Iterated Local Search for the Trigger Arc Travelling Salesman Problem

Admir Kadriu and Kadri Sylejmani

Faculty of Electrical and Computer Engineering,
University of Prishtina, Prishtina, Kosovo
admir.kadriu@uni-pr.edu, kadri.sylejmani@uni-pr.edu

Abstract. The Trigger Arc Travelling Salesman Problem (TATSP) is a recently introduced variant of the TSP in which arc costs depend dynamically on the traversal order: traversing a *trigger* arc before a *target* arc may change the target's cost, with only the last trigger encountered being active. The problem arises in warehouse routing with compactable storage systems and has so far only been addressed by an integer linear programming formulation that fails to scale beyond approximately 25 nodes. We propose an Iterated Local Search (ILS) with a simulated annealing acceptance criterion, using first-improvement hill climbing over an Or-opt neighbourhood adapted for trigger-arc semantics via a delta cost evaluation scheme that traces trigger fan-out. Computational experiments on 55 benchmark instances from the MESS 2024 competition show that the approach achieves an average gap of 1.79% on the dense C1 instances (20–60 nodes) and 0.07% on the sparser C2 instances (18–142 nodes), establishing 5 new best-known solutions on the latter family while scaling well beyond the reach of exact methods.

Keywords: Travelling Salesman Problem · Iterated Local Search · Metaheuristics · Dynamic Arc Costs

1 Introduction and Problem Definition

The Travelling Salesman Problem (TSP) is one of the most extensively studied problems in combinatorial optimisation [1]. In the classical formulation, a salesman must visit a set of cities exactly once and return to the starting city, minimising the total travel cost. Numerous variants have been proposed to capture real-world complexities, including problems with traversal-dependent edge costs or structure [2,3], yet most assume that arc costs are fixed and independent of the solution structure. Cerrone et al. [4] recently introduced the *Trigger Arc Travelling Salesman Problem* (TATSP), a variant in which arc costs depend dynamically on the traversal order of arcs in the tour. The TATSP is motivated by the optimisation of picking routes in warehouses equipped with compactable storage systems, where mobile shelving units slide on rails: opening one aisle to retrieve items may close adjacent aisles, thereby changing the cost of accessing

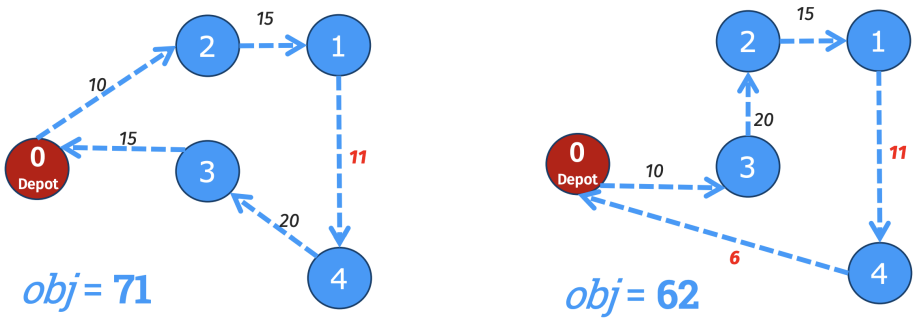

Fig. 1. Two tours on a 5-node TATSP instance with five trigger relations (Table 1). Triggered costs are shown in red. Left: tour 0-2-1-4-3-0, cost 71. Right: tour 0-3-2-1-4-0, cost 62 (optimal).

Table 1. Trigger relations for the example instance in Figure 1.

Relation	Trigger $\rightarrow$ Target	Triggered cost
r_1	$(0,2) \rightarrow (1,4)$	3
r_2	$(2,1) \rightarrow (1,4)$	11
r_3	$(2,1) \rightarrow (3,2)$	3
r_4	$(2,1) \rightarrow (4,0)$	6
r_5	$(3,0) \rightarrow (1,4)$	5

them later in the route. More generally, the model captures real-world scenarios in which the traversal of an arc by a vehicle can lead to modifications in the structure of the road graph [4].

We now state the problem formally, following [4]. Let $G = (N, A)$ be a directed graph with node set $N = \{0, 1, \ldots, n\}$, where node 0 is the depot, and arc set A. Each arc $a \in A$ has a base cost $c(a)$. For each arc a, there is a (possibly empty) set of *trigger relations* $R_a = \{(a_1, a) \mid a_1 \in A\}$. A relation $r = (a_i, a_j)$ carries a modified cost $c(r)$ and is said to be *active* in a tour T if both a_i and a_j belong to T and a_i is traversed before a_j, with no other relation $r' = (a_k, a_j) \in R_{a_j}$ such that a_k appears between a_i and a_j in T. In other words, only the *last* trigger arc encountered before a target arc determines its cost — a semantics we refer to as "last trigger wins." If no relation is active for an arc, it retains its base cost. The objective is to find a minimum-cost Hamiltonian cycle starting and ending at the depot.

Figure 1 illustrates the effect of trigger relations on a 5-node instance with five relations (Table 1). Tour $0 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 3 \rightarrow 0$ (left) traverses arc $(2,1)$ before arc $(1,4)$, activating r_2 and changing the cost of $(1,4)$ from 10 to 11; the earlier r_1 activated by $(0,2)$ is overridden, giving a total cost of 71. Tour $0 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 4 \rightarrow 0$ (right) also activates r_2 on $(1,4)$ and additionally r_4 on $(4,0)$, reducing its cost from 10 to 6, yielding the optimal cost of 62.

The only existing exact approach is the integer linear programming (ILP) formulation proposed by Cerrone et al. [4]. Their computational experiments on complete directed graphs with up to 25 nodes and relation densities up to $16n^2$ show that the ILP solver struggles as instance size and relation density grow: at 25 nodes, MIP gaps reach 36% in Balanced scenarios and up to 42% in Decrease scenarios within a 1000-second time limit. Even at 20 nodes with high relation density, some instances cannot be solved to optimality. The authors explicitly note that "the implementation of heuristic techniques capable of quickly identifying good solutions to the problem could be interesting and useful."

In this paper, we propose an Iterated Local Search (ILS) [6] for the TATSP that combines a first-improvement hill climbing procedure using an Or-opt neighbourhood with a Simulated Annealing acceptance criterion [5]. We evaluate our method on the 55 benchmark instances from the MESS 2024 competition [7] and compare against the best-known solutions as well as the concurrent GRASP approach of Salvà Soler and de Lambertye [8]. Our ILS achieves an average gap of 1.79% on the dense C1 instances and 0.07% on the sparser C2 instances, establishing 5 new best-known solutions on the latter family while scaling to instances with up to 142 nodes.

2 Solution Approach

We propose an Iterated Local Search (ILS) [6] for the TATSP. The algorithm starts from a random initial tour, improves it via a first-improvement hill climbing procedure using an Or-opt neighbourhood adapted for trigger-arc semantics, and escapes local optima through segment-based perturbations combined with a simulated annealing (SA) [5] acceptance criterion. A key technical challenge is cost evaluation: because trigger relations link distant arcs, even a single-node move can change the effective cost of arcs far from the modification site. We address this with a delta evaluation scheme that traces trigger fan-out and falls back to full recomputation only when the affected region is large.

2.1 Cost Evaluation

In a standard TSP, moving a node affects at most a constant number of arcs, enabling $O(1)$ delta evaluation. In the TATSP, this property does not hold. Traversing an arc t before an arc a may change the cost of a to a triggered value c', and only the *last* such trigger encountered before a is active [4]. Consequently, reordering a single node can shift trigger–target orderings throughout the tour, potentially altering costs at positions far from the modification.

Full evaluation. Given a tour $\pi = (v_0, v_1, \ldots, v_n, v_0)$, the cost of each arc $a_i = (v_i, v_{i+1})$ is determined by walking backward through the previously traversed arcs to find the last trigger for a_i. If such a trigger exists, a_i takes its triggered cost; otherwise it retains its base cost. This yields an $O(n^2)$ worst-case procedure when trigger lookups are supported by a hash map (see Complexity below).

Delta evaluation. To avoid full recomputation on every candidate move, each solution caches its per-position arc costs and an arc-to-position mapping. Given a parent solution s and a neighbor s', delta evaluation proceeds as follows:

1. **Identify changed arcs.** Compare the tours of s and s' position by position to find directly changed arcs.
2. **Compute trigger fan-out.** For each changed arc that participates as a trigger in some relation (t, a, c'), mark the target arc a as affected (if it appears in the tour).
3. **Selective recomputation.** Copy the parent's cached costs and recompute only the affected positions using the backward-walk procedure.

When more than 50% of arcs are affected—either directly or through trigger fan-out—the method falls back to full evaluation, since selective recomputation offers no advantage in that regime.

Complexity. Full evaluation costs $O(n^2)$ in the worst case: each of the n arcs requires a backward walk of up to n positions, with each position checked against a precomputed hash map in $O(1)$. For delta evaluation, let c denote the number of tour positions where s and s' differ, and let $d_{\max}$ be the maximum trigger fan-out of any arc (i.e. the largest number of targets activated by a single trigger). Step 1 identifies the c changed positions in $O(n)$ time. Step 2 expands each changed position's fan-out, adding at most $c \cdot d_{\max}$ positions to the affected set; let a be the total number of affected positions. Step 3 recomputes each of the a affected positions via a backward walk costing up to $O(n)$, for a total of $O(a \cdot n)$. The overall delta evaluation cost is therefore $O(n + c \cdot d_{\max} + a \cdot n)$, with the fallback to full evaluation when $a > n/2$. On instances with moderate relation density, $a \ll n$, and delta evaluation yields a substantial speedup over full recomputation.

Handling infeasible solutions. Since the benchmark instances are not complete graphs, neighbourhood moves may produce tours that traverse arcs absent from the instance. Rather than constraining the neighbourhood operators to generate only feasible moves, we allow infeasible solutions during the search and penalise each missing arc with a cost of $5\,\bar{c}$, where $\bar{c}$ is the average base arc cost. This penalty is large enough to steer the search toward feasible solutions while preserving the simplicity and generality of the neighbourhood operators. Solution validity is verified after the search terminates.

2.2 Neighborhood Structure and Local Search

We considered three neighborhood operators, each generating candidate moves in randomized order. All operators preserve the depot at the first and last positions of the tour.

– **Or-opt** (k=1): Remove a single city from its current position and reinsert it at every other internal position.

Algorithm 1 Hill Climbing (First-Improvement)

Require: Solution s, instance I, neighborhood N
Ensure: Locally optimal solution s
  repeat
    improved $\leftarrow$ **false**
    for each neighbor s' generated by $N(s)$ in random order **do**
      if $f(s') < f(s)$ **then**
        $s \leftarrow s'$
        improved $\leftarrow$ **true**
        break {first improvement}
      end if
    end for
  until *improved* = **false**
  return s

- **Swap**: Exchange two cities at internal positions i and j.
- **Two-opt**: Reverse the sub-segment of the tour between internal positions i and j.

Each operator is implemented as a lazy generator that yields neighbors one at a time, with the cost of each neighbor computed via delta evaluation from the current solution. This avoids materializing the full neighborhood while preserving fast cost updates. A comparative analysis of the three operators (Section 3.2) shows that Or-opt consistently dominates, so all results use Or-opt as the sole neighbourhood.

The local search procedure is a first-improvement hill climbing (Algorithm 1). The algorithm scans candidates in random order and accepts the first improving move. When no improving move is found, the current solution is certified as a local optimum.

2.3 Iterated Local Search

The ILS framework [6] alternates between local search and perturbation to explore the space of local optima. Our implementation (Algorithm 2) constructs an initial tour by randomly permuting the internal nodes (with the depot fixed at positions 0 and $n+1$) and applies hill climbing to reach a first local optimum. In each subsequent iteration, the current solution is perturbed and then re-optimized by hill climbing. An SA acceptance criterion [5] governs whether the new solution replaces the current one, while the overall best solution is tracked separately.

Perturbation. A random contiguous segment of the tour is selected, with length equal to $\lfloor 0.2 \cdot n \rfloor$ internal nodes (minimum 2). One of three operators is then applied uniformly at random:

1. **Reverse**: reverse the order of nodes within the segment.

Algorithm 2 Iterated Local Search for the TATSP

Require: Instance I, parameters $maxIter$, ρ, T_0, α, $patience$
Ensure: Best solution s^*
  $s \leftarrow \text{RandomTour}(I)$
  $s \leftarrow \text{HillClimbing}(s, I)$ {Algorithm 1}
  $s^* \leftarrow s$; $T \leftarrow T_0$; $noImprove \leftarrow 0$
  for $i = 1$ **to** $maxIter$ **do**
    $s' \leftarrow \text{Perturb}(s, \rho)$ {reverse, relocate, or shuffle}
    $s' \leftarrow \text{HillClimbing}(s', I)$
    if $f(s') \leq f(s)$ **or** $\text{rand}(0,1) < \exp\big(-(f(s') - f(s))/T\big)$ **then**
      $s \leftarrow s'$
      if $f(s') < f(s^*)$ **then**
        $s^* \leftarrow s'$; $noImprove \leftarrow 0$
      else
        $noImprove \leftarrow noImprove + 1$
      end if
    else
      $noImprove \leftarrow noImprove + 1$
    end if
    $T \leftarrow \alpha \cdot T$
    if $noImprove \geq patience$ **then**
      break
    end if
  end for
  return s^*

2. **Relocate**: remove the segment and reinsert it at a random internal position.
3. **Shuffle**: randomly permute the nodes within the segment.

This perturbation is strong enough to escape basins of attraction while preserving most of the tour structure.

Acceptance criterion. A candidate solution s' always replaces the current solution s if $f(s') \leq f(s)$. Otherwise, it is accepted with probability $\exp\big(-(f(s') - f(s))/T\big)$, where T is the current temperature. The temperature follows a geometric cooling schedule $T \leftarrow \alpha \cdot T$ after each iteration, with initial temperature $T_0 = 50$ and cooling rate $\alpha = 0.99$.

Termination. The search terminates after a maximum of 5 000 iterations or after 200 consecutive iterations without improvement to the best-known solution, whichever comes first.

3 Computational Results

3.1 Instances and Experimental Setup

We evaluate our ILS on the 55 benchmark instances from the MESS 2024 competition [7], grouped into two families. The first family, **C1**, consists of 21 dense

instances (grf1–grf21): near-complete directed graphs with 20, 40, or 60 nodes (graph density 51–79%) and relation densities ranging from 2% to 65%, yielding between roughly 1 700 and 4.5 million trigger relations. Cost changes are balanced: triggers are approximately equally likely to increase or decrease an arc's cost.

The second family, **C2**, comprises 34 sparser instances (grf101–grf134) with 18 to 142 nodes. These graphs have lower density (8–29%) with more uniform relation densities (8–16%) and are organised into five sub-families of increasing size. Unlike C1, triggers here predominantly increase arc costs, penalising certain traversal orderings.

All experiments were run in parallel on a node of the Hyjnesha computing cluster at the University of Prishtina, equipped with two Intel Xeon Gold 6130 CPUs (64 logical cores total) and 128 GB of RAM. The solver is implemented in Python. We use the parameter setting described in Section 2.3: $I_{\max} = 5000$ iterations, perturbation strength 20%, initial temperature $T_0 = 50$, cooling rate $\alpha = 0.99$, and early termination after 200 consecutive non-improving iterations. Each instance was solved in 10 independent runs with different random seeds. Average running times range from under two minutes for the smallest 20-node C1 instances to several hours for the largest C2 instances with over 100 nodes.

3.2 Analysis of Neighbourhood Operators

To select the most effective neighbourhood operator, we ran the ILS with each of the three operators in isolation on a representative subset of instances from both families (5 runs per variant). Table 2 reports the results.

Table 2. Neighbourhood variant comparison: ILS with each operator in isolation (5 runs per variant). Dashes indicate no run completed within a reasonable time under the same iteration limit; superscripts denote the number of completed runs when fewer than 5. Best values per row in bold.

		Or-opt		Swap		2-opt	
Inst.	n	Best	Avg	Best	Avg	Best	Avg
grf1	20	**101.34**	102.44	104.62	106.46	107.00	112.63
grf8	40	**52.88**	53.59	56.25	61.24	58.55	72.85
grf17	60	**306.77**	312.39	323.95	328.00	345.80	352.05
grf101	18	**16.96**	16.97	17.12	17.15	17.11	17.19
grf105	26	**24.26**	24.33	24.38	24.88	24.38^{2}	25.76^{2}
grf112	50	**46.26**	46.42	–	–	–	–
grf117	52	**47.99**	48.29	48.33^{1}	48.33^{1}	–	–
grf129	72	**66.60**	67.19	–	–	–	–
grf131	100	$\mathbf{91.96}^{3}$	92.22^{3}	95.82^{1}	95.82^{1}	–	–
grf122	102	**93.80**	94.02	–	–	–	–
grf127	110	$\mathbf{100.96}^{4}$	101.27^{4}	–	–	–	–

Or-opt consistently produces the best solutions: on C1, the gap between Or-opt and 2-opt best values grows from 6% at 20 nodes to 13% at 60 nodes, while Swap is intermediate. On C2, the advantage is even more pronounced: for instances with $n \geq 50$, Swap and 2-opt rarely complete all five runs (and often complete none), whereas Or-opt scales to 110 nodes. Because trigger relations can propagate cost changes far from the modification site, the single-city reinsertion

of Or-opt explores a wider range of trigger–target reorderings per move than the segment-based operators, while keeping the per-iteration cost low. All results reported below therefore use Or-opt as the sole neighbourhood.

3.3 Comparison with State-of-the-Art Results

Table 3 reports the best and average objective values across all runs for every instance, together with the best-known solution value (BKS) from the MESS 2024 competition dashboard [7] and the percentage gap of our best solution from the BKS, defined as $\text{Gap} = 100 \cdot (f_{\text{best}} - f_{\text{BKS}})/f_{\text{BKS}}$. A negative gap indicates that our method found a new best-known solution.

Table 3. ILS results on C1 and C2 instances. Columns show the number of nodes n, the best-known solution (BKS) from the MESS 2024 competition, the best, average, and standard deviation (σ) of objective values across 10 independent runs, and the percentage gap of the best solution from the BKS.

Inst.	n	BKS	Best	Avg	σ	Gap%	Inst.	n	BKS	Best	Avg	σ	Gap%
C1 (grf1–grf21)							grf108	37	34.20	34.20	34.39	0.11	0.00
grf1	20	101.13	101.13	102.47	1.23	0.00	grf109	37	34.26	34.32	34.41	0.05	0.18
grf2	20	44.53	44.53	45.61	1.30	0.00	grf110	37	34.31	34.32	34.44	0.06	0.03
grf3	20	98.35	98.35	101.09	1.29	0.00	grf111	42	38.77	38.81	39.04	0.17	0.10
grf4	20	38.59	38.59	38.77	0.57	0.00	grf112	50	46.10	46.22	46.38	0.14	0.26
grf5	20	113.17	113.17	114.48	0.98	0.00	grf113	58	53.41	53.51	53.89	0.90	0.19
grf6	20	49.93	49.93	50.70	1.17	0.00	grf114	66	60.74	60.72	61.05	0.24	0.00
grf19	20	44.53	44.53	45.05	0.86	0.00	grf115	74	67.85	68.01	68.47	0.97	0.24
grf7	40	190.84	197.76	202.77	2.39	3.63	grf116	82	75.07	75.27	75.82	1.05	0.27
grf8	40	52.88	52.88	55.11	1.40	0.00	grf117	52	48.07	48.11	48.76	1.07	0.08
grf9	40	184.48	185.63	193.00	3.74	0.62	grf118	62	57.04	56.91	57.24	0.16	−0.23
grf10	40	55.88	56.30	59.07	1.86	0.75	grf119	72	66.23	66.16	66.83	1.03	0.00
grf11	40	204.90	210.23	217.87	2.89	2.60	grf120	82	75.38	75.31	75.98	1.12	0.00
grf12	40	62.36	64.28	70.39	4.70	3.08	grf121	92	84.27	84.42	84.85	0.27	0.18
grf20	40	52.88	52.88	55.56	1.92	0.00	grf122	102	93.37	93.61	94.22	1.08	0.26
grf13	60	282.27	292.85	298.64	3.60	3.75	grf123	62	57.04	56.89	57.67	0.42	−0.26
grf14	60	61.39	61.97	65.77	3.31	0.94	grf124	74	68.19	68.17	69.54	2.20	0.00
grf15	60	270.13	287.49	291.23	2.05	6.43	grf125	86	79.10	78.86	79.53	0.46	−0.30
grf16	60	68.83	71.94	76.48	2.74	4.52	grf126	98	90.00	90.09	90.99	1.47	0.10
grf17	60	293.36	302.97	312.24	4.50	3.28	grf127	110	100.76	100.74	102.78	3.81	−0.02
grf18	60	78.65	84.89	91.17	3.65	7.93	grf128	122	111.46	111.71	113.27	2.01	0.22
grf21	60	61.47	61.53	68.61	5.04	0.10	grf129	72	66.32	66.33	66.88	0.26	0.02
C1 avg. gap						1.79	grf130	86	79.01	79.52	82.74	5.09	0.65
C2 (grf101–grf134)							grf131	100	91.91	91.84	93.35	2.64	0.00
grf101	18	16.96	16.96	16.98	0.03	0.00	grf132	114	104.49	104.56	105.64	1.63	0.07
grf102	22	20.49	20.49	20.57	0.06	0.00	grf133	128	117.25	117.59	121.11	3.94	0.29
grf103	22	20.54	20.54	20.54	0.00	0.00	grf134	142	129.78	130.32	134.03	4.50	0.42
grf104	27	25.12	25.12	25.20	0.06	0.00							
grf105	26	24.26	24.26	24.33	0.06	0.00							
grf106	32	29.57	29.57	29.72	0.06	0.00							
grf107	37	34.26	34.22	34.34	0.05	−0.12							
C2 avg. gap						0.07	New BKS found						5/34

C1 instances. On the dense C1 instances, the ILS matches the best-known solution on all seven 20-node instances and two of the seven 40-node instances.

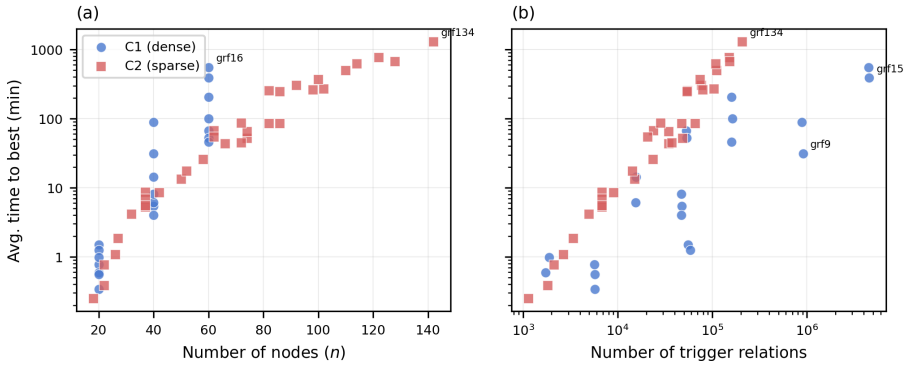

Fig. 2. Average time to best solution vs. (a) number of nodes and (b) number of trigger relations, coloured by instance family (log scale on all axes except (a) x-axis).

The average gap from the BKS across all 21 C1 instances is 1.79%. Performance degrades on larger and denser instances: the 60-node group has an average gap of roughly 3.9%, driven primarily by instances with high relation density (grf15, grf18) where the rugged search landscape with millions of trigger relations creates many local optima. Run-to-run variance is also noticeable on these instances, with the gap between the best and worst solution in 10 runs reaching up to 16% on the hardest 60-node instances.

C2 instances. On the sparser C2 family the ILS produces remarkably strong results, achieving an average gap of just 0.07% and establishing **5 new best-known solutions** out of 34 instances (indicated by negative gaps in Table 3). The gap between the best and average solution is typically below 1% for instances up to about 80 nodes and stays within 3% even at $n = 142$. Objective values scale nearly linearly with the number of nodes—roughly 0.92–0.95 per node—reflecting the regular structure of these sparser graphs. The new BKS are spread across all sub-families, including instances with up to 110 nodes (grf127), demonstrating that the ILS effectively exploits the sparser trigger structure of these instances.

Scalability. Figure 2 shows the average time to reach the best solution found in each run. Panel (a) reveals that, for a given node count, C1 instances require considerably more time than C2 instances due to their higher graph and relation density. Panel (b) shows that the number of trigger relations is a strong predictor of solving time across both families: instances with similar relation counts require comparable effort regardless of family membership. Average times range from under two minutes for the smallest instances to several hours for the largest C2 instances (grf134, 142 nodes) and the most relation-dense C1 instances (grf15, grf16, over four million relations).

Comparison with the GRASP of Salvà Soler and de Lambertye. The concurrent work by Salva Soler and de Lambertye [8] proposes a GRASP with a MIP-based construction heuristic and multi-neighbourhood local search, implemented in C++ with Gurobi. On C1, they report an average gap of 0.77% within a 60-second time limit, compared to our 1.79%. On C2, they report an average gap of 0.40%, whereas our ILS achieves 0.07% and finds 5 new best-known solutions. The gap on C1 is expected: their MIP-based construction provides high-quality starting solutions for the dense instances, and the C++ implementation with Gurobi is substantially faster than our Python solver. However, our ILS shows a clear advantage on C2, where the sparser trigger structure is better suited to the delta evaluation and iterative perturbation strategy.

Comparison with exact methods. A direct cost comparison with the ILP formulation of Cerrone et al. [4] is not possible, as their experiments use a different set of instances. However, their results demonstrate that the exact approach already encounters MIP gaps of 17–42% on complete graphs with only 25 nodes and 10 000 trigger relations, and hits the time limit of 1 000 seconds at 20 nodes with 6 400 relations. In contrast, our ILS handles instances with up to 142 nodes and millions of trigger relations. On the 20-node C1 instances, average running times are under two minutes; 40-node instances complete in minutes to about two hours; and even the largest C2 instances (110–142 nodes) complete within a day on a single core. This confirms that metaheuristic approaches are essential for tackling practically-sized TATSP instances.

4 Conclusion

We presented an Iterated Local Search for the Trigger Arc Travelling Salesman Problem [4], a TSP variant in which traversing certain arcs dynamically alters the costs of other arcs. Our method combines a simulated annealing acceptance criterion with a first-improvement hill climbing phase using an Or-opt neighbourhood—selected after preliminary experiments showed it consistently outperformed configurations including swap and 2-opt—augmented with a delta cost evaluation scheme that exploits trigger-relation caching. Computational experiments on the 55 benchmark instances from the MESS 2024 competition [7] demonstrate that the approach achieves an average gap of 1.79% on the dense C1 instances and just 0.07% on the sparser C2 family, establishing 5 new best-known solutions on the latter. The ILS scales to instances with up to 142 nodes, well beyond the practical reach of existing exact methods, which are limited to roughly 25 nodes. Compared with the concurrent GRASP approach of Salva Soler and de Lambertye [8], our method is particularly effective on C2, while the GRASP shows stronger performance on the dense C1 instances thanks to its MIP-based construction heuristic.

Several directions for future work appear promising. Replacing the random initial solution with a construction heuristic—such as a nearest-neighbour or a trigger-aware greedy procedure—could provide higher-quality starting points

and reduce the number of ILS iterations needed to reach good solutions. An adaptive perturbation mechanism that learns which perturbation strategy is most effective for a given instance structure would further improve robustness. Parallel multi-start search could exploit modern multi-core architectures to explore the solution space more broadly. Finally, hybrid matheuristic approaches that solve subproblems exactly via integer linear programming while using metaheuristic search at the global level offer a compelling avenue, as does evaluation on larger instances and real-world warehouse routing data.

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2011). https://doi.org/10.1515/9781400841103
2. Bossek, J., Casel, K., Kerschke, P., Neumann, F.: The node weight dependent traveling salesperson problem: Approximation algorithms and randomized search heuristics. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference. pp. 1286–1294. ACM (2020). https://doi.org/10.1145/3377930.3390243
3. Carmesin, S., Woller, D., Parker, D., Kulich, M., Mansouri, M.: The Hamiltonian cycle and travelling salesperson problems with traversal-dependent edge deletion. Journal of Computational Science **74**, 102156 (2023). https://doi.org/10.1016/j.jocs.2023.102156
4. Cerrone, C., Truvolo, M., Dragone, R., Battaglia, R.: The trigger arc TSP: Optimise the picking process in warehouses with compactable storage systems. In: Data Science and Intelligent Systems. Lecture Notes in Computer Science, vol. 14778, pp. 279–289. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-78238-1_26
5. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983). https://doi.org/10.1126/science.220.4598.671
6. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: Framework and applications. In: Handbook of Metaheuristics, pp. 129–168. Springer, 3rd edn. (2019). https://doi.org/10.1007/978-3-319-91086-4_5
7. Metaheuristics Summer School: MESS 2024 Competition Dashboard. https://fourclicks.eu/fck/mess2024/frontend/#/home/dashboard (2024), accessed: 2026-03-14
8. Salvà Soler, J., de Lambertye, G.: A fast GRASP metaheuristic for the trigger arc TSP with MIP-based construction and multi-neighborhood local search. arXiv preprint arXiv:2508.08477 (2025). https://doi.org/10.48550/arXiv.2508.08477