Embedded 2-opt for LNS: A Process-Aware Window Mechanism Approach

Donghan Wu
School of Mathematics and Statistics
Xi'an Jiaotong University
Xi'an China
yumeko@stu.xjtu.edu.cn

Jialong Shi
School of Mathematics and Statistics
Xi'an Jiaotong University
Xi'an, China
jialong.shi@xjtu.edu.cn

***Abstract*—Large Neighborhood Search (LNS) and 2-opt local search are key algorithms for routing problems like TSP and CVRP. Standard hybrid frameworks usually separate these steps, using LNS for global exploration and 2-opt for later optimization. This order often leads to fault propagation, where suboptimal edges from repair are only fixed after costly global scans. To address this limitation, we investigate a *Process-Aware Window Mechanism* for 2-opt based LNS. Unlike other granular search methods that use fixed geometric metrics, our method uses the dynamic information from the repair operator. We set an optimization 'window' around the insertion index of new nodes. This change makes 2-opt a proactive, built-in correction tool instead of just a post-processing step. We test six window selection strategies, from static to adaptive. Our experimental analysis shows that this frequent, low-complexity micro-optimization can improve convergence speed and result stability compared to traditional global search methods.**

***Index Terms*—Large Neighborhood Search, 2-opt, Local Search, Candidate Lists, Traveling Salesman Problem, Capacitated Vehicle Routing Problem.**

I. Introduction

Routing problems such as the Traveling Salesman Problem (TSP) and the Capacitated Vehicle Routing Problem (CVRP) [4] are classic challenges in combinatorial optimization. Shaw [1] introduced Large Neighborhood Search (LNS) to address these NP-hard problems, while Croes [3] developed the 2-opt operator to improve local structures. Combining local search with LNS, usually as a step after repair, has become a common hybrid approach, used from early adaptive LNS methods [2] and recent hybrid algorithms [7]–[9].

In most cases, these two steps are kept separate, with local search performed only after the repair is finished. Because a global 2-opt scan has $O(n^2)$ complexity, it cannot be run often, which makes it hard to achieve good anytime performance. As a result, finding more efficient ways for the repair operator and local search to work together is very useful in practice.

This paper looks at new, efficient ways to handle the local search step in the LNS framework. We focus on a "Process-Aware Window Mechanism." By using the real-time "insertion position index" from the LNS repair operator, we set a dynamic optimization window that restricts the search scope to a localized segment. This lets us build the local search directly into the construction process without changing the main logic of the solver.

To explore how well this mechanism works, we design six different window selection strategies, from simple static windows to adaptive ones based on search history. We combine these strategies to create several algorithm variants and test them on TSP and CVRP benchmarks. Our results show that this frequent, small-scale correction method is better than the traditional, less frequent global search in both convergence speed and solution stability.

The rest of the paper is organized as follows: Section II reviews LNS and related work. Section III explains the window mechanism and the six selection strategies. Section IV the experiments and insights. Section V concludes the paper.

Corresponding author: Jialong Shi (jialong.shi@xjtu.edu.cn).This work was supported by National Natural Science Foundation of China(grants 12571590).

II. Routing Problems and Algorithms

A. Problem Definitions: TSP and CVRP

We look at two main routing models. The Traveling Salesman Problem (TSP) finds the shortest possible route that visits a set of cities exactly once. The CVRP [4] has constraints that several vehicles start from a central depot, and each has a weight limit Q to meet the customers' demands q_i. Since the number of possible routes grows quickly, exact mathematical models cannot handle large cases, so heuristic algorithms are often used instead.

B. Search Algorithms: LNS and 2-opt

Large Neighborhood Search (LNS) is a widely used meta-heuristic. It removes parts of a solution and then inserts them back into the route. This process helps explore new possible solutions. Edges in these new routes, solvers use 2-opt. This method swaps pairs of edges to uncross paths and accepts the change if it makes the route shorter. But trying all swaps takes as much as $O(n^2)$ time for each route. This high cost does not work well with LNS, especially when running thousands of destroy-repair cycles. Running a full 2-opt scan after every insertion is not practical. For this reason, we need to limit the search, and our window mechanism helps with this.

III. Window Mechanism

A. Core Procedure

This method uses a window mechanism to combine Large Neighborhood Search (LNS) with 2-opt local search. After the repair operator inserts nodes, local optimization is done at the first insertion position of each changed route. The algorithm records the first insertion position pos_i for each route during node insertion, creating the set $\{(path_i, pos_i)\}$. After the repair phase, a local window $[pos_i - w_i, pos_i + w_i]$ is set for each route in this set, as illustrated in Fig. 1. Then, 2-opt scanning is performed within this w_i-edge neighborhood, using the first improving exchange found. The window size is w_i. The w_i-edge neighborhood around the first insertion is treated as a promising candidate list. This approach significantly reduces the computational overhead of each local search and prevents repeated optimizations on the same route.

Algorithm 3 shows the main steps of the window mechanism:

Algorithm 1 Process-Aware Window Mechanism Procedure

Require: Initial solution s_0, destroy ops $\mathcal{D}$, repair ops $\mathcal{R}$, maxEval
Ensure: Best found solution s^*

$s \leftarrow s_0$, $s^* \leftarrow s_0$, evalCount $\leftarrow 0$
while evalCount $<$ maxEval **do**
  Select $d \in \mathcal{D}$ and $r \in \mathcal{R}$ randomly
  $s_{\text{partial}} \leftarrow d(s)$
  $\{s', \mathcal{I}\} \leftarrow r(s_{\text{partial}})$ $\{\mathcal{I} = \{(path_i, pos_i)\}\}$
  for all $(path_i, pos_i) \in \mathcal{I}$ **do**
    $w_i \leftarrow$ ComputeWindowSize()
    $s' \leftarrow$ Window2opt$(s', path_i, pos_i, w_i)$
    evalCount $\leftarrow$ evalCount + Evaluations(w_i)
  end for
  if Accept(s', s) **then**
    $s \leftarrow s'$
  end if
  if $f(s') < f(s^*)$ **then**
    $s^* \leftarrow s'$
  end if
end while
return s^*

TSP vs. CVRP: TSP is a single-route unconstrained problem requiring only one window w per iteration. CVRP involves multiple routes, so we determine a total window budget W per iteration and allocate it among repaired routes proportionally to their lengths: $w_i = \left\lfloor W \cdot \frac{|path_i|}{\sum_{j \in \mathcal{I}} |path_j|} \right\rfloor + 2$, where $|path_i|$ denotes the number of nodes in route i and w_{base} secures a minimum search intensity.

B. Window Selection Strategies

Window size directly trades off search breadth against computational cost: larger windows capture more improvements but increase evaluations quadratically. We systematically designed six window selection strategies forming a spectrum from static simplicity to dynamic intelligence, optimizing this trade-off across several scenarios.

- **Global 2-opt Baseline (Baseline):** This strategy serves as a performance upper bound, executing global 2-opt scanning after each iteration without any window mechanism. Its purpose is to quantify the speedup and quality boundaries of the window approach.
- **Fixed Small Window (FixedWin):** Uses a constant window size $w = w_{\text{fixed}}$. This simplest strategy offers predictable computational cost but lacks adaptability to search state.
- **Random Window Sampling (RandSamp):** Samples window size independently each iteration from a uniform distribution $w \in [w_{\min}, w_{\max}]$. This introduces randomness to enhance exploration diversity and prevent deterministic patterns from getting trapped.
- **Iteration-Proportional Window (IterProp):**The window increases in a predictable way as evaluation progresses: $w_t = w_{\min} + \left\lfloor \frac{\text{evalCount}}{\text{maxEval}} \cdot (w_{\max} - w_{\min}) \right\rfloor$. Using small windows at the start helps control costs, and larger windows later allow for more precise improvements. This method does not react to the current search state in real time, but it ensures that results are always reproducible and predictable. Using small windows at the start helps control costs, and larger windows later allow for more precise improvements. This method does not react to the current search state in real time, but it ensures that results are always reproducible and predictable.
- **Stall-Driven Linear Window (StallLin):** The window size is kept at $w = w_{\text{base}}$ to minimize overhead while stallCount $\leq \tau$. Once the stall count exceeds τ, the window expands linearly towards $w_{\max}$. This approach balances efficiency with robustness: it preserves a low computational cost during effective search phases but widens the scope to escape local optima when stagnation is detected.
- **Adaptive History Window (AdaptHist):** To maintain search diversity, stochasticity is introduced into the window sizing. The algorithm samples w_{current} from a restricted range near $w_{\min}$ for local intensification, but shifts to a broader range near $w_{\max}$ upon detecting stagnation. This integration of probabilistic perturbation with stagnation feedback prevents the entrapment in local optima often observed with static window strategies.
- **Stall Escape (StallEsc):** Implementing a binary switching mechanism, this strategy defaults to a minimal window w_{base} to maintain low computational overhead. A global 2-opt search is triggered only when stagnation is detected (stallCount $> \tau$). This approach effectively concentrates computational resources, offering rapid local refinement during normal progression while reserving exhaustive search capabilities for escaping local optima.

IV. Experimental Evaluation

In this section, we share the results of testing our experimental analysis. Our experiments had three main goals: to

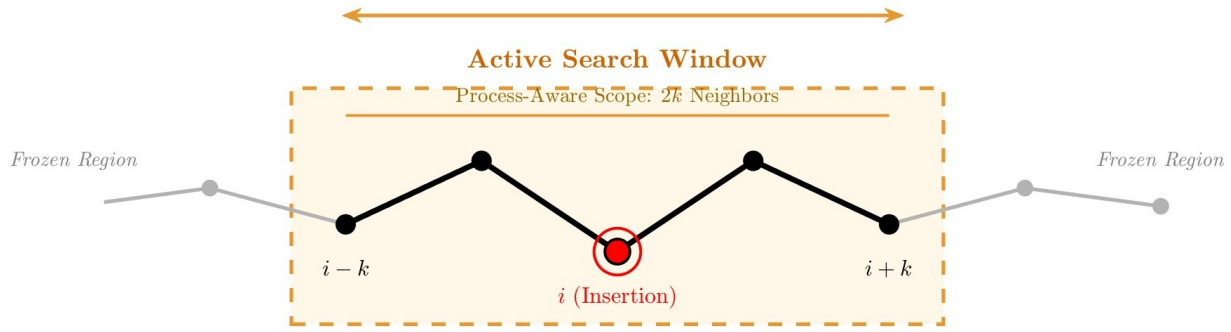


Fig. 1. Schematic of the proposed Process-Aware Window Mechanism. Upon the insertion of a node (red) at index i, the algorithm dynamically defines an active search window covering the range $[i-k, i+k]$ (highlighted in orange). The 2-opt operator is strictly confined to this localized segment, while the distal parts of the route (gray) remain frozen. This ensures that computational resources are allocated exclusively to the most volatile region of the solution.

check the locality hypothesis, to study how the budget is allocated, and to compare our method's performance with existing approaches.

We implemented all algorithms in MATLAB and ran them on an Intel Core Ultra 9 285H workstation with 32GB of RAM. We used standard benchmark problems from TSPLIB [5] and CVRPLIB [6], with sizes from $N = 48$ to 299.

To ensure a fair comparison between lightweight local moves and computationally intensive global searches, we adopted a computational evaluation budget ($MaxEval = \eta \cdot n^2$) as the stopping criterion, rather than a fixed iteration count.

A. Motivation Validation: Distribution of Defects

We began by testing whether topological defects caused by repair operators are limited to specific areas. To do this, we used a controlled Insert-Probe-Clean protocol on the `eil51` instance. Before each node was inserted, we made sure the solution reached a local optimum to remove any effects from previous steps. Right after inserting at index I, we checked for possible improvements (hit rate) in two areas: the Insertion Segment ($[I-k, I+k]$) and a separate, randomly chosen Distal Random Segment.

The statistical results shown in Fig. 2 support the locality hypothesis. The targeted probe in the insertion segment had a hit rate of 5.10%, while the random control group had a rate of 1.88%. The Wilcoxon Signed-Rank Test shows this difference is statistically significant ($p = 1.02 \times 10^{-18}$). The spatial distribution of valid swaps (Fig. 2b) also shows that more than 70% of optimization potential is within a topological distance of 3 from the insertion point.These results justify limiting the local search scope to a highly localized window.

B. Behavioral Analysis: Budget Allocation

The efficiency of the mechanism stems from a fundamental shift in computational budget allocation. We analyzed the execution frequency of the 2-opt operator under a fixed budget on the `KroA200` instance.

As shown in Fig. 3, the Global Baseline strategy averaged only 45.9 executions because the $O(n^2)$ cost of full scans is too high. In contrast, window-based strategies like *FixedWin* and *IterProp* used the same budget to perform over 570 micro-optimizations. This much higher frequency lets the algorithm fix structural defects as they arise during construction, which helps prevent suboptimal edges from spreading—a common problem in low-frequency global search methods.

C. Main Results on Benchmarks

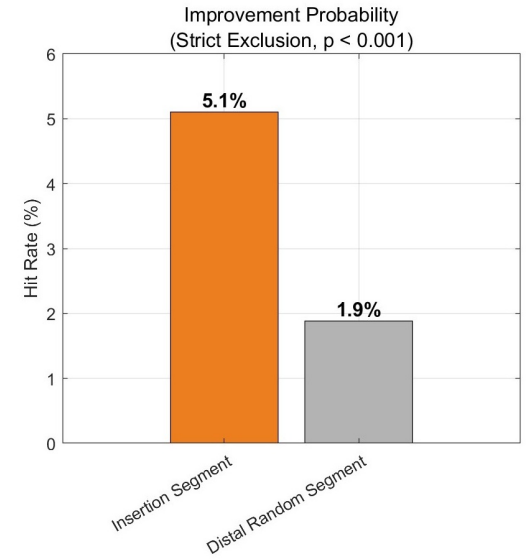
We evaluated seven algorithm variants over various TSP and CVRP instances. Performance is assessed using three metrics:

1) **Gap (%)**: Percentage deviation from the known optimal solution.
2) **AUC**: Area Under the Convergence Curve, representing the efficiency of budget utilization.
3) **Stability**: Standard deviation of the Gap across 30 independent runs.

To take into account varying objective scales, an instance-wise rank aggregation method is employed to determine the overall standing of each strategy.

1) Results on TSP Instances: Table I shows the optimality gaps for all seven algorithm variants across ten benchmark instances. To compare performance at different problem sizes, we calculated the Mean Rank for each strategy based on solution quality. The experimental data leads to the following conclusions.

Hierarchical Performance Analysis. The hierarchy in instance-wise rank aggregation demonstrates a clearly superior performance. Our **AdaptHist** method achieves the overall best performance at Mean Rank **2.50** with a movement close to **RandSamp** (Rank 2.90). Surprisingly, the **Baseline** which uses low-frequency global optimization performs poorly at Mean Rank 5.00. The difference in ranking demonstrates that, on its own, low-frequency global optimization fails to lead to consistently high-quality convergence. The higher ranking of AdaptHist demonstrates that steered by historical feedback, the size of the optimization window that gives the best trade-off between local intensification and exploration.



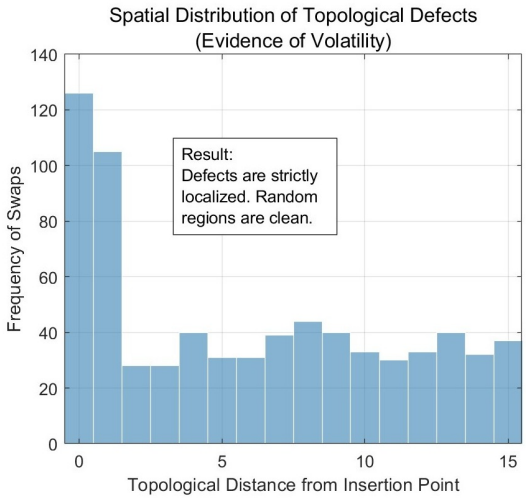


Fig. 2. Validation of the Locality Principle. (a) Improvement probability in the insertion neighborhood (5.10%) is significantly higher than in distal segments (1.88%, $p < 10^{-18}$). (b) Spatial distribution of improvements shows that most potential gains are near the insertion index.

TABLE I
GAPS (%) ON TSP BENCHMARKS (30 RUNS)

Instance	N	**Base.**	**Fix.**	**StallL.**	**Rand.**	**Adapt.**	**Iter.**	**StallE.**
eil51	51	**1.19**	2.14	1.91	1.96	2.01	2.12	0.97
berlin52	52	**0.15**	3.17	1.76	2.55	1.55	2.91	1.35
kroB100	100	0.95	0.41	0.44	0.38	**0.38**	0.70	0.81
kroC100	100	1.21	1.35	0.97	0.98	**0.89**	1.41	1.17
bier127	127	**1.55**	3.37	2.68	3.16	2.65	3.41	1.71
kroA150	150	1.72	1.07	1.00	0.94	**0.94**	1.13	1.27
kroB150	150	1.66	2.30	0.79	0.91	**0.81**	1.32	1.55
kroA200	200	2.55	2.86	1.83	1.79	**1.82**	1.73	2.40
kroB200	200	2.71	1.35	1.30	1.20	**1.20**	1.31	1.77
pr299	299	3.97	2.31	2.30	2.32	**2.10**	2.25	3.00
Mean Rank	-	5.00	5.50	3.00	2.90	**2.50**	4.70	4.40

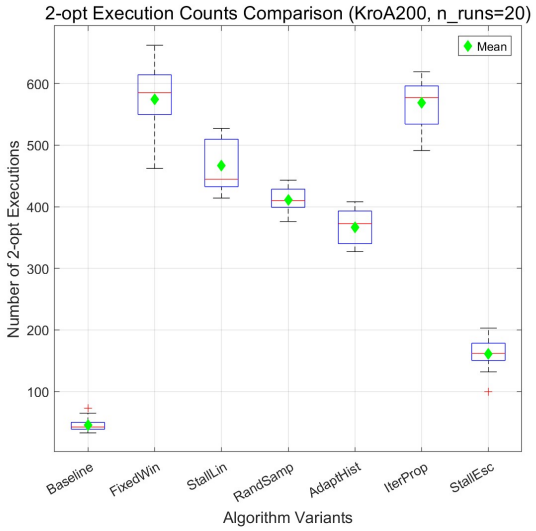


Fig. 3. 2-opt execution counts under fixed budget ($MaxEval \approx 8.4 \times 10^7$). Window strategies (FixedWin, IterProp) allow > 500 executions, while the Global Baseline is limited to < 50 due to high cost.

Scale Effect and Scalability. A close look at Table I shows how problem size affects results. For small cases ($N \leq 52$), the Baseline and StallEsc (Hybrid) strategies work best because $O(n^2)$ scans are affordable and allow thorough defect removal. But when $N \geq 150$, things change. In the largest case, `pr299`, the Baseline's Gap rises to 3.97%, while AdaptHist keeps the Gap at 2.10%. This shows that using a physical window restriction is not just a way to speed things up—it is essential for handling larger search spaces.

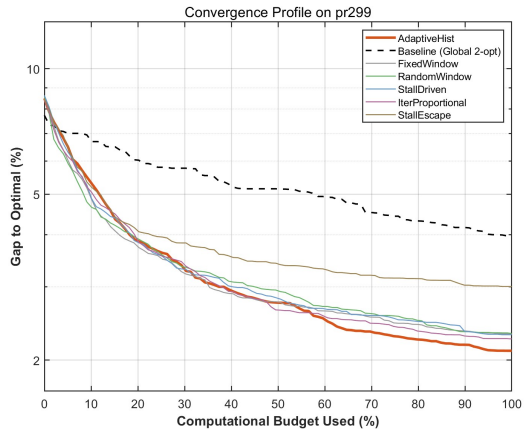


Fig. 4. Convergence on `pr299`. AdaptHist (red) improves rapidly, surpassing the final quality of the Global Baseline (black) using less than 20% of the budget.

Convergence Efficiency. Fig. 4 shows how the algorithm converges on the `pr299` instance. The window-based strategies drop quickly in the early stages of the search. The AdaptHist strategy, in particular, reaches a solution within 1% of its final quality using less than 20% of the computational budget needed by the Baseline. This speed comes from the

windowed 2-opt, which makes thousands of small adjustments while the Baseline completes just one global scan.

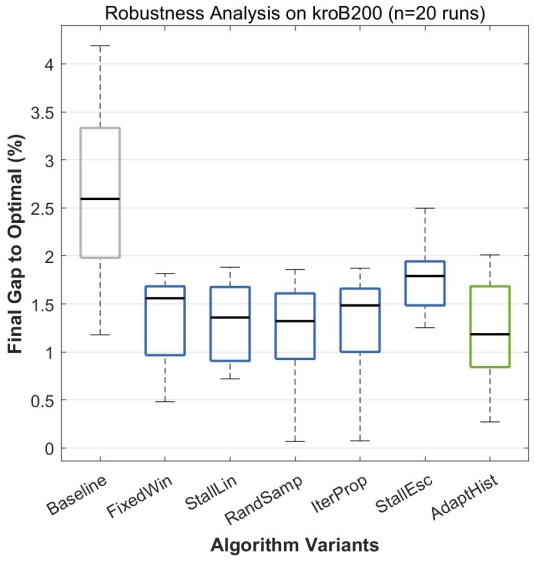


Fig. 5. Robustness on `kroA200` ($n = 20$). Window strategies show tighter distributions (lower variance) compared to the Baseline.

Robustness. Stability is important for practical applications. The boxplot analysis (Fig. 5) shows that the Baseline strategy has a high standard deviation (0.66 on `kroB200`), so it is sensitive to the initial random seeds. In contrast, strategies with stricter window constraints, like *IterProp* and *FixedWin*, have much tighter distributions (Std. Dev. ≈ 0.34). This means the embedded repair mechanism helps keep solution quality more consistent and lowers the variance often seen in randomized metaheuristics.

Anomaly. The `bier127` instance is an exception where the Baseline performs better than all window variants. This unusual result probably comes from long-range dependencies in the optimal topology that go beyond what the window operators can handle. For some difficult topologies, using a hybrid method such as StallEsc or adding regular global restarts may help support the window mechanism.

2) Results on CVRP Instances: The CVRP differs from the TSP by adding strict capacity limits, which are managed using a penalty function. This changes the solution space, making it more complex with more noise and local high-cost areas. Table II shows performance metrics for the benchmark set and highlights how these changes impact algorithm performance.

Stochasticity Outperforms Adaptation.

Table II shows that strategy effectiveness is reversed compared to the TSP phase. The *AdaptHist* strategy, which worked well without constraints, now has a mean Gap rank of 4.17. In contrast, the simpler *RandSamp* and *FixedWin* strategies perform better, with mean ranks of 2.33 and 4.33.

These results suggest that in penalty-based problems, historical feedback often gives mixed signals, like conflicting information about distance and penalty reduction. Adaptive methods can overfit to this noise and get stuck in local optima.

TABLE II
GAPS (%) ON CVRP BENCHMARKS (30 RUNS)

Instance	Base.	StallL.	Rand.	Adapt.	Iter.	Fix.	StallE.
A-n60-k9	2.32	1.78	2.01	1.92	**1.47**	1.71	1.67
A-n69-k9	1.55	1.58	**1.46**	1.64	1.54	1.65	1.58
A-n80-k10	1.60	1.35	**1.13**	1.44	1.89	1.78	1.61
M-n101-k10	4.27	2.87	2.75	**1.91**	3.00	2.21	2.83
M-n121-k7	7.53	7.21	7.01	7.22	7.46	**6.89**	7.14
M-n200-k16	15.93	14.62	**14.28**	16.36	15.05	18.08	16.64
Mean Rank	5.33	3.50	**2.33**	4.17	4.17	4.33	4.17

On the other hand, stochastic or fixed window strategies, which do not depend on past gradients, are more robust and help navigate complex penalty surfaces.

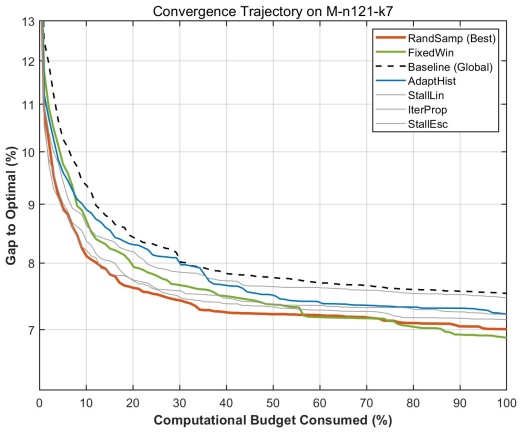


Fig. 6. Convergence on `M-n121-k7`. The Global Baseline (black) stagnates. *FixedWin* (green) and *RandSamp* (red) maintain descent and outperform *AdaptHist* (blue).

Stagnation of Global Search

For the medium-scale instance `M-n121-k7` (Fig. 6), the Global **Baseline** performs poorly with a rank of 5.33. Its curve levels off early at a Gap of about 7.5%, showing it struggles to escape penalty basins. This happens because the low-frequency global search cannot fix high-penalty routes well. In contrast, the *FixedWin* and *RandSamp* strategies reach final gaps of 6.89% and 7.01%, showing that frequent corrections are important when constraints are strict.

Adaptation Failure at Scale

The results for the largest instance, `M-n200-k16` (Table II), show the biggest limitations of adaptation. In this complex, constrained space, the *AdaptHist* strategy has a Gap of 16.36%, which is worse than the unoptimized *Baseline* at 15.93%. This shows that adjusting parameters too much can cause overfitting when gradient signals are noisy. Meanwhile, *RandSamp* is the best performer at 14.28%, showing that stochastic micro-optimization is the most reliable way to handle complexity.

3) Discussion: Landscape Structure and Strategic Adaptability: Data from both the TSP and CVRP problems show that different algorithms perform better depending on the

problem, as shown in Fig.7. Comparing these problems shows how the solution landscape influences which search strategy is most effective.

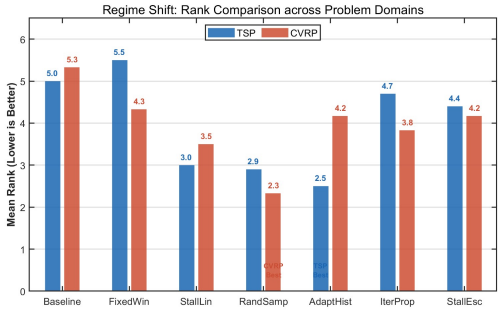


Fig. 7. Mean rank comparison across problem domains. *AdaptHist* (Blue) and *RandSamp* (Red) show distinct advantages in TSP and CVRP respectively, while the Global Baseline (Gray) consistently underperforms.

Strategy Inversion: Learning vs. Randomness. Fig. 7 shows that the most effective strategy depends on the problem type. For the unconstrained TSP, which has a smooth landscape, the **AdaptHist** strategy (blue bar) works best. This suggests that when gradient signals are reliable, using feedback from past steps helps the algorithm find solutions more quickly. In contrast, for the CVRP, which has a rugged landscape, the **RandSamp** strategy (red bar) performs better. This means the penalty function adds a lot of noise to the feedback. In these situations, adaptive methods can overfit to misleading signals, while random strategies help avoid getting stuck in local optima caused by soft constraints.

The Universal Truth: "Frequency $>$ Scope". No matter which window strategy is used, the Global **Baseline** (gray bar) consistently performs poorly in both problems. This result supports the main idea of the study: in LNS, the frequency of error corrections is more important than their scope. The Baseline method does not work well because its costly $O(n^2)$ scans are too infrequent to fix errors from the repair operator. In contrast, the embedded window framework is effective, whether the window is adaptive or random, because it uses the computational budget for frequent low-cost micro-corrections that catch problems early.

Design Implications. These results provide a practical guideline for designing solvers:

- For problems with smooth objectives (e.g., TSP), prioritize **adaptive strategies** to maximize convergence speed.
- For problems with complex constraints and penalty functions (e.g., CVRP), prefer lightweight **stochastic strategies** (Random or Fixed) to maximize robustness and prevent overfitting.

V. Conclusion

This study presents an experimental analysis of a process-aware embedded mechanism for LNS. By shifting from reactive global post-processing to preventive micro-correction, we utilize the spatial locality of topological defects. The proposed window-based mechanism improves efficiency and solution quality.

Our analysis shows that optimal strategy depends on the problem structure. For TSP, adaptive strategies work best; for CVRP, stochastic strategies are more robust against penalty noise. Ultimately, in constrained routing, the *frequency* of correction is often more important than the *scope*. This method addresses the delayed response of traditional frameworks and provides a scalable foundation for future solvers.

Acknowledgment

This work was supported by National Natural Science Foundation of China(grants 12571590).

References

[1] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming (CP98)*, M. Maher and J.-F. Puget, Eds. Berlin, Heidelberg: Springer, 1998, pp. 417–431.

[2] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation Science*, vol. 40, no. 4, pp. 455–472, 2006.

[3] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

[4] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[5] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[6] E. Uchoa, D. Pecin, A. Pessoa, M. Poggi, T. Vidal, and S. Subramanian, "New benchmark instances for the Capacitated Vehicle Routing Problem," *European Journal of Operational Research*, vol. 257, no. 3, pp. 845–858, 2017.

[7] A. Saker, A. Eltawil, and I. Ali, "Adaptive large neighborhood search metaheuristic for the capacitated vehicle routing problem with parcel lockers," *Logistics*, vol. 7, no. 4, p. 72, 2023.

[8] Y. Li, Z. Yang, S. Zhang, and W. Liu, "A study of the capacitated vehicle routing problem with time-window and three-dimensional loading constraints in land–sea transport," *Sustainability*, vol. 16, no. 23, p. 10272, 2024.

[9] H. Ma, T. Yang, Z. Wang, J. He, and X. Ren, "Improved adaptive large neighborhood search combined with simulated annealing (IALNS-SA) algorithm for vehicle routing problem with simultaneous delivery and pickup and time windows," *Electronics*, vol. 14, no. 12, p. 2375, 2025.

[10] P. Toth and D. Vigo, "The granular tabu search and its application to the vehicle-routing problem," *Informs Journal on Computing*, vol. 15, no. 4, pp. 333–346, 2003.

[11] R. F. Fachini, V. A. Armentano, and F. M. B. Toledo, "A granular local search matheuristic for a heterogeneous fleet vehicle routing problem with stochastic travel times," *Networks and Spatial Economics*, vol. 22, no. 1, pp. 33–64, 2022.

[12] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A hybrid genetic algorithm for multidepot and periodic vehicle routing problems," *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012.

[13] A. Subramanian, E. Uchoa, and L. S. Ochi, "A hybrid algorithm for the vehicle routing problem with simultaneous pickup and delivery," *Computers & Operations Research*, vol. 40, no. 11, pp. 2896–2910, 2013.

[14] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[15] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, pp. 106–130, 2000.

[16] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[17] A. Hottung and K. Tierney, "Neural large neighborhood search for the capacitated vehicle routing problem," in *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*, 2020, pp. 2632–2639.