}}} - q_{i_{n+1}} \leq Q$.

The removal of i_{n+1} restores the feasibility of route $r_{i_{n+1}}$ that was violated by the previous insertion of j_n.

— $i_{n+1} \notin \mathcal{F}^s_i$.

Customer i_{n+1} can be relocated.

Every customer i_{n+1} satisfying the previous conditions is considered as the new starting point for a 10EX for which a potential endpoint is generated by scanning the currently active move generators T^γ that have i_{n+1} as the object of the relocation, i.e., $\{(i_{n+1}, j_{n+1}), j_{n+1} \in V_c\} \cap T^\gamma$. A customer $j_{n+1} \in V_c$ belonging to route $r_{j_{n+1}}$ is selected to be the target of the relocation if it satisfies the following joint conditions:

— $j_{n+1} \neq 0$.

The depot alone does not allow to identify a specific route.

— $r_{i_{n+1}} \neq r_{j_{n+1}}$.

Customer i_{n+1} is relocated into a route different from the origin one.

— $\delta_s + \delta_{i_{n+1} j_{n+1}} < 0$.

Sequence s still provides an improvement to the objective function.

— $j_{n+1} \notin \mathcal{F}^s_j$.

Customer j_{n+1} can be the target of a relocation.

Every customer j_{n+1} satisfying the previous conditions is considered as an endpoint for the (i_{n+1}, j_{n+1}) 10EX move and a new tree node s', son of the current one s, is created. State variables of s' are defined as follows:

— $q^{s'}_{r_{i_{n+1}}} = q^s_{r_{i_{n+1}}} - q_{i_{n+1}}$;
— $q^{s'}_{r_{j_{n+1}}} = q^s_{r_{j_{n+1}}} + q_{j_{n+1}}$;
— $\delta_{s'} = \delta_s + \delta_{i_{n+1} j_{n+1}}$;
— $\mathcal{F}^{s'}_i = \mathcal{F}^s_i \cup \{\pi_{i_{n+1}}, i_{n+1}, \sigma_{i_{n+1}}, \pi_{j_{n+1}}, j_{n+1}\}$;
— $\mathcal{F}^{s'}_j = \mathcal{F}^s_j \cup \{i_{n+1}, \sigma_{i_{n+1}}, j_{n+1}\}$.

The tree frontier is explored by following a best-δ-first strategy, that is, the sequence providing the greatest improvement is always extended first. This is obtained by using an additional heap data structure managing the tree nodes. As can be inferred by the above description, sequences are not limited in depth and the same route can be accessed by a 10EX move more than once. A limit is, however, imposed on the total maximum number of explored tree nodes which in the proposed implementation is $n_{EC} = 25$. Finally, the feasibility check step ends as soon as a feasible sequence is found, i.e., the last 10EX move does not violate the capacity of the target route, or the maximum number of tree nodes is explored.

- Update List: $\mathcal{F}_i^{s^*}$ with s^* a feasible improving sequence.

  For each 10EX (i,j) move generator composing the sequence s^*, the update can be restricted to move generators $\{(\pi_i,v),(i,v),(v,i),(\sigma_i,v),(v,\sigma_i),(j,v),(v,j),(\pi_j,v) : v \in V\} \cap T^{\gamma}$

- Execution: execute the 10EX moves of a feasible sequence. Note that, due to the restrictions imposed during the sequence space exploration, the order in which moves are executed does not affect the final result.

Appendix C: Computational details for $\mathbb{X}$ instances

Detailed results about computations on the $\mathbb{X}$ dataset can be found in Tables 3 – 5 and Figure 7. Moreover, to better assess whether the results obtained by FILO are statistically different with respect to competing algorithms, we followed the procedure used in Christiaens and Vanden Berghe (2020). In particular, we conducted a one-tailed Wilcoxon signed-rand test (Wilcoxon (1945)) in which we consider a null hypothesis H_0

$$H_0 : \textsc{AverageCost}(FILO) = \textsc{AverageCost}(X)$$

and an alternative hypothesis H_1

$$H_1 : \textsc{AverageCost}(FILO) > \textsc{AverageCost}(X)$$

where X can be ILS-SP, HGSADC, KGLS, and SISR. We tested the above hypotheses on small, medium, large and over all the instances. The p-values associated with the tested hypothesis are shown in Table 2 (left). The p-values for a similar analysis in which we compared FILO (long) with competing methods are shown in Table 2 (right).

A hypothesis is rejected when its associated p-value is greater than a significance level α. Failing to reject H_0 means that the average results of the two compared methods are not statistically different. On the other hand, when H_0 is rejected, the average results obtained by the methods are statistically different and the alternative hypothesis H_1 can be tested to find whether the average results obtained by FILO are statistically greater than those of the competing method. Rejecting H_1 implies that FILO performs better than the competing method.

When performing multiple comparisons involving the same data, the probability of erroneously rejecting a null hypothesis increases. To control these errors, the significance level α is typically adjusted to lower values. Bonferroni correction (Dunn (1961)) is a simple method used to adjust α when performing multiple

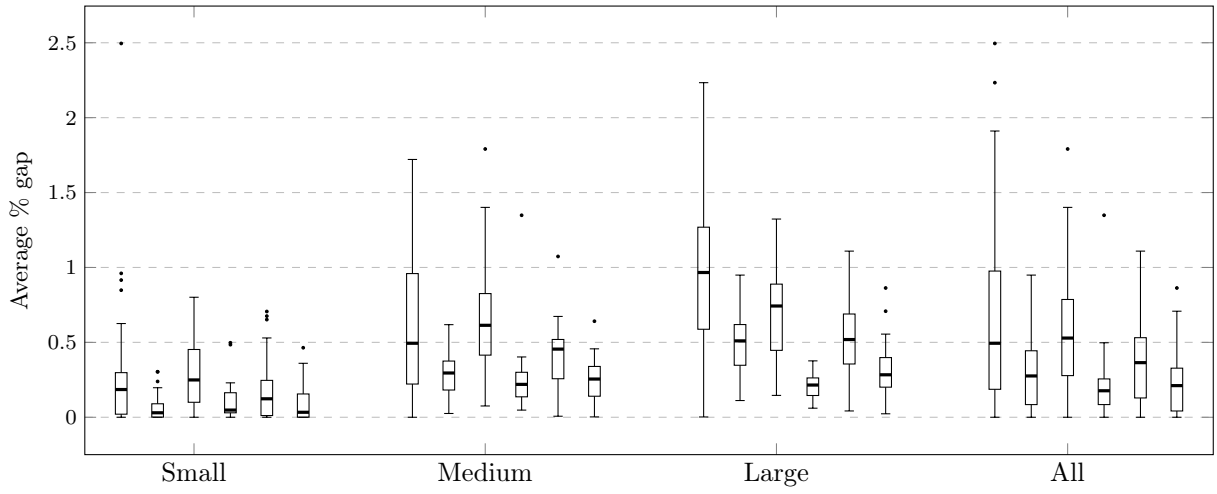


Figure 7 **Comparison of average gaps obtained by algorithms on the $\mathbb{X}$ dataset. For each group, boxplots, from left to right, are associated with: ILS-SP, HGSADC, KGLS, SISR, FILO, and FILO (long).**

Table 2 Computations on the 𝕏 dataset: p-values for FILO on the left and FILO (long) on the right.

	Small					Small			
	ILS-SP	HGSADC	KGLS	SISR		ILS-SP	HGSADC	KGLS	SISR
H_0	0.309491	**0.000273**	0.020427	0.064141	H_0	**0.000273**	0.808654	**0.000140**	0.130121
H_1	0.154746	0.999876	0.010214	0.969381	H_1	**0.000136**	0.404327	**0.000070**	0.065061
	Similar	Worse	Similar	Similar		Better	Similar	Better	Similar
	Medium					Medium			
	ILS-SP	HGSADC	KGLS	SISR		ILS-SP	HGSADC	KGLS	SISR
H_0	0.071754	**0.001737**	**0.000000**	**0.000066**	H_0	**0.001057**	0.046584	**0.000000**	0.346122
H_1	0.035877	0.999183	**0.000000**	0.999970	H_1	**0.000528**	0.023292	**0.000000**	0.830925
	Similar	Worse	Better	Worse		Better	Similar	Better	Similar
	Large					Large			
	ILS-SP	HGSADC	KGLS	SISR		ILS-SP	HGSADC	KGLS	SISR
H_0	**0.000335**	1.000000	**0.000234**	**0.000000**	H_0	**0.000000**	**0.000837**	**0.000000**	**0.002227**
H_1	**0.000167**	0.503678	**0.000117**	1.000000	H_1	**0.000000**	**0.000419**	**0.000000**	0.998963
	Better	Similar	Better	Worse		Better	Better	Better	Worse
	All					All			
	ILS-SP	HGSADC	KGLS	SISR		ILS-SP	HGSADC	KGLS	SISR
H_0	**0.000036**	0.007739	**0.000000**	**0.000000**	H_0	**0.000000**	**0.000111**	**0.000000**	0.065432
H_1	**0.000018**	0.996173	**0.000000**	1.000000	H_1	**0.000000**	**0.000056**	**0.000000**	0.967546
	Better	Similar	Better	Worse		Better	Better	Better	Similar

p-values in bold are associated with rejected hypothesis when $\alpha = 0.003125$.
The last row of each group contains a p-value interpretation when $\alpha = 0.003125$. In particular, FILO is not statistically different from the competing method when H_0 cannot be rejected (Similar), FILO is statistically better when both H_0 and H_1 are rejected (Better), and, finally, FILO is statistically worse when H_0 is rejected and H_1 is not rejected (Worse).

comparisons. In particular, given n comparisons, the significance level is set to α/n. In our case, for each FILO configuration, we tested a total number of $n = 8$ hypotheses corresponding to the partitioning of instances (Small, Medium, Large, and All) and to the two hypotheses (H_0 and H_1). Thus, by assuming an initial significance level $\alpha_0 = 0.025$, the adjusted value becomes $\alpha = \alpha_0/8 = 0.003125$.

As can ben seen from Table 2 (left)

- FILO performs better than ILS-SP on large instances and on all the 𝕏 dataset, and it has a similar performance on small and medium instances;
- FILO has a similar performance compared to HGSADC on large instances and on the whole 𝕏 dataset, however, HGSADC performs better on small and medium instances;
- FILO performs better than KGLS on medium and large instances, as well as on all the 𝕏 dataset, and it has a similar performance on small instances;
- finally, FILO has a similar performance compared to SISR on small instances, however, SISR performs better on medium, large and on all the 𝕏 dataset.

Table 2 (right) shows a similar analysis comparing FILO (long) with the other methods. In particular,

- FILO (long) performs better than ILS-SP on all partitions of instances;

- FILO (long) performs better than HGSADC on large and on all the $\mathbb{X}$ dataset, and it has a similar performance on small and medium instances;
- FILO (long) performs better than KGLS an all partitions of instances;
- finally, FILO has a similar performance compared to SISR on small, medium and on the whole $\mathbb{X}$ dataset, however, SISR performs better on large instances.

Table 3 Computations on small-sized $\mathbb{X}$ instances.

		ILS-SP		HGSADC		KGLS		SISR		FILO				FILO (long)			
ID	BKS	Avg	$\hat{t}^1$	Avg	$\hat{t}^1$	Avg	$\hat{t}$	Avg	$\hat{t}$	Best	Avg	Worst	t	Best	Avg	Worst	t
X-n101-k25	27591	0.00	0.06–0.07	0.00	0.85–0.91	0.21	2.69	0.00	0.58	0.00	0.00	0.22	1.10	0.00	0.00	0.00	11.89
X-n106-k14	26362	0.05	1.22–1.31	0.08	2.43–2.61	0.19	2.83	0.07	0.95	0.00	0.06	0.11	1.82	0.00	0.02	0.08	19.64
X-n110-k13	14971	0.00	0.12–0.13	0.00	0.97–1.04	0.00	2.94	0.00	0.73	0.00	0.00	0.00	1.63	0.00	0.00	0.00	16.43
X-n115-k10	12747	0.00	0.12–0.13	0.00	1.09–1.17	0.00	3.07	0.00	0.15	0.00	0.00	0.00	1.65	0.00	0.00	0.00	16.94
X-n120-k6	13332	0.04	1.03–1.11	0.00	1.40–1.50	0.00	3.21	0.00	1.16	0.00	0.00	0.00	1.74	0.00	0.00	0.00	18.36
X-n125-k30	55539	0.24	0.85–0.91	0.01	1.64–1.76	0.47	3.34	0.03	2.25	0.00	0.53	1.37	1.24	0.00	0.16	0.61	11.75
X-n129-k18	28940	0.20	1.15–1.24	0.03	1.64–1.76	0.11	3.45	0.03	1.09	0.00	0.06	0.19	1.58	0.00	0.03	0.05	17.72
X-n134-k13	10916	0.29	1.28–1.37	0.17	2.01–2.15	0.00	3.58	0.22	2.04	0.00	0.15	0.30	1.58	0.00	0.06	0.16	16.81
X-n139-k10	13590	0.10	0.97–1.04	0.00	1.40–1.50	0.00	3.72	0.04	1.45	0.00	0.00	0.00	1.98	0.00	0.00	0.00	21.84
X-n143-k7	15700	0.29	0.97–1.04	0.00	1.88–2.02	0.18	3.83	0.04	1.53	0.15	0.17	0.43	1.55	0.00	0.14	0.17	17.78
X-n148-k46	43448	0.01	0.49–0.52	0.00	1.95–2.09	0.35	3.96	0.05	2.04	0.00	0.12	0.35	1.36	0.00	0.01	0.33	15.10
X-n153-k22	21220	0.85	0.30–0.33	0.03	3.34–3.59	0.80	4.10	0.04	4.07	0.02	0.22	0.69	1.66	0.02	0.05	0.34	15.30
X-n157-k13	16876	0.00	0.49–0.52	0.00	1.95–2.09	0.00	4.20	0.02	2.69	0.00	0.00	0.00	2.57	0.00	0.00	0.00	28.74
X-n162-k11	14138	0.16	0.30–0.33	0.02	2.01–2.15	0.06	4.34	0.14	2.47	0.02	0.18	0.23	1.82	0.00	0.09	0.18	18.95
X-n167-k10	20557	0.25	0.55–0.59	0.03	2.25–2.41	0.16	4.47	0.02	2.33	0.00	0.05	0.17	1.91	0.00	0.00	0.02	20.19
X-n172-k51	45607	0.02	0.36–0.39	0.00	2.31–2.48	0.44	4.61	0.03	3.85	0.00	0.01	0.14	1.15	0.00	0.00	0.00	12.65
X-n176-k26	47812	0.92	0.67–0.72	0.30	4.62–4.96	0.39	4.71	0.08	3.78	0.00	0.65	1.25	1.35	0.00	0.19	1.13	14.28
X-n181-k23	25569	0.01	0.97–1.04	0.09	3.83–4.11	0.23	4.85	0.04	4.00	0.00	0.02	0.10	2.22	0.00	0.00	0.01	23.70
X-n186-k15	24145	0.17	1.03–1.11	0.01	3.59–3.85	0.20	4.98	0.14	2.91	0.01	0.10	0.30	1.55	0.00	0.04	0.21	15.94
X-n190-k8	16980	0.96	1.28–1.37	0.05	7.36–7.90	0.33	5.09	0.03	6.62	0.00	0.09	0.44	1.80	0.00	0.02	0.05	18.47
X-n195-k51	44225	0.02	0.55–0.59	0.04	3.71–3.98	0.49	5.23	0.17	4.44	0.00	0.18	0.53	1.25	0.00	0.06	0.26	12.03
X-n200-k36	58578	0.20	4.56–4.89	0.08	4.86–5.22	0.29	5.36	0.10	4.87	0.18	0.68	1.91	1.32	0.08	0.36	0.51	16.17
X-n204-k19	19565	0.31	0.67–0.72	0.03	3.22–3.46	0.52	5.47	0.50	3.56	0.00	0.13	0.70	1.41	0.00	0.01	0.12	14.90
X-n209-k16	30656	0.36	2.31–2.48	0.08	5.23–5.61	0.27	5.60	0.04	4.36	0.00	0.12	0.27	1.31	0.00	0.05	0.12	13.70
X-n214-k11	10856	2.50	1.40–1.50	0.20	6.20–6.66	0.67	5.74	0.48	6.33	0.13	0.43	1.11	1.37	0.04	0.23	0.93	14.22
X-n219-k73	117595	0.00	0.49–0.52	0.01	4.68–5.02	0.09	5.87	0.05	5.82	0.00	0.00	0.01	4.94	0.00	0.00	0.00	54.01
X-n223-k34	40437	0.24	5.17–5.55	0.15	5.05–5.42	0.63	5.98	0.23	5.53	0.08	0.28	0.66	1.22	0.00	0.16	0.27	12.84
X-n228-k23	25742	0.21	1.46–1.57	0.14	5.96–6.39	0.34	6.12	0.19	7.64	0.03	0.21	0.45	1.33	0.00	0.16	0.25	13.84
X-n233-k16	19230	0.55	1.82–1.96	0.30	4.13–4.44	0.58	6.25	0.21	5.89	0.16	0.42	0.71	1.56	0.00	0.30	0.54	17.24
X-n237-k14	27042	0.14	2.13–2.28	0.09	5.41–5.81	0.38	6.36	0.18	5.24	0.00	0.03	0.50	2.13	0.00	0.01	0.16	23.67
X-n242-k48	82751	0.15	10.82–11.61	0.24	7.54–8.09	0.47	6.49	0.16	7.20	0.09	0.31	0.58	1.51	0.00	0.16	0.39	16.81
X-n247-k50	37274	0.63	1.28–1.37	0.03	12.40–13.31	0.17	6.63	0.13	13.38	0.06	0.71	1.30	1.56	0.00	0.46	1.05	16.08
Mean		0.31	1.46 – 1.57	0.07	3.65 – 3.92	0.28	4.66	0.11	3.78	0.03	0.18	0.47	1.69	0.00	0.09	0.25	18.06

[1] computed by considering the range of single-thread rating of compatible CPUs.

Table 4 Computations on medium-sized $\mathbb{X}$ instances.

		ILS-SP		HGSADC		KGLS		SISR		FILO				FILO (long)			
ID	BKS	Avg	$\hat{t}$[1]	Avg	$\hat{t}$[1]	Avg	$\hat{t}$	Avg	$\hat{t}$	Best	Avg	Worst	t	Best	Avg	Worst	t
X-n251-k28	38684	0.40	6.57–7.05	0.29	7.11–7.63	0.60	6.74	0.28	7.13	0.17	0.36	0.66	1.57	0.00	0.25	0.38	17.38
X-n256-k16	18839	0.24	1.22–1.31	0.22	3.95–4.24	0.32	6.87	0.26	8.36	0.22	0.22	0.23	2.13	0.22	0.22	0.22	23.80
X-n261-k13	26558	1.17	4.07–4.37	0.27	7.72–8.29	0.55	7.00	0.32	8.58	0.17	0.48	1.27	1.56	0.01	0.31	0.50	16.98
X-n266-k58	75478	0.11	6.08–6.53	0.37	13.01–13.96	0.65	7.14	0.19	7.86	0.18	0.51	0.87	1.80	0.07	0.38	0.57	20.67
X-n270-k35	35291	0.21	5.53–5.94	0.22	6.81–7.31	0.46	7.25	0.20	8.29	0.05	0.26	0.44	1.40	0.05	0.15	0.27	14.17
X-n275-k28	21245	0.05	2.19–2.35	0.17	7.29–7.83	0.26	7.38	0.11	9.67	0.00	0.08	0.37	1.87	0.00	0.02	0.39	20.23
X-n280-k17	33503	0.80	5.84–6.26	0.31	11.61–12.46	0.61	7.52	0.37	12.87	0.05	0.46	0.76	1.47	0.04	0.36	0.52	15.52
X-n284-k15	20215	1.16	5.23–5.61	0.35	12.10–12.99	0.86	7.62	0.35	11.13	0.15	0.53	1.03	1.58	0.06	0.26	0.53	15.51
X-n289-k60	95151	0.31	9.79–10.51	0.33	12.95–13.90	0.77	7.76	0.21	10.40	0.31	0.60	0.86	1.77	0.24	0.40	0.61	20.07
X-n294-k50	47161	0.20	7.54–8.09	0.21	8.94–9.59	0.61	7.89	0.24	10.69	0.14	0.32	0.59	1.12	0.12	0.23	0.35	11.80
X-n298-k31	34231	0.37	4.19–4.50	0.18	6.63–7.11	0.30	8.00	0.13	10.55	0.00	0.27	0.46	1.18	0.01	0.19	0.31	12.36
X-n303-k21	21738	0.73	8.63–9.27	0.52	10.52–11.29	0.64	8.14	0.18	12.58	0.21	0.45	0.89	1.23	0.09	0.33	0.66	11.91
X-n308-k13	25859	0.94	5.77–6.20	0.14	9.30–9.98	0.82	8.27	1.35	18.69	0.01	0.51	1.83	1.62	0.02	0.46	1.42	18.27
X-n313-k71	94044	0.27	10.64–11.42	0.24	13.62–14.62	0.85	8.41	0.15	13.75	0.29	0.52	0.81	1.39	0.15	0.32	0.57	15.23
X-n317-k53	78355	0.00	5.23–5.61	0.04	13.62–14.62	0.08	8.51	0.05	16.00	0.00	0.01	0.04	3.02	0.00	0.00	0.01	31.68
X-n322-k28	29834	0.53	8.94–9.59	0.41	9.24–9.92	0.68	8.65	0.31	12.29	0.25	0.48	0.88	1.25	0.05	0.34	0.54	13.19
X-n327-k20	27532	1.02	11.61–12.46	0.35	11.06–11.88	0.44	8.78	0.36	15.71	0.17	0.47	0.81	1.71	0.09	0.29	0.55	19.83
X-n331-k15	31102	0.43	9.54–10.24	0.19	14.83–15.92	0.13	8.89	0.08	14.84	0.00	0.02	0.30	2.00	0.00	0.00	0.01	21.59
X-n336-k84	139111	0.25	13.01–13.96	0.30	23.10–24.80	1.40	9.03	0.19	16.58	0.36	0.61	0.93	1.41	0.19	0.38	0.55	13.80
X-n344-k43	42050	0.56	13.74–14.75	0.38	13.19–14.16	0.83	9.24	0.26	15.64	0.28	0.58	0.80	1.35	0.03	0.33	0.56	12.67
X-n351-k40	25896	0.98	15.32–16.44	0.46	20.49–21.99	1.03	9.43	0.33	19.27	0.33	0.67	1.03	1.36	0.26	0.45	0.72	13.06
X-n359-k29	51505	1.11	29.73–31.91	0.42	21.21–22.77	0.94	9.64	0.14	16.80	0.16	0.48	0.85	1.51	0.00	0.24	0.45	16.30
X-n367-k17	22814	0.83	7.96–8.55	0.11	13.37–14.36	0.69	9.86	0.09	26.26	0.00	0.19	0.68	1.65	0.00	0.04	0.15	17.03
X-n376-k94	147713	0.00	4.32–4.63	0.03	17.20–18.47	0.11	10.10	0.05	23.28	0.00	0.01	0.03	4.17	0.00	0.01	0.03	43.35
X-n384-k52	65940	0.66	20.97–22.51	0.50	24.44–26.23	0.70	10.32	0.25	18.84	0.19	0.46	0.74	1.71	0.13	0.27	0.51	18.01
X-n393-k38	38260	0.52	12.64–13.57	0.30	17.39–18.66	0.32	10.56	0.35	22.11	0.10	0.29	0.61	1.41	0.03	0.12	0.37	14.97
X-n401-k29	66187	0.80	36.72–39.41	0.27	30.09–32.30	0.58	10.78	0.09	27.64	0.09	0.22	0.44	1.98	0.02	0.10	0.20	19.96
X-n411-k19	19712	1.23	14.47–15.53	0.16	21.09–22.64	1.79	11.05	0.29	42.48	0.22	0.52	1.49	1.82	0.24	0.37	0.84	18.87
X-n420-k130	107798	0.04	13.49–14.49	0.12	32.34–34.71	0.51	11.29	0.08	34.84	0.08	0.25	0.50	1.16	0.04	0.14	0.26	11.29
X-n429-k61	65467	0.43	23.22–24.93	0.28	25.23–27.08	0.54	11.53	0.19	25.46	0.19	0.39	0.75	1.40	0.01	0.19	0.33	14.55
X-n439-k37	36391	0.14	24.07–25.84	0.17	20.97–22.51	0.31	11.80	0.23	30.62	0.01	0.09	0.25	1.64	0.01	0.02	0.10	17.74
X-n449-k29	55254	1.72	36.41–39.09	0.54	39.45–42.35	0.89	12.07	0.28	27.64	0.34	0.62	1.02	1.46	0.18	0.34	0.67	15.14
X-n459-k26	24145	1.31	36.84–39.54	0.53	26.02–27.93	0.39	12.34	0.40	41.10	0.09	0.31	0.86	1.57	0.00	0.21	0.39	16.51
X-n469-k138	221824	0.16	22.07–23.69	0.36	52.70–56.57	0.73	12.61	0.18	34.91	0.73	1.07	1.36	1.62	0.47	0.64	0.80	16.03
X-n480-k70	89449	0.47	30.64–32.89	0.35	40.73–43.72	0.58	12.90	0.12	36.73	0.15	0.36	0.55	1.63	0.02	0.21	0.41	17.09
X-n491-k59	66487	1.11	31.73–34.06	0.62	43.71–46.92	1.16	13.20	0.24	37.39	0.29	0.53	0.84	1.40	0.12	0.32	0.50	13.88
Mean		0.59	14.05 – 15.09	0.30	18.42 – 19.77	0.64	9.40	0.25	19.64	0.17	0.39	0.75	1.66	0.08	0.25	0.45	17.51

[1] computed by considering the range of single-thread rating of compatible CPUs.

Table 5 Computations on large-sized $\mathbb{X}$ instances.

		ILS-SP		HGSADC		KGLS		SISR		FILO				FILO (long)			
ID	BKS	Avg	$\hat{t}^1$	Avg	$\hat{t}^1$	Avg	$\hat{t}$	Avg	$\hat{t}$	Best	Avg	Worst	t	Best	Avg	Worst	t
X-n502-k39	69226	0.17	49.12–52.72	0.15	38.66–41.50	0.15	13.50	0.07	44.30	0.00	0.04	0.10	2.52	0.00	0.03	0.07	25.92
X-n513-k21	24201	0.96	21.28–22.84	0.40	20.12–21.60	0.36	13.79	0.38	56.08	0.06	0.35	0.86	1.84	0.00	0.15	0.40	19.54
X-n524-k153	154593	0.27	16.60–17.81	0.25	49.06–52.66	0.48	14.09	0.14	110.12	0.08	0.50	0.99	1.38	0.04	0.24	0.56	13.68
X-n536-k96	94868	0.88	37.75–40.52	0.49	65.35–70.15	1.06	14.41	0.32	54.33	0.71	0.83	1.00	1.58	0.53	0.71	0.83	16.02
X-n548-k50	86700	0.20	38.90–41.76	0.34	51.18–54.94	0.25	14.74	0.11	46.91	0.00	0.10	0.25	1.86	0.00	0.05	0.13	19.60
X-n561-k42	42717	0.97	41.88–44.96	0.35	36.84–39.54	0.64	15.09	0.35	53.68	0.21	0.43	0.74	1.32	0.08	0.29	0.54	13.49
X-n573-k30	50673	0.99	68.08–73.08	0.48	114.40–122.80	0.68	15.41	0.26	82.19	0.22	0.37	0.66	1.91	0.15	0.25	0.38	20.00
X-n586-k159	190316	0.32	47.72–51.22	0.27	106.56–114.39	0.41	15.76	0.15	62.77	0.45	0.70	0.98	1.62	0.22	0.37	0.63	16.39
X-n599-k92	108451	0.86	44.38–47.63	0.57	76.53–82.15	0.87	16.11	0.22	54.84	0.30	0.51	0.74	1.68	0.19	0.30	0.45	17.58
X-n613-k62	59545	1.51	45.47–48.81	0.70	71.30–76.54	0.93	16.49	0.31	64.08	0.39	0.67	1.11	1.16	0.06	0.40	0.76	11.35
X-n627-k43	62173	1.18	98.90–106.16	0.56	145.71–156.41	0.71	16.87	0.23	64.95	0.17	0.34	0.54	1.80	0.09	0.20	0.41	18.65
X-n641-k35	63705	1.41	85.35–91.61	0.76	96.53–103.62	0.52	17.24	0.23	67.28	0.13	0.38	0.66	1.88	0.11	0.22	0.45	19.10
X-n655-k131	106780	0.00	28.69–30.80	0.11	91.49–98.20	0.18	17.62	0.06	79.72	0.01	0.04	0.08	3.23	0.00	0.02	0.05	33.44
X-n670-k130	146332	0.92	37.20–39.93	0.61	160.54–172.33	1.00	18.02	0.27	144.67	0.66	1.11	1.70	1.42	0.43	0.86	1.24	14.16
X-n685-k75	68225	1.12	44.86–48.16	0.63	95.25–102.25	1.03	18.43	0.21	98.27	0.44	0.63	0.88	1.42	0.16	0.43	0.67	13.65
X-n701-k44	81923	1.37	127.72–137.09	0.69	153.91–165.22	0.77	18.86	0.17	89.10	0.36	0.53	0.69	1.57	0.06	0.28	0.51	15.91
X-n716-k35	43387	1.81	137.26–147.34	0.59	160.66–172.46	0.89	19.26	0.22	115.14	0.49	0.70	1.06	1.67	0.14	0.28	0.50	17.14
X-n733-k159	136190	0.63	67.84–72.82	0.29	148.63–159.54	0.86	19.72	0.15	104.16	0.18	0.36	0.48	1.26	0.09	0.21	0.31	12.84
X-n749-k98	77314	1.24	77.32–83.00	0.71	190.81–204.82	1.32	20.15	0.25	106.41	0.54	0.71	0.88	1.45	0.28	0.42	0.63	13.90
X-n766-k71	114456	1.12	147.17–157.97	0.60	232.82–249.91	0.84	20.61	0.27	126.85	0.46	0.68	1.07	1.59	0.24	0.43	0.76	15.76
X-n783-k48	72394	1.84	143.16–153.67	0.85	163.94–175.98	0.89	21.07	0.37	123.80	0.34	0.61	0.87	1.75	0.18	0.35	0.57	18.54
X-n801-k40	73331	0.92	262.97–282.28	0.55	175.80–188.71	0.26	21.55	0.14	99.72	0.02	0.23	0.41	1.80	-0.02	0.11	0.26	18.28
X-n819-k171	158121	0.82	90.51–97.16	0.49	227.53–244.24	0.89	22.04	0.19	125.47	0.65	0.84	1.05	1.39	0.44	0.56	0.67	14.31
X-n837-k142	193737	0.67	105.28–113.02	0.38	281.69–302.38	0.76	22.52	0.12	121.32	0.37	0.53	0.70	1.78	0.21	0.31	0.42	18.57
X-n856-k95	88990	0.32	93.43–100.29	0.28	175.31–188.19	0.34	23.03	0.16	116.38	0.02	0.14	0.28	1.76	-0.00	0.06	0.16	18.00
X-n876-k59	99303	1.12	248.80–267.07	0.59	301.14–323.26	0.95	23.57	0.18	158.13	0.32	0.47	0.63	1.74	0.15	0.26	0.38	17.46
X-n895-k37	53928	1.91	249.35–267.66	0.95	195.68–210.05	0.73	24.09	0.29	154.56	0.33	0.58	0.94	1.77	0.02	0.26	0.55	17.93
X-n916-k207	329179	0.54	137.44–147.53	0.31	340.90–365.93	0.58	24.65	0.10	156.60	0.50	0.70	0.85	1.75	0.17	0.39	0.55	18.98
X-n936-k151	132812	1.29	123.10–132.13	0.53	323.09–346.81	0.87	25.19	0.23	300.18	0.39	0.91	1.38	1.32	0.24	0.50	0.79	12.70
X-n957-k87	85469	0.55	189.17–203.06	0.41	263.15–282.47	0.34	25.76	0.18	147.22	0.04	0.14	0.24	1.86	0.00	0.09	0.20	18.93
X-n979-k58	118988	1.06	417.73–448.41	0.43	336.76–361.49	0.63	26.35	0.11	201.19	0.26	0.37	1.02	2.36	0.12	0.23	0.35	23.76
X-n1001-k43	72369	2.23	481.93–517.32	0.81	333.72–358.23	0.95	26.94	0.22	206.79	0.41	0.65	0.83	1.65	0.15	0.33	0.53	16.40
Mean		0.98	118.95 – 127.68	0.50	163.28 – 175.27	0.69	19.47	0.21	110.54	0.30	0.50	0.77	1.72	0.14	0.30	0.49	17.56

[1] computed by considering the range of single-thread rating of compatible CPUs.
New best solutions: (X-n801-k40, 73313);(X-n856-k95, 88989)

Appendix D: Computational details for very large-scale instances

This section contains computational details associated with large-scale datasets. In particular, Figures 8 – 10 show by means of boxplots the average gaps obtained by algorithms on the $\mathbb{B}$, $\mathbb{K}$, and $\mathbb{Z}$ dataset, respectively. Average solution values are analyzed by conducting analyses similar to those for the $\mathbb{X}$ dataset described in Section C. In particular, the null hypothesis H_0 and the alternative hypothesis H_1 are the same. However, contrarily to the previous analysis, we did not partition dataset instances in smaller groups. Thus the total number of analysis performed for each dataset is $n = 2$, one for each hypothesis. The initial confidence level $\alpha_0 = 0.025$ is thus adjusted through the Bonferroni correction to $\alpha = 0.025/2 = 0.0125$. Tables 6 and 7 show the p-values associated with the $\mathbb{B}$ and $\mathbb{K}$ datasets, respectively. Finally, due to the very limited number of instances of the $\mathbb{Z}$ dataset, the Wilcoxon signed-rank test cannot be used because it cannot give a significant result.

As can be seen from Table 6

- FILO performs better than KGLS$^{\text{XXL}}$ and it has a performance similar to that of KGLS$^{\text{XXL}}$ (long);
- FILO (long) performs better than KGLS$^{\text{XXL}}$ and KGLS$^{\text{XXL}}$ (long).

As can be seen from Table 7, both FILO and FILO (long) performs better than KGLS$^{\text{XXL}}$ and KGLS$^{\text{XXL}}$ (long).

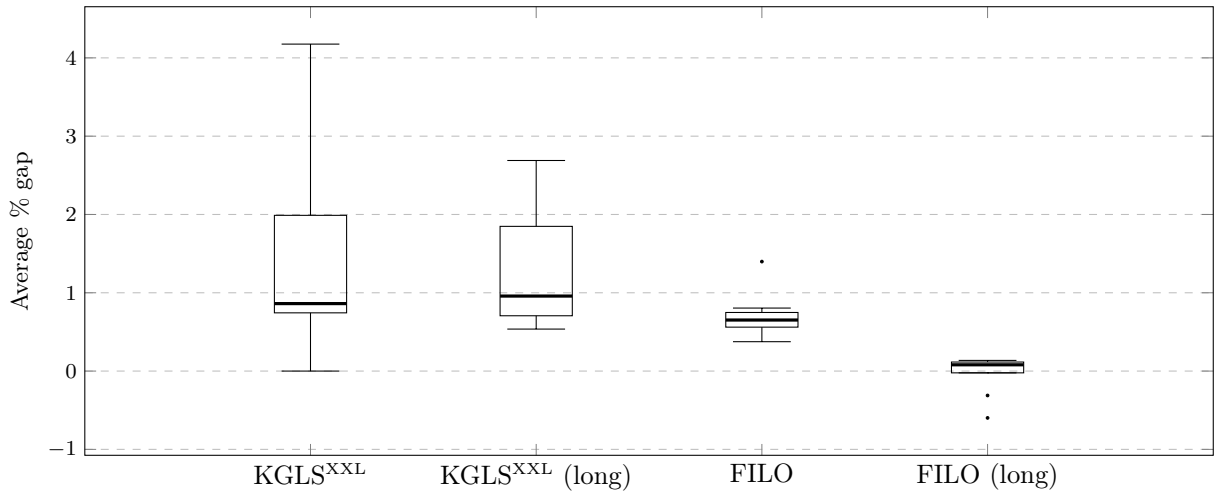


Figure 8 **Comparison of average gaps obtained by algorithms on the $\mathbb{B}$ dataset.**

Table 6 **Computations on the $\mathbb{B}$ dataset: p-values for FILO on the left and for FILO (long) on the right.**

	KGLS$^{\text{XXL}}$	KGLS$^{\text{XXL}}$ (long)
H_0	**0.001953**	0.037109
H_1	**0.000977**	0.018555
	Better	Similar

	KGLS$^{\text{XXL}}$	KGLS$^{\text{XXL}}$ (long)
H_0	**0.001953**	**0.001953**
H_1	**0.000977**	**0.000977**
	Better	Better

p-values in bold are associated with rejected hypothesis when $\alpha = 0.0125$.
The last row contains a p-value interpretation when $\alpha = 0.0125$. In particular, FILO is not statistically different from the competing method when H_0 cannot be rejected (Similar), FILO is statistically better when both H_0 and H_1 are rejected (Better), and, finally, FILO is statistically worse when H_0 is rejected and H_1 is not rejected (Worse).

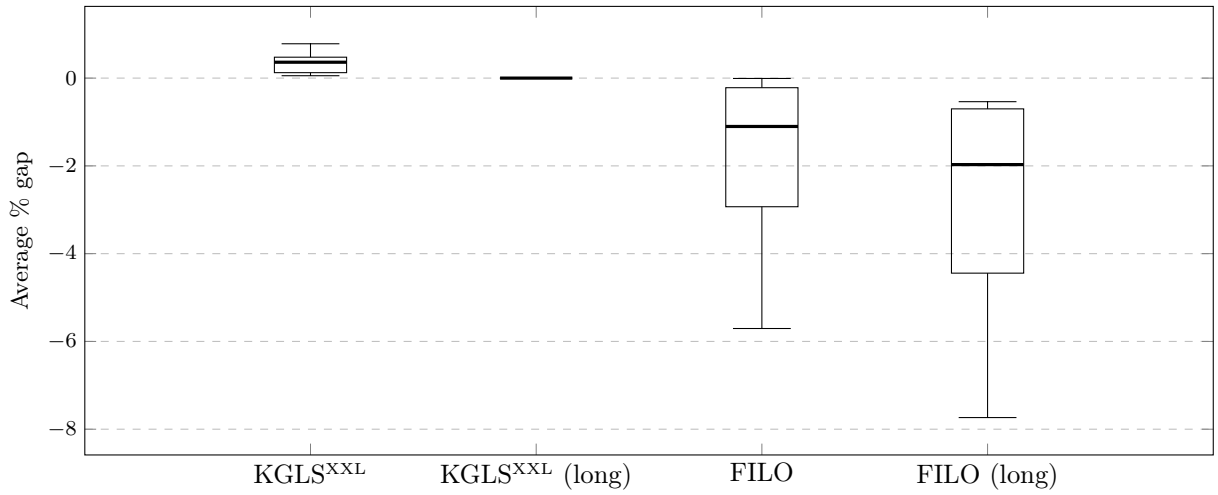


Figure 9 **Comparison of average gaps obtained by algorithms on the $\mathbb{K}$ dataset.**

Table 7 **Computations on the $\mathbb{K}$ dataset: p-values for FILO on the left and for FILO (long) on the right.**

	KGLS$^{\text{XXL}}$	KGLS$^{\text{XXL}}$ (long)
H_0	**0.0078125**	**0.0078125**
H_1	**0.00390625**	**0.00390625**
	Better	Better

	KGLS$^{\text{XXL}}$	KGLS$^{\text{XXL}}$ (long)
H_0	**0.0078125**	**0.0078125**
H_1	**0.00390625**	**0.00390625**
	Better	Better

p-values in bold are associated with rejected hypothesis when $\alpha = 0.0125$.
The last row contains a p-value interpretation when $\alpha = 0.0125$. In particular, FILO is not statistically different from the competing method when H_0 cannot be rejected (Similar), FILO is statistically better when both H_0 and H_1 are rejected (Better), and, finally, FILO is statistically worse when H_0 is rejected and H_1 is not rejected (Worse).

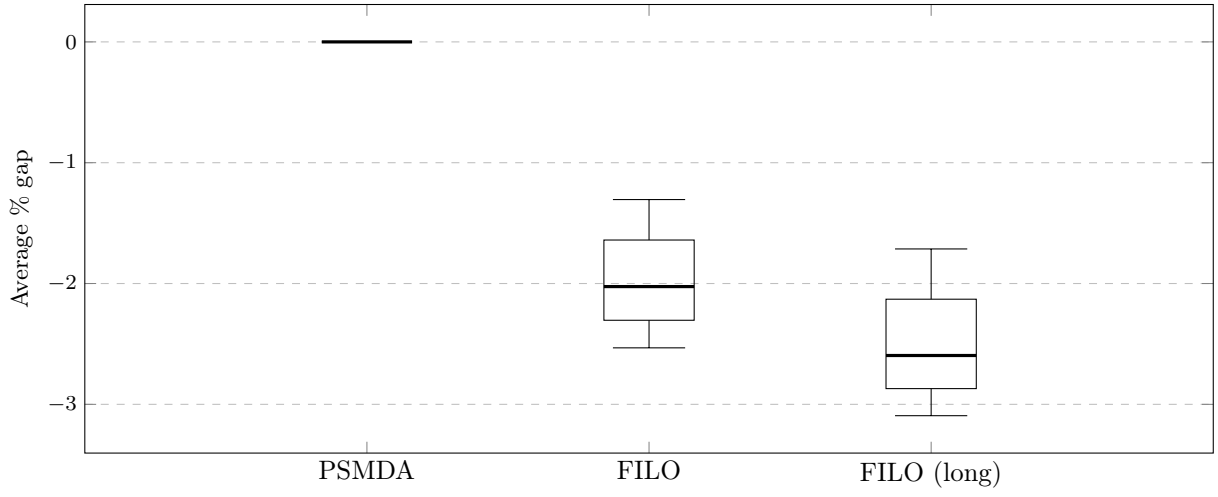


Figure 10 **Comparison of average gaps obtained by algorithms on the $\mathbb{Z}$ dataset.**

References

Arnold F, Gendreau M, Sörensen K, 2019 *Efficiently solving very large-scale routing problems. Computers & Operations Research* 107:32 – 42, URL http://dx.doi.org/10.1016/j.cor.2019.03.006.

Christiaens J, Vanden Berghe G, 2020 *Slack induction by string removals for vehicle routing problems. Transportation Science* 54(2):417–433, URL http://dx.doi.org/10.1287/trsc.2019.0914.

Dunn OJ, 1961 *Multiple comparisons among means. Journal of the American Statistical Association* 56(293):52–64, URL http://www.jstor.org/stable/2282330.

Wilcoxon F, 1945 *Individual comparisons by ranking methods. Biometrics Bulletin* 1(6):80–83, URL http://www.jstor.org/stable/3001968.