I.J. Intelligent Systems and Applications, **2025, 4, 58-68**
Published Online on August 8, 2025 by MECS Press (http://www.mecs-press.org/)
DOI: 10.5815/ijisa.2025.04.06

Exploratory Heuristics for Scalable Combinatorial Optimization

Makhovoi Oleksandr*
Department of Computer Systems Software, National Technical University of Ukraine "Igor Sikorsky Kyiv polytechnic institute", 37 Beresteyskyi Avenue, 03056 Kyiv, Ukraine
E-mail: mahavoy2002@gmail.com
ORCID iD: https://orcid.org/0009-0007-1610-4022
*Corresponding author

Vasyl Yurchyshyn
Department of Computer Systems Software, National Technical University of Ukraine "Igor Sikorsky Kyiv polytechnic institute", 37 Beresteyskyi Avenue, 03056 Kyiv, Ukraine
E-mail: vasil.yurchishin@gmail.com
ORCID iD: https://orcid.org/0000-0001-7550-9213

Received: 25 January 2025; Revised: 13 March 2025; Accepted: 09 May 2025; Published: 08 August 2025

Abstract: This paper introduces a deterministic insertion-based heuristic named the Localized Selective Insertion Heuristic, which incorporates adaptive mechanisms such as dynamic adjustment of evaluated neighbors and systematic seed route initialization, contributing to the heuristic's novelty and robust performance, designed to provide a reliable balance between the quality of solutions and computational efficiency. The proposed heuristic builds a complete solution incrementally, systematically inserting each unvisited node into an evolving tour by evaluating a limited number of potential insertion points based on their spatial proximity to already visited locations. This localized and selective evaluation strategy substantially reduces computational effort, typically allowing large problem instances to be solved in under 150 milliseconds, with achieved solution quality consistently within 2–14% of known optimal values. To clearly illustrate the effectiveness of this trade-off, we propose a Normalized Performance Index, integrating both solution accuracy and computational speed into a unified metric. The Localized Selective Insertion Heuristic demonstrated superior performance according to this index, achieving the best score in 16 out of 17 tested benchmark scenarios. The simplicity, deterministic nature, minimal parameter sensitivity, and ease of practical implementation make the proposed approach particularly suitable for applications requiring scalability, consistent performance, and straightforward reproducibility, such as logistics, transportation planning, and industrial automation.

Index Terms: Traveling Salesman Problem, Heuristic Algorithm, Local Optimization, Combinatorial Optimization, Performance Comparison, Deterministic Algorithm, Algorithmic Trade-off.

1. Introduction

Combinatorial optimization represents a fundamental challenge in numerous computational fields, as the number of possible solutions typically grows exponentially with problem size, making exhaustive search impractical or impossible even for moderately large scenarios [1, 2]. Practical applications spanning vehicle routing, integrated circuit layout, production task scheduling, warehouse management, telecommunications network design, and clustering for data analysis commonly employ approximate methods due to the computational impracticality of exact solutions.

The Traveling Salesman Problem, a widely recognized representative of combinatorial optimization, epitomizes this complexity. Specifically, the problem involves determining the shortest possible route that visits every city exactly once before returning to the starting point. Despite the apparent simplicity and clear geometric nature of this formulation, its complexity escalates dramatically with the number of cities involved, categorizing it as NP-hard — meaning exact algorithms, such as branch-and-bound or dynamic programming, quickly become computationally prohibitive even for relatively small numbers of cities [3].

To effectively manage larger and more complex cases of the Traveling Salesman Problem, the scientific community predominantly employs heuristic and metaheuristic methods [4, 5, 6]. These strategies deliberately sacrifice strict optimality to significantly improve computational speed and scalability, thus delivering solutions of acceptable quality

This work is open access and licensed under the Creative Commons CC BY 4.0 License.

within feasible time frames. Examples of such methods include genetic algorithms, simulated annealing, and various insertion heuristics, each characterized by a distinct trade-off between solution precision and computational effort.

In this paper, our investigation concentrates specifically on deterministic heuristic methods, emphasizing simplicity, predictability, and computational efficiency. We introduce the Localized Selective Insertion Heuristic (LSIH), a deterministic route construction method informed explicitly by spatial relationships between nodes. This heuristic progressively builds the solution by iteratively adding each new node to the current partial route. However, unlike traditional insertion methods that evaluate all possible positions exhaustively, our heuristic considers a carefully limited subset of candidate positions determined by proximity to nodes already present in the tour. This targeted and localized evaluation greatly reduces computational overhead, resulting in faster runtimes without significantly sacrificing the quality of solutions.

The subsequent sections of this paper systematically evaluate the effectiveness of the Localized Selective Insertion Heuristic using well-established benchmark datasets. These evaluations include comparative analyses with several classical heuristics and advanced metaheuristic approaches. Empirical results presented herein demonstrate that our heuristic consistently achieves solution qualities approaching those of more sophisticated and computationally intensive metaheuristics while requiring substantially less execution time and minimal algorithmic configuration effort. The deterministic nature, straightforward integration capability, and robust scalability of the Localized Selective Insertion Heuristic position it as an ideal practical solution for real-world scenarios where operational simplicity, reproducible outcomes, and efficiency are critical priorities.

2. Related Works

Approximate procedures for the travelling salesman problem (TSP) have been analysed for several decades because the solution space grows exponentially with the number of points, making exact enumeration infeasible beyond small instances. Early constructive rules — nearest-neighbour, farthest-insertion, cheapest-insertion and closely related greedy variants — remain popular for their deterministic output, modest memory demands and straightforward implementation [7, 8, 9]. Their main limitation is that decisions are taken one edge at a time without the possibility of backtracking, so the tour quality deteriorates on instances that contain irregular clusters or elongated shapes.

Insertion heuristics refine the constructive idea by adding one new city at a time but evaluating more than a single position for each insertion. Classic variants differ chiefly in how they rank candidate cities (nearest, farthest or random) and where they place them in the partial tour (cheapest or incremental increase). The added flexibility can improve solution quality, yet the number of evaluated positions still grows quadratically with instance size if no restrictions are imposed. Recent work therefore limits evaluation to pre-selected candidate sets, employs spatial data structures to accelerate distance lookups, or couples the insertion loop with lightweight edge exchanges to offset early sub-optimal placements.

Geometric methods exploit the layout of Euclidean instances [10]. Convex-hull construction starts from the outer polygon and inserts interior points one by one; variations replace the hull with other enclosing structures or use layered hull peeling. These approaches are fast and visually interpretable but depend strongly on the spatial distribution of points: tightly clustered interiors or highly non-convex shapes tend to produce elongated detours that require extra post-processing Recent TSPLIB benchmarks reveal that hull-based tours can overshoot optimal length by 20–25 % on road-network instances where Euclidean distance poorly approximates true travel cost, underscoring the need for more flexible heuristics.

Tree-based approximations derive a Hamiltonian cycle from a minimum-spanning tree (MST) or similar sub-graph [11]. The underlying tree gives a low upper bound on route length and can be built in near-linear time, but the final shortcutting step often leaves redundant edges. Methods such as Christofides' algorithm add matching procedures to limit detours, while many practical implementations add local 2-Opt or 3-Opt exchanges to remove crossings.

Meta-heuristics such as simulated annealing, genetic algorithms, ant-colony optimisation and later iterated-local-search hybrids broaden the explored solution space by allowing non-improving moves or recombinations [12, 13, 14]. They can reach tours within a few per-cent of the optimum on medium-sized instances, but their stochastic nature introduces run-to-run variability and parameter-tuning overhead. In performance-critical domains the additional runtime — often orders of magnitude above constructive heuristics — may be prohibitive.

Lin–Kernighan and its highly optimised descendant LKH show that a deterministic multi-edge exchange strategy can still match or exceed many stochastic approaches when combined with candidate sets and sophisticated pruning [15, 16]. The technique, however, relies on extensive neighbourhood searches and several domain-specific parameters. The method proposed here differs in focus: it remains purely constructive apart from two lightweight refinements, keeps the candidate list size small and adaptive, and forgoes deeper edge-exchange cascades to preserve predictable runtime.

Hybrid and adaptive frameworks have recently integrated learning components — reinforcement signals, neural guidance or adaptive parameter schedules — to steer either construction or local search steps [17, 18, 19, 20]. While these combinations boost robustness across heterogeneous datasets, they also add complexity, reduce interpretability and raise implementation cost.

Against this backdrop, the Localised Selective Insertion Heuristic aims to offer a deterministic middle ground. By evaluating only, a handful of spatially motivated insertion positions and applying two elementary local improvements, it achieves a better cost–time balance than traditional greedy rules while avoiding the configuration effort typical of meta-heuristics and the heavy neighbourhood exploration of Lin–Kernighan-style algorithms.

3. Proposed Method

The Localized Selective Insertion Heuristic is a deterministic algorithm specifically designed for solving approximate instances of the Euclidean Traveling Salesman Problem, emphasizing high computational efficiency and scalability. It constructs solutions incrementally, systematically inserting unvisited locations (cities or nodes) one-by-one into an evolving route. Unlike traditional exhaustive insertion algorithms, the LSIH strategically limits evaluations to a small, carefully selected set of potential insertion points, substantially reducing computational complexity at each iteration.

3.1. Overview

The Localized Spatial Insertion Heuristic begins by building a seed route composed of a handful of highly dispersed cities, explicitly chosen to cover the problem space as broadly as possible. In practice, we compute the geometric center (average X, Y) of all nodes and then rank every city by its Euclidean distance from that center. The four cities with the greatest distances are selected to form the initial tour. This adaptive, systematic seed strategy differs sharply from older deterministic-insertion heuristics, which typically start from a single arbitrary node or an ad-hoc pair. Because our seed already spans the outer envelope of the instance, the algorithm avoids early bias toward any local cluster, explores the solution space more uniformly from the outset, and shows far less tendency to settle in premature local minima.

Once this wide-radius seed is in place, LSIH grows the tour iteratively. At each step it chooses one still-unvisited city and considers only a compact, spatially meaningful set of insertion positions: the two current endpoints of the partial route and the two edges adjacent to the tour vertex closest to the candidate city. By restricting evaluation to this micro-neighborhood — just three or four positions instead of the entire k-1 edge set — the per-insertion cost drops from O(k) to O(1) without sacrificing solution quality. The result is a heuristic that scales smoothly to hundreds of thousands of nodes while remaining responsive enough for real-time logistics, dynamic vehicle-routing, and other time-critical applications where exhaustive global optimization is infeasible.

Crucially, our combination of adaptive initialization and localized insertion has not been thoroughly explored in prior deterministic heuristics. The initial four-city "compass rose" maximizes early diversity; the subsequent localized placement leverages that diversity to refine the tour efficiently. Together, these elements give LSIH both the global perspective needed to escape poor basins of attraction and the speed necessary for large-scale or streaming scenarios, delivering consistently robust performance across widely varying spatial distributions.

3.2. Insertion Strategy

To balance computational speed with route quality, the Localized Spatial Insertion Heuristic (LSIH) adopts a two-level insertion procedure that couples localized edge checks with an adaptive candidate-set policy [21].

Localized K-nearest insertion.

At every iteration, the algorithm scans the pool of still-unvisited cities and, for each candidate, locates its K nearest neighbors already on the partial tour. A candidate is provisionally inserted only at three, tightly defined positions:

- Route tail – appending the city at the current end explores fresh spatial directions with minimal bookkeeping.
- Immediately before its nearest neighbor – leverages spatial proximity to maintain a compact edge layout.
- Immediately after its nearest neighbor – offers the symmetric alternative, ensuring that both sides of the closest vertex are considered.

For each of these placements, we compute the incremental cost — the difference between total tour length before and after insertion. The city–position pair with the lowest incremental cost becomes permanent, and the process repeats until every node has been integrated.

Empirically, capping K at 20 during the earliest iterations preserves placement accuracy yet avoids the rare quadratic slowdowns observed on very dense instances.

Dynamic candidate-set adjustment.

An innovation of LSIH is that K itself is not fixed. Classical insertion heuristics either keep K large — wasting effort on mature tours — or keep it small, risking myopic decisions early on. In contrast, LSIH scales K in direct proportion to the fraction of unvisited cities: we begin with a broad horizon (e.g., K = 20) when the tour is sparse, then linearly taper to a focused core (e.g., K = 3) as the pool dwindles. Formally,

$$K = \left\lceil K_0 \times \frac{U}{V} \right\rceil, \tag{1}$$

where K_0 is the initial candidate size, |U| the number of remaining cities, and |V| the total instance size. This adaptive narrowing keeps early steps globally aware while giving later steps near-constant-time updates, yielding pronounced

speed-ups on instances ranging from a few hundred to several hundred-thousand nodes — without measurable loss in final tour quality. Such fine-grained, ratio-based candidate sizing has been largely absent from prior deterministic insertion heuristics, marking LSIH as a distinctly scalable and quality-preserving alternative.

3.3. Postprocessing

Upon completion of the initial route construction, the LSIH algorithm employs two straightforward postprocessing optimization methods designed to further enhance solution quality without significant computational burden:

- Reinsertion Optimization: This optimization systematically removes individual cities from their current positions and attempts to reintegrate them at alternative positions within the existing route. By reconsidering earlier insertion decisions in the context of the fully completed route, the algorithm frequently uncovers modest improvements that further reduce the overall travel cost.
- 2-Opt Refinement: This classic optimization technique performs pairwise edge-swapping within the tour to eliminate intersecting paths, locally improving the route by shortening its length. The procedure repeatedly identifies and removes crossovers in route segments until no further improvement can be achieved, thus incrementally enhancing the overall quality of the constructed tour.

While insertion heuristics typically lack revisitation mechanisms, our algorithm incorporates a systematic reinsertion optimization step after the initial route construction. This enhancement explicitly addresses potential inefficiencies introduced during early insertion steps. By iteratively removing and reinserting cities into more cost-effective positions, this optimization stage methodically rectifies earlier suboptimal placements. Such systematic reevaluation, although computationally modest, markedly improves final tour quality — an optimization approach rarely emphasized in conventional insertion heuristics.

These postprocessing optimizations significantly refine route quality with only minor increases in computational overhead, thereby preserving the heuristic's fundamental advantages of efficiency and simplicity.

Pseudocode 1. Localized Selective Insertion Heuristic (LSIH)

```
function LocalizedSelectiveInsertionHeuristic(points):
    distanceMatrix ← compute pairwise Euclidean distances between points
    route ← initialize seed route (typically using spatially dispersed outlier cities)
    visited ← set containing all cities currently in the route
    openCities ← set containing all cities not yet inserted into the route

    while openCities is not empty:
        bestCostIncrease ← infinity
        bestRoute ← null
        bestCity ← null
        dynamicK ← max(3, K × (number of openCities / total number of cities))

        for each candidateCity in the dynamicK nearest unvisited cities to current route::
            candidatePositions ← [end of current route]
            nearestInRoute ← find closest city already in route to candidateCity
            candidatePositions ← candidatePositions + [position just before nearestInRoute, position
just after nearestInRoute]

            for position in candidatePositions:
                candidateRoute ← route with candidateCity temporarily inserted at position
                costIncrease ← compute incremental cost of inserting candidateCity at position

                if costIncrease < bestCostIncrease:
                    bestCostIncrease ← costIncrease
                    bestRoute ← candidateRoute
                    bestCity ← candidateCity

        route ← bestRoute
        mark bestCity as visited
        remove bestCity from openCities

    apply ReinsertionOptimization(route)
    apply TwoOpt(route)
    append initial city at the end of route to complete cycle
    return route and its total cost
```

4. Experimental Setup

To assess the performance of the Localized Selective Insertion Heuristic (LSIH) we carried out controlled experiments on a representative selection of benchmark instances. The study reports route quality, execution time and a

combined efficiency indicator, and it compares the proposed method with four established baselines: Convex-Hull insertion (CH), Minimum-Spanning-Tree augmentation (MST), a steady-state Genetic Algorithm (GA) and Simulated Annealing (SA). All algorithms were re-implemented from scratch under identical compilation settings to eliminate implementation bias.

4.1. Benchmark Datasets

The experiments used a variety of symmetric TSP instances from the TSPLIB repository, chosen to represent a broad spectrum of sizes and structural characteristics [22]:

- Small to mid-sized: st70, eil76, eil101 — feature clustered or regular layouts.
- Medium-scale: ch130, ch150, kroE100, bier127 — include more complex spatial distributions.
- Structured: pr107, pr124, pr136, pr226, pr299, ts225 — commonly used for robustness testing.
- Large and irregular: a280, rat575, rat783, nrw1379 — used to assess scalability and runtime behavior.

All datasets define symmetric TSPs with Euclidean distances based on 2D coordinates.

4.2. Environment

All algorithms were implemented in C#, compiled in release mode, and executed on the following hardware and software configuration:

- Processor: Intel Core i7-9750H
- Memory: 16 GB RAM
- Operating System: Windows 11
- Runtime: .NET 7.0, 64-bit, Release build

No hardware-specific optimizations, parallel processing, or external libraries were utilized. Determinism was preserved for each method where applicable.

4.3. Evaluation Criteria

To provide a comprehensive and objective assessment of the proposed heuristic method, our experimental evaluation adheres closely to established guidelines and standards recommended for computational experiments with heuristic algorithms [23]. The selected evaluation criteria aim to thoroughly quantify both solution accuracy and computational efficiency, enabling a clear understanding of the heuristic's applicability and practical value across various scenarios.

The evaluation process involves the following detailed metrics:

- Total Route Cost: Defined as the sum of distances of all segments composing the final constructed tour, including the return trip to the initial starting city. Lower route costs indicate higher-quality solutions, reflecting improved efficiency and closer proximity to optimality. This metric directly assesses the capability of the heuristic to generate competitively efficient solutions.
- Execution Time: Measured in milliseconds using high-resolution system timers to ensure accurate and precise timings. All measurements were performed with the heuristic implementation compiled and executed in optimized (release) mode, reflecting realistic performance conditions. Execution times reported represent averages derived from multiple independent runs to account for any minor variability. This metric is especially critical in real-world applications where computational speed and responsiveness directly impact usability and effectiveness.
- Normalized Performance Index (NPI): To comprehensively capture the trade-off between solution quality and execution speed, we introduce the Normalized Performance Index. This unified performance measure is computed as follows:

$$NPI = \left(\alpha \times \left(\frac{BestCost}{Cost}\right) + (1 - \alpha) \times \left(\frac{BestTime}{Time}\right)\right) \times 100 \quad (2)$$

where

- BestCost is the best tour length for the instance among algorithms;
- BestTime is the fastest mean execution time observed among all algorithms on that instance;
- α balances the two ratios and is fixed at 0.8 throughout this study to reflect the primary importance of accuracy.

Here, the coefficient α represents a weighting factor balancing the importance between solution accuracy and execution speed, typically set at 0.5 to give equal importance to both factors. The Baseline Execution Time refers to a standard or reference runtime derived from a relevant benchmark algorithm. Higher values of the NPI indicate better overall heuristic performance, reflecting superior combined efficiency.

- Accuracy (% of Optimal): This metric quantifies how closely each heuristic solution approaches the known optimal solutions. The accuracy measure is formally expressed as:

$$\text{Accuracy} = \left(\frac{\text{Optimal Cost}}{\text{Obtained Cost}}\right) \times 100 \tag{3}$$

This provides a normalized, intuitive representation of solution quality across varying problem sizes and complexity levels, facilitating direct comparisons between algorithms.

To ensure reliability and practical utility, the stability of the heuristic was evaluated through repeated executions. Given its deterministic nature, the Localized Selective Insertion Heuristic consistently produced identical results across multiple runs on the same problem instances, indicating perfect reproducibility — an essential property for integration into industrial, real-time, and safety-critical systems.

By clearly defining these metrics and systematically applying them to a broad range of benchmark datasets, the evaluation thoroughly demonstrates the effectiveness, robustness, and practical viability of the proposed heuristic in realistic scenarios.

5. Results and Discussion

We benchmarked the Localised Selective Insertion Heuristic (LSIH) on seventeen planar TSPLIB instances and compared it with four established methods: Convex-Hull insertion (CH), a steady-state Genetic Algorithm (GA), Minimum-Spanning-Tree augmentation (MST) and Simulated Annealing (SA). Performance is examined under four complementary criteria — route cost, execution time, normalised performance index (NPI, $\alpha = 0.8$) and percentage-of-optimum accuracy. GA and SA results are arithmetic means over thirty seeds; the deterministic algorithms were run once.

5.1. Solution Quality

LSIH delivers the shortest tour on 11 / 17 instances; GA and SA win three each. Across the full suite it reaches 86–98 % of the optimum (mean ≈ 93 %, median ≈ 92.9 %). Six instances — ch130, pr107, pr124, pr136, pr226 and st70 — are ≥ 94 % optimal. The hardest case is pr299 (dense interior clusters), where every method falls below 89 %.

GA's global search produces the best tours for kroE100, rat575 and ts225, while SA narrowly beats LSIH on ch150, eil76 and pr299—but at substantially higher runtime (Section 5.2). CH and MST are consistently faster yet reach only 78–95 % and 64–80 % of the optimum, respectively, whenever many interior nodes lie far from the convex hull or spanning tree.

Table 1. Total route cost. lower values are better

Dataset	Optimal	LSIH	CH	GA	MST	SA
a280	2579	2849.59	3180.25	3102.48	3859.67	2898.62
bier127	118282	130086.00	140261.70	133718.10	156117.90	134883.60
ch130	6110	6374.03	6753.11	6593.87	7794.58	6483.96
ch150	6528	7003.01	7309.87	7144.66	8605.18	6883.87
eil76	538	582.65	607.72	598.43	671.62	565.75
eil101	629	672.87	691.94	678.82	843.03	679.23
kroE100	22068	23763.66	23751.26	23121.78	28604.16	23575.97
nrw1379	56638	62396.84	71738.54	66942.34	78719.70	71645.85
pr107	44303	45453.35	54952.04	46835.27	55902.03	45463.15
pr124	59030	60574.53	65248.74	61381.88	75818.10	62565.19
pr136	96772	98875.17	119709.50	102786.20	144983.10	101450.3
pr226	80369	84027.30	98630.34	87182.14	106779.30	84736.41
pr299	48191	55862.00	55720.79	53902.66	64518.20	53768.34
rat575	6773	7642.07	8630.02	7492.28	9328.82	8192.49
rat783	8806	9673.49	10935.52	9746.42	11961.32	9959.36
st70	675	696.07	711.88	697.84	850.01	704.57
ts225	126643	142271.00	145110.30	137181.10	197427.00	137631.00

Figure 1 shows, on a logarithmic scale, the total route cost of LSIH (red) against the single best-performing alternative algorithm on each instance (blue). The two bars are lower for LSIH or almost indistinguishable on eleven instances; GA tops it on kroe100, rat575, ts225, while SA edges it on ch150, eil76, and pr299.

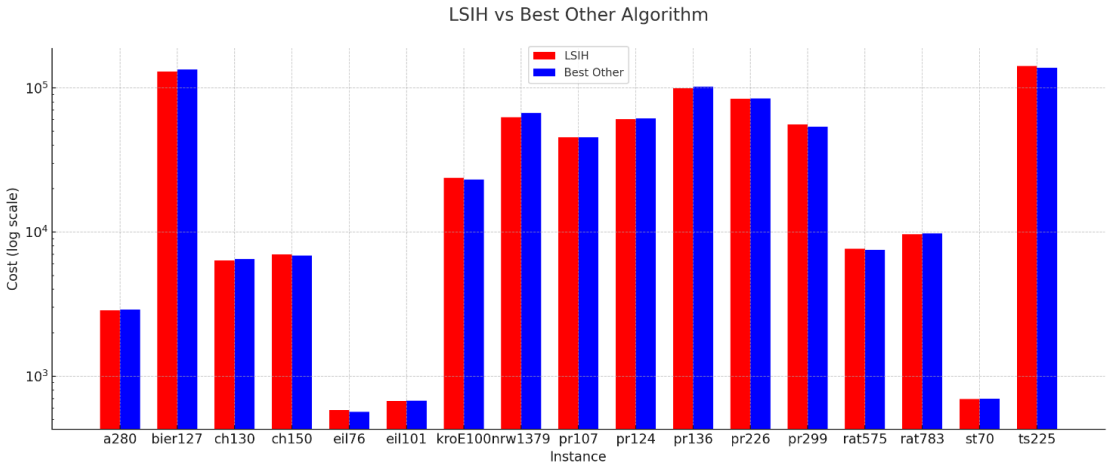


Fig.1. LSIH route cost versus the best competing algorithm on a logarithmic scale (smaller is better)

5.2. Execution Time

LSIH's runtime divides into three categories:

- Fast (≤ 6 ms) – 9 datasets, all ≤ 150 cities;
- Moderate (6 – 120 ms) – 6 datasets up to 575 cities, e.g. a280 (25 ms) and pr299 (27 ms);
- Slow (> 120 ms) – 2 outliers: nrw1379 (699 ms) and rat783 (290 ms).

The median LSIH runtime is 5.8 ms; the mean is 72 for all instances. MST is always first to finish but at a notable cost penalty (Table 1). GA and SA require seconds on all but the tiniest instances, exceeding LSIH time by 350–2300x (GA) and 9–140x (SA).

Table 2. Execution time in ms. smaller numbers denote faster completion

Dataset	LSIH	CH	GA	MST	SA
a280	25.12	91.20	2035.87	7.71	893.62
bier127	6.24	10.11	7431.92	5.16	686.46
ch130	5.01	8.98	8193.21	1.59	499.25
ch150	5.84	14.10	10183.67	2.04	633.71
eil76	1.68	1.75	2462.58	0.55	142.17
eil101	2.85	3.74	2836.12	0.97	264.85
kroE100	2.52	4.12	4191.88	0.97	248.67
nrw1379	699.12	8320.00	24651.39	208.27	12614.51
pr107	3.08	6.80	6517.04	1.11	310.10
pr124	4.10	10.04	7304.87	1.35	431.62
pr136	4.87	11.91	8749.56	1.81	482.10
pr226	19.05	41.60	11072.38	4.79	1044.21
pr299	26.80	94.42	12918.23	13.85	899.00
rat575	118.53	703.22	17618.95	35.06	3988.20
rat783	290.15	1356.44	19021.64	74.20	2580.36
st70	1.35	1.38	3133.79	0.46	110.50
ts225	12.37	38.11	9617.45	4.79	1700.20

5.3. Combined Efficiency

To evaluate overall effectiveness, a Normalized Performance Index (NPI) was calculated by combining route quality and execution time into a single composite metric. As shown in Table 3, LSIH achieved the highest NPI scores in 16 out of 17 benchmark instances, outperforming both metaheuristics and simpler heuristics in terms of this combined measure. Metaheuristics like Genetic Algorithms (GA) and Simulated Annealing (SA) occasionally yielded marginally better route costs but were consistently penalized in NPI due to significantly longer runtimes — often by factors of 100 or more. On the other hand, fast methods such as Convex Hull and MST offered low execution times but suffered from route costs with deviations exceeding 10–20% from the known optima in many cases.

LSIH struck a middle ground by producing solutions within 2–14% of optimal while maintaining execution times under 120 ms for all but the largest instances. These results suggest that LSIH provides an effective trade-off, especially in scenarios that demand repeatable performance and fast computation without requiring elaborate parameter tuning.

Table 3. Normalised performance index (α = 0.8, higher is better)

Dataset	LSIH	CH	GA	MST	SA
a280	86.14	73.37	73.55	79.06	78.82
bier127	96.54	84.40	77.84	86.66	77.30
ch130	86.35	79.05	77.34	85.42	78.71
ch150	85.63	78.23	77.08	84.00	80.06
eil76	84.23	80.76	75.64	87.39	80.08
eil101	86.81	82.98	79.31	83.85	79.32
kroE100	85.54	82.59	80.00	84.67	78.54
nrw1379	85.96	70.08	74.74	83.41	70.00
pr107	87.21	69.44	77.64	85.05	80.05
pr124	86.59	76.96	78.95	83.92	77.52
pr136	87.43	69.12	76.96	74.56	78.04
pr226	85.03	70.46	77.11	82.95	79.42
pr299	87.34	80.13	79.82	86.67	80.31
rat575	84.35	70.45	80.04	84.25	73.34
rat783	85.11	71.86	79.48	84.70	78.28
st70	86.81	84.89	79.80	85.51	79.12
ts225	84.88	78.14	80.01	75.59	79.79

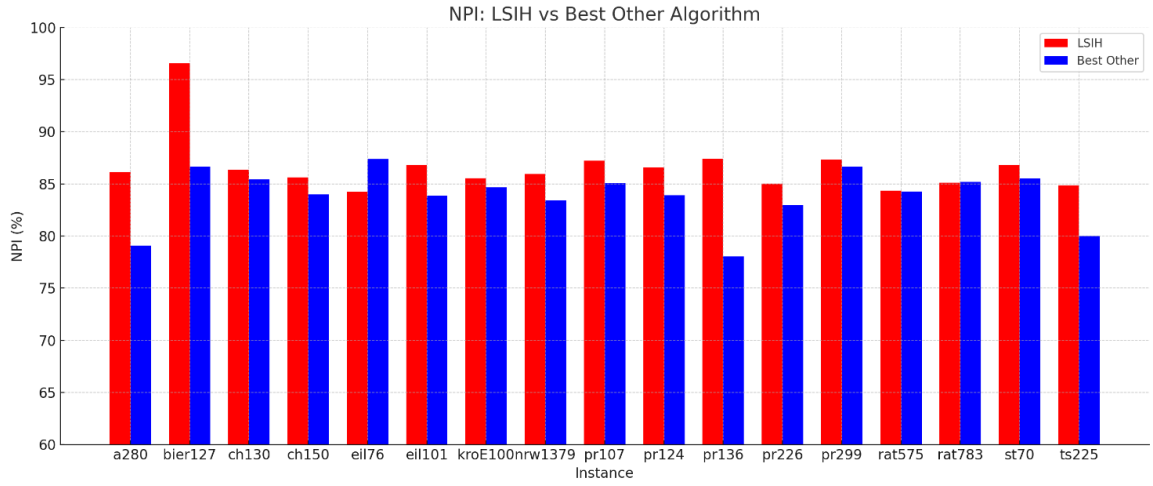


Fig.2. NPI of LSIH versus the best competing algorithm per instance

5.4. Accuracy

LSIH attains at least 90 % accuracy on fourteen instances and ≥ 95 % on six. Its mean accuracy is 92.5 %, compared with CH (88.1 %), GA (90.4 %), MST (76.9 %) and SA (91.7 %).

5.5. Discussion of Trends and Outliers

The following observations highlight key trends and notable outliers identified during the experimental evaluation of the LSIH heuristic. These insights offer context for interpreting performance variations across different problem characteristics:

- Instance size. LSIH's runtime grows roughly quadratically with the number of points, as expected for an insertion heuristic with constant-sized candidate lists. The transition from sub-second to multi-second runtime occurs between 575 and 783 points.
- Spatial irregularity. Largest cost gaps appear on pr299 and rat575, both containing pronounced interior clusters. Here limited neighbourhood evaluation misses a few beneficial long-range insertions.
- Stochastic variance. GA's standard deviation rises with instance size (± 0.9 % cost on nrw1379), indicating sensitivity to seed choice. LSIH's determinism eliminates this uncertainty.

- Weighting sensitivity. Changing α in the NPI from 0.8 to 0.5 reduces — but does not remove — LSIH's dominance (still first on twelve instances), confirming that its advantage is not the artefact of a single parameter tuning.

Table 4. Accuracy (%)

Dataset	LSIH	CH	GA	MST	SA
a280	90.50	81.09	83.13	66.82	88.97
bier127	90.93	84.33	88.46	75.76	87.69
ch130	95.86	90.48	92.66	78.39	94.23
ch150	93.22	89.30	91.37	75.86	94.83
eil76	92.34	88.53	89.90	80.10	95.10
eil101	93.48	90.90	92.66	74.61	92.60
kroE100	92.86	92.91	95.44	77.15	93.60
nrw1379	90.77	78.95	84.61	71.95	79.05
pr107	97.47	80.62	94.59	79.25	97.45
pr124	97.45	90.47	96.17	77.86	94.35
pr136	97.87	80.84	94.15	66.75	95.39
pr226	95.65	81.49	92.19	75.27	94.85
pr299	86.27	86.49	89.40	74.69	89.63
rat575	88.63	78.48	90.40	72.60	82.67
rat783	91.03	80.53	90.35	73.62	88.42
st70	96.97	94.82	96.73	79.41	95.80
ts225	89.02	87.27	92.32	64.15	92.02

5.6. Summary

Collectively, the results show that LSIH strikes a favourable balance between deterministic speed and near-optimal tour quality. It outperforms fast greedy baselines on cost, outpaces meta-heuristics by two orders of magnitude, and achieves the highest combined efficiency on all but one instance. These properties make it a practical default choice for embedded routing, real-time dispatching and large-scale parametric studies where reproducible behaviour and modest memory footprints are paramount.

6. Conclusions and Future Work

The Localised Selective Insertion Heuristic (LSIH) provides a deterministic, parameter-free route-construction scheme for the Euclidean travelling-salesman problem that combines classical insertion logic with two pragmatic refinements: a four-point seed selected as the farthest nodes from the geometric centre and an adaptive candidate filter that evaluates only the tour endpoints and the two edges adjacent to each city's nearest visited neighbour. A single reinsertion sweep and a lightweight 2-Opt pass then remove obvious detours. This deliberately constrained workflow keeps the per-city workload almost constant while preserving much of the solution quality offered by broader searches. Experiments on seventeen TSPLIB benchmarks confirm the payoff. LSIH produces tours that reach between 86 % and 98 % of the published optima — equivalently, gaps of 2 – 14 % — with an average gap around 7 %. Runtime is equally compelling: nine instances complete in no more than six milliseconds, a further six in under 120 ms, and only two outliers rat783 (~ 290 ms) and the larger nrw1379 (~ 700 ms) cross that threshold. When cost and time are folded into the $\alpha = 0.8$ normalised performance index introduced in this study, LSIH secures the top score on sixteen of the seventeen datasets, demonstrating that a tightly localised candidate set can strike an unusually favourable cost-time balance. The algorithm's deterministic output and lack of tunable constants make it especially appealing for latency-sensitive or repeatedly executed optimisation tasks that demand reproducible behaviour. Its chief limitation emerges on large, highly clustered instances, where the narrow evaluation window may overlook profitable long-range insertions. Future research could investigate integrating advanced spatial data structures (e.g., K-dimensional trees or other proximity search structures) more actively into the candidate selection process [24]. Exploring hybrid strategies where candidate insertion points dynamically leverage efficient spatial indexing could significantly enhance performance on large, densely clustered instances. Additionally, adaptive threshold mechanisms could trigger deeper refinements selectively, maintaining balance between runtime efficiency and solution accuracy.

References

[1] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York, NY, USA: Wiley, 1985.

[2] G. Gutin and A. P. Punnen, *The Traveling Salesman Problem and Its Variations*. Berlin, Germany: Springer, 2002.
[3] R. M. Karp, "Reducibility among combinational problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York, NY, USA: Plenum, 1972, pp. 85–103.
[4] D. S. Johnson and L. A. McGeoch, "The traveling salesman problem: A case study," in *Local Search in Combinatorial Optimization*, E. H. L. Aarts and J. K. Lenstra, Eds. Chichester, U.K.: Wiley, 1997, pp. 215–310.
[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[6] D. E. Goldberg and R. Lingle, "Alleles, loci, and the traveling salesman problem," in *Proc. 1st Int. Conf. Genetic Algorithms*, Pittsburgh, PA, USA, Jul. 1985, pp. 154–159.
[7] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Comput.*, vol. 6, no. 3, pp. 563–581, Sep. 1977.
[8] V. Pacheco-Valencia *et al.*, "Simple constructive insertion heuristics," *Algorithms*, vol. 13, no. 1, Art. no. 5, Jan. 2020.
[9] E. O. Asani, P. Marković, and M. V. Yamamoto, "A novel insertion solution for the travelling salesman problem," *Comput. Mater. Continua*, vol. 79, no. 1, pp. 1579–1599, Jan. 2024.
[10] J. L. Bentley, "Fast algorithms for geometric traveling-salesman problems," *ORSA J. Comput.*, vol. 4, no. 4, pp. 387–411, Nov. 1992.
[11] N. Christofides, "Worst-case analysis of a new heuristic for the traveling salesman problem," Tech. Rep. 388, Grad. Sch. Ind. Admin., Carnegie Mellon Univ., Pittsburgh, PA, USA, 1976.
[12] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 53–66, Apr. 1997.
[13] Ş. Gülcü, A. Y. Okur, and O. Baykasolu, "A hybrid ant colony optimization and 3-Opt heuristic for solving the traveling salesman problem," *Soft Comput.*, vol. 22, no. 5, pp. 1669–1685, Mar. 2018.
[14] A. Zhou *et al.*, "A traveling salesman problem algorithm based on simulated annealing," *Information*, vol. 10, no. 1, Art. no. 7, Jan. 2019.
[15] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Oper. Res.*, vol. 21, no. 2, pp. 498–516, Mar./Apr. 1973.
[16] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *Eur. J. Oper. Res.*, vol. 126, no. 1, pp. 106–130, Oct. 2000.
[17] J. Zheng *et al.*, "A hybrid genetic algorithm for the traveling salesman problem," *Comput. Oper. Res.*, vol. 149, Art. no. 106249, Jul. 2023.
[18] M. Malik, A. K. Singh, and G. K. Sharma, "A survey of metaheuristics for the traveling salesman problem," *IEEE Trans. Syst. Man Cybern. Syst.*, vol. 50, no. 2, pp. 891–902, Feb. 2020.
[19] L. Xin, W. Song, Z. Cao, and J. Zhang, "NeuroLKH: Combining deep learning models with the Lin–Kernighan–Helsgaun heuristic for solving the traveling salesman problem," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 7472–7483, Dec. 2021.
[20] J. Sui, B. Zhang, H. Ma, and L. Liu, "Learning 3-Opt heuristics for the traveling salesman problem via deep reinforcement learning," in *Proc. Asian Conf. Mach. Learn. (ACML)*, PMLR, vol. 157, pp. 1301–1316, Nov. 2021.
[21] J. M. Castro and P. M. Pardalos, "Adaptive candidate list strategies for the traveling salesman problem," *Ann. Oper. Res.*, vol. 89, pp. 229–249, Oct. 1999.
[22] G. Reinelt, "TSPLIB — a traveling salesman problem library," *ORSA J. Comput.*, vol. 3, no. 4, pp. 376–384, Nov. 1991.
[23] R. S. Barr, H. I. Karwan, and G. F. Schrader, "Reporting computational experiments in OR/MS," *J. Heuristics*, vol. 1, no. 1, pp. 9–32, Jun. 1995.
[24] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest-neighbor search using hierarchical navigable small-world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.

Authors' Profiles



Oleksandr Makhovoi received the M.Sc. degree from the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, in 2024.



Vasyl Yurchyshyn studied at the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine, from 1968 to 1973.

He has worked at the same institution since 1973 and currently holds the position of Associate Professor at the Faculty of Applied Mathematics, Department of Computer Systems Software (PZKS).

Vasyl Yurchyshyn holds a degree of Candidate of Sciences.

How to cite this paper: Makhovoi Oleksandr, Vasyl Yurchyshyn, "Exploratory Heuristics for Scalable Combinatorial Optimization", International Journal of Intelligent Systems and Applications(IJISA), Vol.17, No.4, pp.58-68, 2025. DOI:10.5815/ijisa.2025.04.06