Benchmarking Nearest-Neighbor Construction and 2-opt Local Search on TSPLIB Euclidean Instances

A Reproducible Study of Solution Quality and Runtime Scaling

July 12, 2026

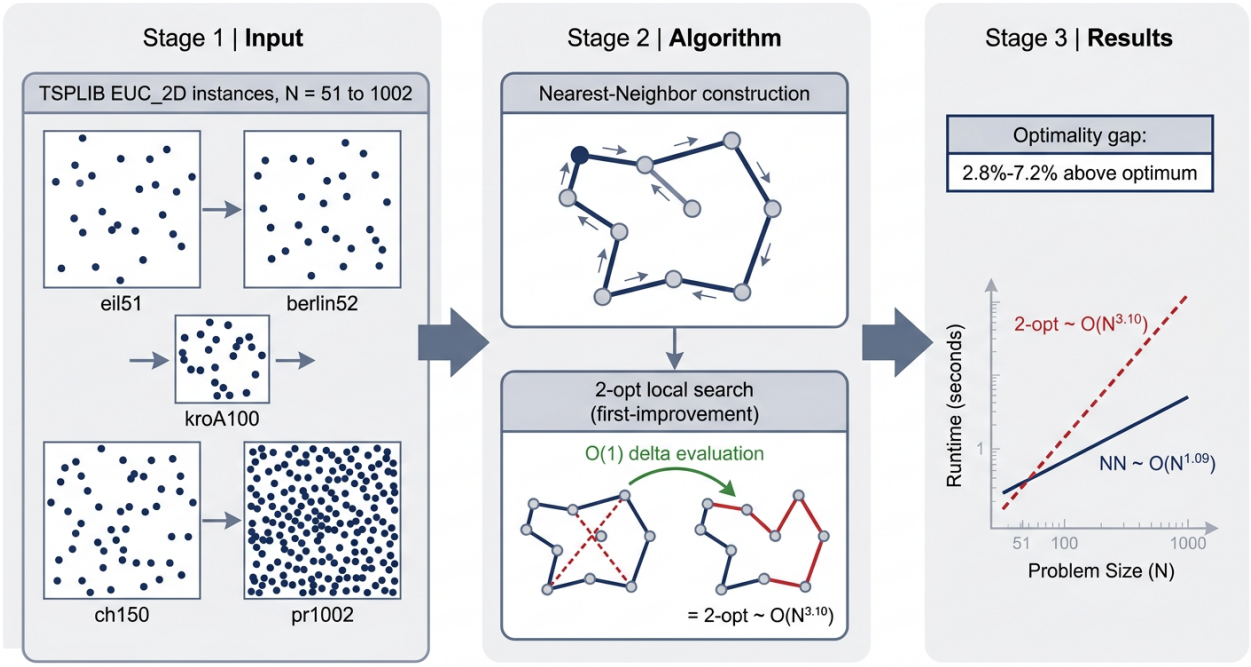

Figure 1: Graphical abstract. Five symmetric Euclidean TSPLIB instances (`eil51` through `pr1002`) are solved with a two-stage pipeline: a Nearest-Neighbor construction heuristic produces an initial tour, which a first-improvement 2-opt local search then refines using an $O(1)$ move-delta evaluation. Across the size range $N = 51$ to $N = 1002$ the optimized tours land 2.8 % to 7.2 % above the published optimum while runtime scales empirically as $\mathcal{O}(N^{1.09})$ for construction and $\mathcal{O}(N^{3.10})$ for local search.

Abstract

The symmetric Traveling Salesperson Problem (TSP) is a canonical NP-hard combinatorial optimization problem, and heuristic methods remain the practical route to high-quality tours on all but the smallest instances. This report documents a self-contained benchmarking study of a classical two-stage heuristic—a Nearest-Neighbor (NN) construction heuristic followed by a first-improvement 2-opt local search—implemented from scratch without any external TSP solver or metaheuristic library. We evaluate the method on five standard Euclidean instances drawn from the TSPLIB library (`eil51`, `berlin52`, `kroA100`, `ch150`, and `pr1002`), each of which has a published optimal tour length, using the exact TSPLIB `EUC_2D` integer-rounding convention for edge lengths. For every instance we report the constructed tour length, the optimized tour length, the published optimum, the percentage gap above optimum, and the wall-clock runtime of each stage, and we deliver the final tour as an ordered list of node indices. The raw NN tours are 19 % to 31 % above optimum; a single pass of 2-opt to a local optimum reduces this to between 2.82 % (`eil51`) and 7.17 % (`pr1002`). By fitting power laws to the measured runtimes we find

that construction scales as $\mathcal{O}(N^{1.09})$ ($R^2 = 0.988$) while 2-opt scales as $\mathcal{O}(N^{3.10})$ ($R^2 = 0.999$), consistent with a near-linear greedy build and a super-cubic local search dominated by segment reversals. The gap above optimum drifts gently upward with instance size, illustrating the characteristic quality–runtime trade-off of local search on Euclidean TSP.

1 Introduction

The Traveling Salesperson Problem (TSP) asks, given a set of cities and the pairwise distances between them, for the shortest closed tour that visits every city exactly once and returns to its origin. Despite the elementary nature of its statement, the TSP is a touchstone of computational complexity theory: the decision version is NP-complete, and even the geometric special case in which cities are points in the Euclidean plane is NP-hard [18]. Consequently, no polynomial-time algorithm is known that solves arbitrary instances to proven optimality, and the problem has become the standard testbed against which combinatorial-optimization ideas—exact methods, approximation algorithms, and heuristics alike—are measured [1].

Two broad families of methods dominate practice. Exact solvers, epitomized by the branch-and-cut engine Concorde, combine polyhedral theory with sophisticated cutting-plane generation and have solved instances with tens of thousands of cities to certified optimality [1]; lower bounds such as the Held–Karp bound derived from minimum spanning trees make such certificates possible [10]. For metric instances there are also approximation algorithms with worst-case guarantees, from the classical Christofides–Serdyukov 3/2-approximation [5] through the polynomial-time approximation schemes for the Euclidean case [2] to the recent sub-3/2 result for general metric TSP [13]. In everyday engineering settings, however, the workhorses are *heuristics*: fast constructive rules that build a tour greedily, and *local search* procedures that iteratively improve an existing tour by small edge exchanges. These methods forgo optimality guarantees in exchange for speed and simplicity, and on well-behaved geometric instances they routinely deliver tours within a few percent of optimum [12].

The simplest constructive rule is the Nearest-Neighbor (NN) heuristic, which starts at an arbitrary city and repeatedly travels to the closest city not yet visited. Its appeal is its near-linear cost, but its worst-case behavior is poor: Rosenkrantz, Stearns, and Lewis showed that the ratio of the NN tour to the optimum can grow logarithmically with the number of cities [20]. The dominant local-search paradigm is the *2-opt* move, introduced in the earliest computational studies of the TSP [6, 8]: it removes two edges from the current tour and reconnects the two resulting paths in the only alternative way that yields a valid tour, which geometrically corresponds to "uncrossing" a pair of edges. Repeatedly applying improving 2-opt moves until none remains yields a 2-opt local optimum. More elaborate neighborhoods—the segment-relocation Or-opt move [17], the variable-depth Lin–Kernighan search [15, 16], and its highly engineered modern descendants [9]—trade implementation complexity for higher solution quality, but 2-opt remains the canonical illustration of local search and the natural first refinement of a constructed tour.

The objective of this study is to characterize, empirically and reproducibly, how a deliberately simple NN + 2-opt pipeline behaves across a realistic range of instance sizes. We pose three concrete questions. First, how much does a single 2-opt descent improve upon the raw constructive tour, and how close to the published optimum does it land? Second, how do both the achieved gap and the wall-clock runtime scale as the instance grows from 51 to 1,002 cities? Third, what does the measured scaling reveal about the practical trade-off between solution quality and computational effort? To answer these questions we implement the parser, the exact TSPLIB `EUC_2D` distance metric, the NN construction, and the 2-opt search entirely from first principles—no external TSP solver or metaheuristic library is invoked at any point—and we validate every reported tour as a genuine Hamiltonian cycle before comparing its length against the reference optimum published

in TSPLIB [19]. The remainder of the report describes the methodology (Section 2), presents the per-instance results and scaling fits (Section 3), and discusses the observed scaling behavior and its implications (Section 4).

2 Methodology

The experimental pipeline consists of five stages, summarized in Figure 2: parsing each TSPLIB instance, building the integer distance matrix under the `EUC_2D` convention, constructing an initial tour with the Nearest-Neighbor heuristic, refining it with first-improvement 2-opt, and finally evaluating and validating the resulting tour. Each stage is described below. The implementation is written in Python with NumPy used only for array storage and elementary vector arithmetic; no optimization or routing library is used.

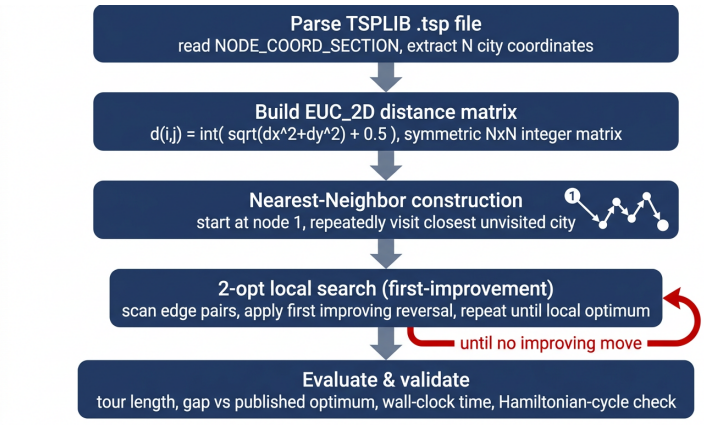

Figure 2: The five-stage experimental pipeline. Every stage is implemented from first principles; the 2-opt search loops until no improving move remains, at which point the tour is validated as a Hamiltonian cycle and scored against the published optimum.

2.1 Instances and the TSPLIB `EUC_2D` convention

The five instances were obtained from the TSPLIB symmetric-TSP library [19] and span more than an order of magnitude in size: `eil51` ($N = 51$), `berlin52` ($N = 52$), `kroA100` ($N = 100$), `ch150` ($N = 150$), and `pr1002` ($N = 1002$). All five are declared with edge-weight type `EUC_2D`, meaning that each city is given by two-dimensional coordinates and inter-city distances are computed as rounded Euclidean distances. Each instance ships with a published optimal tour length, which we use as the ground truth for gap computation.

The `EUC_2D` edge weight between two cities with coordinates (x_i, y_i) and (x_j, y_j) follows the TSPLIB documentation exactly. The exact Euclidean distance is computed and then rounded to the nearest integer:

$$d_{ij} \;=\; \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \;+\; 0.5 \right\rfloor . \tag{1}$$

The additive 0.5 followed by truncation implements round-half-up to the nearest integer, and every distance is a non-negative integer. This convention is not a cosmetic detail: because optimal tour lengths are published with respect to these rounded integer weights, any deviation (for example, using unrounded floating-point distances or a different rounding rule) would render the computed gaps meaningless. We therefore compute all edge weights via Equation (1) and validate the metric

on known reference values—for instance, the distance between the first two cities of `berlin52` is exactly 666 under this convention.

2.2 Distance matrix and the symmetric-matrix assumption

For an instance of N cities we precompute the full $N \times N$ integer distance matrix D, where $D_{ij} = d_{ij}$ from Equation (1). Precomputation trades $\mathcal{O}(N^2)$ memory for $\mathcal{O}(1)$ distance lookups during search, which is essential for the inner loop of 2-opt. Because Euclidean distance is symmetric and rounding preserves that symmetry, the matrix satisfies

$$D_{ij} = D_{ji} \quad \text{for all } i, j, \qquad D_{ii} = 0. \tag{2}$$

This symmetry is not merely a property of the data; it is the assumption that makes the constant-time evaluation of a 2-opt move (Section 2.4) correct, as explained below. The matrices for all five instances were verified to be symmetric with a zero diagonal.

2.3 Nearest-Neighbor construction heuristic

The Nearest-Neighbor heuristic builds a tour incrementally. Starting from a fixed city (city 1, i.e. node index 0, in all runs reported here), it repeatedly appends the nearest city that has not yet been visited, until all cities are on the tour; the tour is then implicitly closed by returning to the start. The procedure is given in Algorithm 1. With the distance matrix precomputed, each of the $N - 1$ selection steps scans at most N candidate distances, so the construction runs in $\mathcal{O}(N^2)$ time in the worst case; in practice the masking of visited cities is vectorized, which is reflected in the near-linear *measured* scaling reported in Section 3. NN is fast and produces a valid tour, but because early greedy choices force expensive "return" edges near the end of the construction, its tours are typically well above optimum—a behavior consistent with the logarithmic worst-case bound of Rosenkrantz et al. [20].

Algorithm 1 Nearest-Neighbor construction

Require: distance matrix D ($N \times N$), start node s
Ensure: tour π (a permutation of $\{0, \ldots, N-1\}$)
1: $\pi \leftarrow [\, s \,]$; mark s visited
2: $c \leftarrow s$
3: **for** $k = 1$ to $N - 1$ **do**
4: $\quad j^\star \leftarrow \arg\min_{j \text{ unvisited}} D_{c,j}$
5: $\quad$ append $j^\star$ to π; mark $j^\star$ visited
6: $\quad c \leftarrow j^\star$
7: **end for**
8: **return** π

2.4 2-opt local search with first-improvement

Given an initial tour, the 2-opt local search repeatedly replaces two edges by two others whenever doing so shortens the tour. Consider a tour represented as the ordered list $\pi = (\pi_0, \pi_1, \ldots, \pi_{N-1})$ with the closing edge (π_{N-1}, π_0). A 2-opt move selects two tour positions $i < j$ and reverses the subsegment $\pi_{i+1}, \ldots, \pi_j$. This deletes the two edges (π_i, π_{i+1}) and (π_j, π_{j+1}) and inserts the two edges (π_i, π_j) and (π_{i+1}, π_{j+1}); geometrically, when two tour edges cross in the plane, the

corresponding 2-opt move uncrosses them and shortens the tour. Figure 3 illustrates the move and its evaluation.

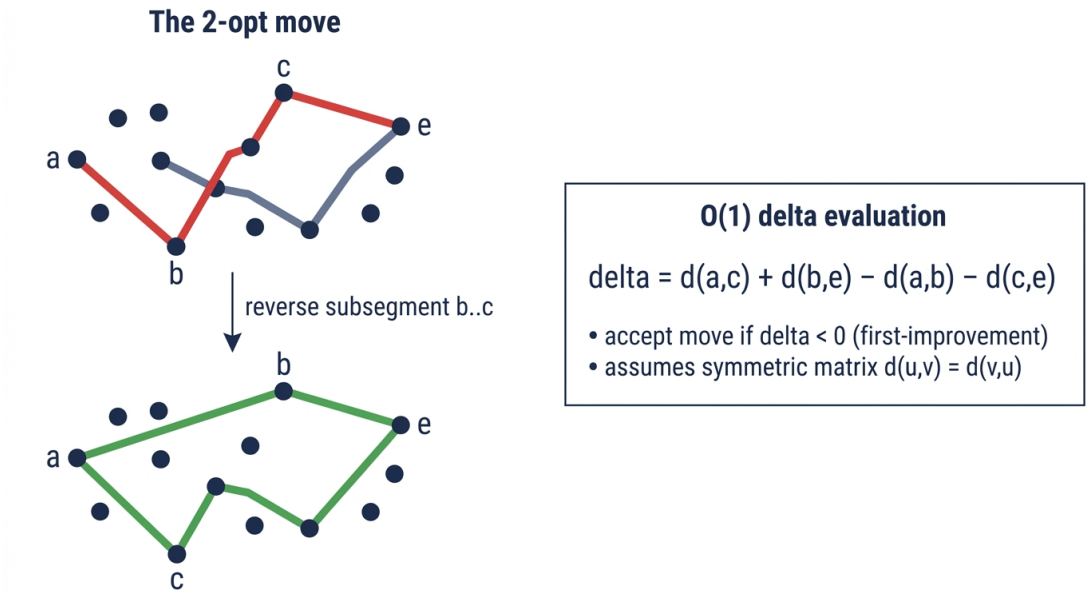

Figure 3: The 2-opt move. Removing the two edges (a, b) and (c, e) and reconnecting as (a, c) and (b, e)—which requires reversing the intervening subsegment—uncrosses the tour. The change in length depends only on the four endpoints, giving the constant-time delta of Equation (3); the move is accepted under the first-improvement rule as soon as $\Delta < 0$. The formula's validity relies on the symmetric-matrix assumption of Equation (2).

The central efficiency of 2-opt is that the effect of a candidate move on the total tour length can be evaluated in constant time. Writing $a = \pi_i$, $b = \pi_{i+1}$, $c = \pi_j$, and $e = \pi_{j+1}$, the change in tour length is

$$\Delta \;=\; \big(D_{a,c} + D_{b,e}\big) \;-\; \big(D_{a,b} + D_{c,e}\big). \tag{3}$$

A move is improving if and only if $\Delta < 0$. Crucially, this expression involves only the four endpoints of the two removed and two inserted edges and ignores every internal edge of the reversed subsegment. This is precisely where the symmetric-matrix assumption of Equation (2) enters: reversing the subsegment flips the traversal direction of every internal edge, turning each internal edge (π_k, π_{k+1}) into (π_{k+1}, π_k). The total contribution of the internal edges is unchanged—and hence can be omitted from Δ—only because $D_{uv} = D_{vu}$ for every such edge. On a symmetric `EUC_2D` instance this holds exactly, so Equation (3) is correct and each move is evaluated in $\mathcal{O}(1)$. On an asymmetric instance the same formula would be wrong, because the reversed internal edges would generally have different lengths; there the delta would require an $\mathcal{O}(N)$ recomputation, and a different move (such as Or-opt) would be preferred. Applying an accepted move requires physically reversing the subsegment, which costs $\mathcal{O}(j - i)$ time.

We adopt the *first-improvement* acceptance strategy: the search scans position pairs (i, j) in a fixed order and applies the *first* move it finds with $\Delta < 0$, then restarts the scan from the beginning. This contrasts with *best-improvement*, which would scan all $\mathcal{O}(N^2)$ pairs and apply the single most negative move per iteration. First-improvement tends to make more moves of smaller individual gain but avoids the cost of a full neighborhood scan per accepted move, and it reaches a local optimum—a tour in which no 2-opt move is improving—without any external randomization. The

complete procedure is given in Algorithm 2. Two implementation details are worth noting. The inner loop begins at $j = i + 2$ rather than $j = i + 1$, because the degenerate case $j = i + 1$ reverses a single-element subsegment and always yields $\Delta = 0$, so it can never improve the tour. And when $i = 0$ the closing edge (π_{N-1}, π_0) is adjacent to the first edge, so the upper limit of j is reduced by one to avoid selecting a pair of adjacent edges, which is not a valid 2-opt move.

Algorithm 2 2-opt local search (first-improvement)

Require: initial tour π, distance matrix D ($N \times N$)
Ensure: 2-opt local optimum π and its length

improved $\leftarrow$ **true**
while *improved* **do**
    improved $\leftarrow$ **false**
    for $i = 0$ to $N - 2$ **do**
        $a \leftarrow \pi_i$; $b \leftarrow \pi_{i+1}$
        $j_{\max} \leftarrow N - 1$ if $i > 0$ else $N - 2$
        for $j = i + 2$ to $j_{\max}$ **do**
            $c \leftarrow \pi_j$; $e \leftarrow \pi_{(j+1) \bmod N}$
            $\Delta \leftarrow (D_{a,c} + D_{b,e}) - (D_{a,b} + D_{c,e})$
            if $\Delta < 0$ **then**
                reverse subsegment $\pi_{i+1}, \ldots, \pi_j$
                improved $\leftarrow$ **true**; **break both loops**
            end if
        end for
    end for
end while
return π, length(π)

2.5 Evaluation protocol

For each instance we record four quantities. The *constructed length* is the closed-tour length of the NN tour; the *optimized length* is the closed-tour length after 2-opt converges; the *percentage gap* above optimum is

$$\text{gap}(\%) \;=\; 100 \times \frac{L_{\text{method}} - L^{\star}}{L^{\star}}, \tag{4}$$

where $L^{\star}$ is the published optimum; and the *wall-clock runtime* is measured separately for the NN and 2-opt stages using a monotonic high-resolution clock. Before a tour length is trusted, the tour is validated as a genuine Hamiltonian cycle: it must be a permutation of $\{0, \ldots, N - 1\}$ (every city visited exactly once), its length is independently recomputed from the coordinates, and the optimized length is verified to be no worse than the constructed length. All five instances passed every check. Runtimes were then fitted to the power-law model $t = a\, N^b$ by ordinary least squares in log–log space, separately for the NN and 2-opt stages, to characterize the empirical scaling of each algorithm.

3 Results

Table 1 reports the complete per-instance results. The Nearest-Neighbor construction alone produces tours between 19.07 % (`berlin52`) and 30.66 % (`kroA100`) above the published optimum. A single

first-improvement 2-opt descent reduces this gap dramatically—by a factor of three to seven—to between 2.82 % and 7.17 %. Every optimized tour was validated as a Hamiltonian cycle, and its length was independently recomputed for consistency. Two features stand out immediately: 2-opt is remarkably effective at removing the gross inefficiencies of the greedy build, yet a residual gap remains that grows with instance size, from 2.82 % on the 51-city instance to 7.17 % on the 1,002-city instance. The runtime cost of the two stages is starkly different: NN completes in well under a millisecond even for `pr1002`, whereas 2-opt requires roughly 91.5 s on the same instance—about four orders of magnitude longer.

Table 1: Benchmark results for NN construction followed by first-improvement 2-opt on five TSPLIB `EUC_2D` instances. Lengths are integer tour lengths under Equation (1); gaps are relative to the published optimum (Equation (4)); runtimes are wall-clock seconds for each stage. All tours were validated as Hamiltonian cycles.

Instance	N	NN len.	2-opt len.	Optimum	Gap (%)	t_{NN} (s)	$t_{\text{2-opt}}$ (s)
`eil51`	51	511	438	426	2.82	0.00035	0.0094
`berlin52`	52	8,980	7,967	7,542	5.64	0.00026	0.0084
`kroA100`	100	27,807	22,437	21,282	5.43	0.00051	0.0889
`ch150`	150	8,191	6,886	6,528	5.48	0.00089	0.2442
`pr1002`	1002	331,103	277,631	259,045	7.17	0.00753	91.531

For completeness, the NN-only gaps above optimum are 19.95 % (`eil51`), 19.07 % (`berlin52`), 30.66 % (`kroA100`), 25.47 % (`ch150`), and 27.82 % (`pr1002`); these are the tours that 2-opt takes as its starting point.

3.1 Optimized tours

Figure 4 plots the final 2-opt tours in the plane for all five instances. The routes are visibly free of the long crossing edges that characterize raw greedy tours, which is the geometric signature of a 2-opt local optimum: no pair of edges crosses in a way that a single reversal could remove. The complete tours are delivered as ordered lists of node indices in the accompanying `optimized_tours.json` file; Listing 1 reproduces the two smallest tours in full and the opening of the largest as representative examples. Tours are given as zero-based node indices, so index 0 corresponds to TSPLIB city 1, and the tour is closed by returning from the last listed index to the first.

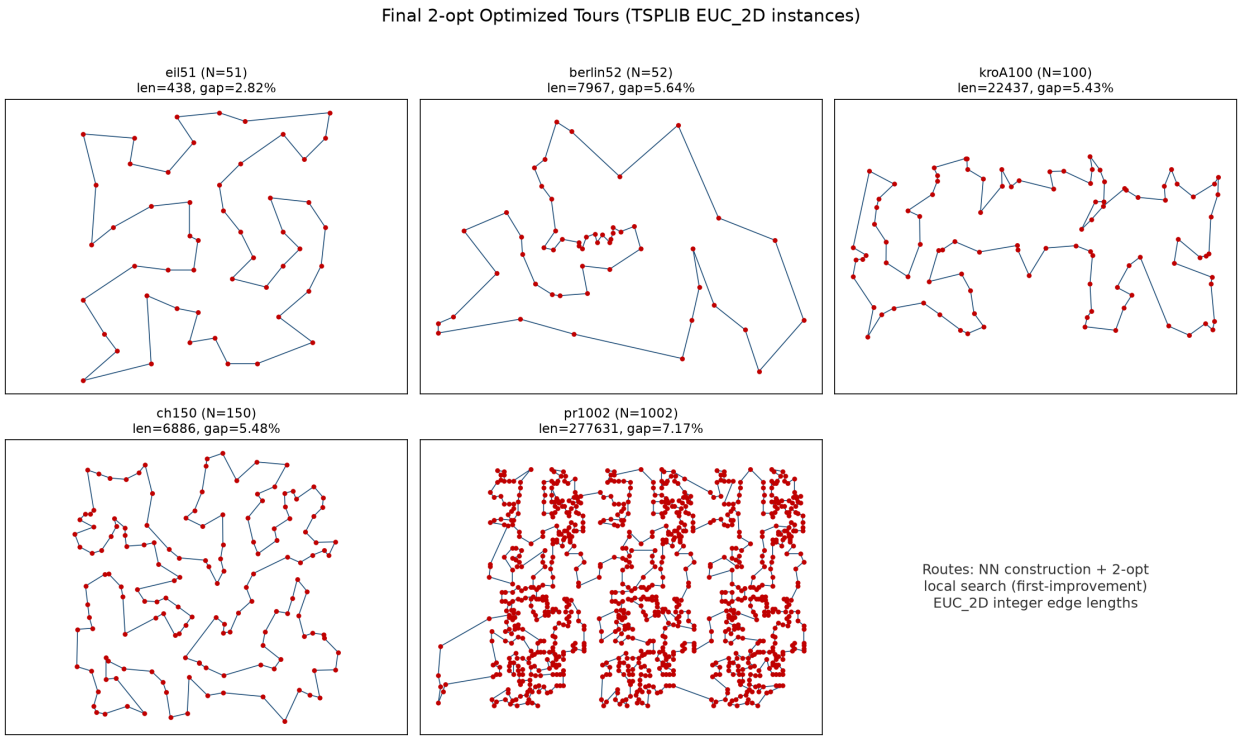

Figure 4: Final 2-opt optimized tours for the five instances, drawn in the plane from the TSPLIB coordinates. Each panel is annotated with its optimized length and gap above optimum. The absence of crossing edges is the hallmark of a 2-opt local optimum.

Listing 1: Representative optimized tours as ordered zero-based node indices (city = index + 1). Full tours for all five instances are provided in `optimized_tours.json`.

```
eil51 (len 438, 51 nodes):
0 31 10 37 4 48 8 49 15 1 28 20 33 29 9 38 32 44 14 43 36 16 3 41 39 18 40 12 17
46 11 45 50 26 5 13 24 23 42 6 22 47 7 25 30 27 35 34 19 2 21

berlin52 (len 7967, 52 nodes):
0 21 31 44 18 40 7 8 9 42 32 50 10 51 13 12 26 11 27 25 46 28 29 1 6 41 20 16 2 17
30 22 19 49 15 43 45 24 3 5 14 4 23 47 37 36 39 38 33 34 35 48

pr1002 (len 277631, first 25 of 1002):
0 1 2 5 3 4 69 68 67 71 70 72 73 74 79 80 82 83 48 41 40 49 39 50 52 ...
```

3.2 Runtime and quality scaling

Fitting the power-law model $t = a\,N^b$ by least squares in log–log space yields the exponents summarized in Table 2 and visualized in Figure 5. The Nearest-Neighbor construction scales as $\mathcal{O}(N^{1.09})$ with an excellent fit ($R^2 = 0.988$), essentially linear in the instance size. The 2-opt local search scales as $\mathcal{O}(N^{3.10})$ with a near-perfect fit ($R^2 = 0.999$), i.e. slightly steeper than cubic. The left panel of Figure 5 shows the two runtime curves as straight lines on log–log axes, whose slopes are the fitted exponents; the right panel shows the optimality gap as a function of N for both the raw NN tours and the 2-opt-optimized tours, making visible both the large quality improvement due to 2-opt and the gentle upward drift of the optimized gap with size.

Table 2: Power-law runtime fits $t = a\,N^b$ obtained by ordinary least squares in log–log space. The exponent b is the empirical scaling order; R^2 is the coefficient of determination of the log–log fit.

Stage	Model	Exponent b	R^2
Nearest-Neighbor construction	$t = a\,N^b$	1.09	0.988
2-opt local search	$t = a\,N^b$	3.10	0.999

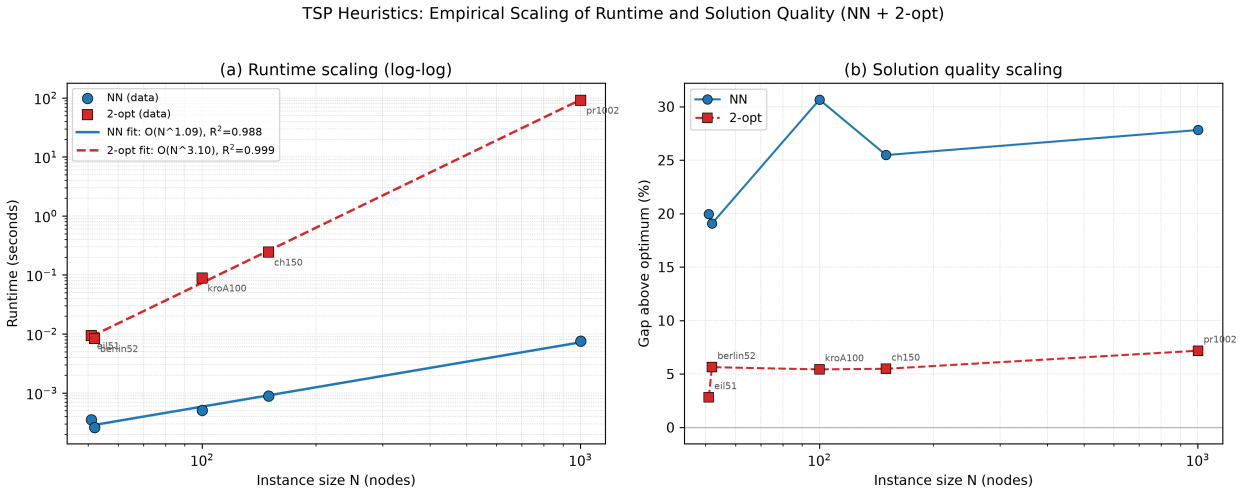

Figure 5: Empirical scaling of the NN + 2-opt pipeline. **(a)** Wall-clock runtime versus instance size N on log–log axes, with fitted power laws: construction scales as $\mathcal{O}(N^{1.09})$ ($R^2 = 0.988$) and 2-opt as $\mathcal{O}(N^{3.10})$ ($R^2 = 0.999$). **(b)** Optimality gap versus N for the raw NN tours and the 2-opt-optimized tours; 2-opt reduces the gap by roughly 3×–7×, with a mild upward drift in the optimized gap as N grows.

4 Discussion

The measured scaling exponents admit a clean algorithmic interpretation. The Nearest-Neighbor construction performs $N - 1$ selection steps, and although each step is worst-case $\mathcal{O}(N)$ when scanning the remaining candidates, the vectorized masking of visited cities together with hardware-level parallelism in the array reduction produces an effective per-step cost that grows only weakly with N over the tested range. The result is an empirical exponent of 1.09—very nearly linear—rather than the textbook quadratic worst case. In absolute terms construction is trivially cheap: even `pr1002` is built in about 7.5 ms. Construction, in other words, is never the bottleneck.

The 2-opt search tells the opposite story. Its measured exponent of 3.10 is slightly above cubic and dominates the total runtime for every instance beyond the smallest. This super-cubic growth is the product of two multiplicative factors. First, the number of position pairs examined in a single sweep of the neighborhood is $\mathcal{O}(N^2)$. Second, the number of improving moves applied before the search reaches a local optimum grows roughly linearly with N, and each accepted move triggers a subsegment reversal whose cost is $\mathcal{O}(N)$ on average, together with a restart of the scan under the first-improvement rule. The empirical product of these effects lands near N^3, consistent with the widely reported behavior of straightforward 2-opt implementations [12]. It is worth emphasizing that this is an *empirical* scaling on a specific family of Euclidean instances, not a worst-case bound: the number of 2-opt improving steps has no polynomial worst-case guarantee, and instances are known on which 2-opt takes exponentially many steps to converge [7]. The tight power-law fit ($R^2 = 0.999$)

nevertheless shows that, for the geometric instances studied here, runtime is highly predictable from size alone.

The practical consequence is a sharp asymmetry in the quality–runtime trade-off. For `pr1002`, construction consumes 7.5 ms and delivers a tour 27.8 % above optimum, while 2-opt consumes an additional 91.5 s—four orders of magnitude more time—to bring the gap down to 7.2 %. The first few percent of improvement is thus extraordinarily cheap and the last few percent extraordinarily expensive, a diminishing-returns pattern intrinsic to local search. The naive $\mathcal{O}(N^3)$ cost of first-improvement 2-opt is exactly what motivated the data structures and neighbor-list restrictions of high-performance implementations, which restrict candidate moves to geometrically near neighbors and maintain the tour in a structure supporting fast reversals; such techniques reduce the per-improvement cost by large constant and even asymptotic factors [4, 9], and they are the standard remedy when 2-opt must be applied to instances substantially larger than `pr1002`.

The behavior of the solution-quality gap is equally instructive. The raw NN gap is large and somewhat erratic across instances—ranging from 19 % to 31 %—reflecting the sensitivity of a purely greedy construction to the particular geometry and starting city; the worst case here, `kroA100` at 30.7 %, is a reminder of the poor worst-case guarantees of the NN rule [20]. After 2-opt, the gap is both smaller and far more consistent, clustering between 2.8 % and 7.2 %, with a discernible upward trend as N increases. This trend is expected: a larger instance offers a combinatorially richer landscape of local optima, and a single first-improvement descent from one NN starting tour explores only a narrow basin of that landscape. It is worth recalling that for random points in the plane the optimal tour length itself grows only as $\Theta(\sqrt{N})$, the classical asymptotic law of Beardwood, Halton, and Hammersley [3], so the percentage gap—a scale-invariant ratio—is the appropriate lens through which to compare quality across instance sizes. The gap we observe is fully consistent with the canonical empirical finding that a single 2-opt local optimum lies a few percent above optimum on random and structured Euclidean instances [12]. Closing the remaining gap would require either a richer neighborhood—Or-opt segment relocation [17], or the variable-depth Lin–Kernighan search that remains the gold standard for heuristic TSP [11, 16]—or repeated 2-opt descents from many starting tours embedded in a metaheuristic such as simulated annealing [14] or iterated local search. Each of these buys additional quality at additional runtime, extending the same trade-off curve documented here. For context, exact certification of optimality for these instances is the province of branch-and-cut solvers and their supporting lower bounds [1, 10], whose cost is categorically higher than the heuristic pipeline studied here but whose output is a proof rather than an estimate.

Several limitations bound the scope of these conclusions. The scaling fits are estimated from only five instances spanning $N = 51$ to $N = 1002$, so the reported exponents are point estimates without formal uncertainty intervals; while the log–log fits are tight ($R^2 \geq 0.988$), extrapolation far beyond 1,000 cities should be treated cautiously, particularly because the constant factors in a pure Python implementation differ from those of a compiled neighbor-list code. All runs use a single deterministic starting city and a single first-improvement descent, so the reported gaps are point estimates rather than distributions over random restarts. Finally, the symmetric-matrix assumption of Section 2.2 confines the $\mathcal{O}(1)$ move evaluation—and hence the reported 2-opt runtimes—to symmetric Euclidean instances; asymmetric problems would require a different delta computation and, most likely, a different neighborhood. Within these bounds, the study delivers a clear and reproducible picture: a from-scratch NN + 2-opt pipeline attains single-digit-percent optimality gaps across a wide size range, with construction cost negligible and near-linear, and refinement cost super-cubic and dominant—a textbook illustration of the quality–runtime trade-off that governs heuristic combinatorial optimization.

Reproducibility

All components—the TSPLIB parser, the `EUC_2D` metric of Equation (1), the Nearest-Neighbor construction of Algorithm 1, and the first-improvement 2-opt of Algorithm 2—were implemented from first principles in Python with NumPy; no external TSP solver or metaheuristic library was used at any stage. The per-instance metrics of Table 1, the power-law fits of Table 2, and the complete optimized tours are provided as machine-readable JSON alongside this report (`benchmark_results.json`, `scaling_fit.json`, and `optimized_tours.json`). Every reported tour was verified to be a valid Hamiltonian cycle and its length independently recomputed.

References

[1] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ, USA, 2006. ISBN 9780691129938.

[2] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998. doi: 10.1145/290179.290180.

[3] Jillian Beardwood, John H. Halton, and John M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959. doi: 10.1017/S0305004100034095.

[4] Jon L. Bentley. Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411, 1992. doi: 10.1287/ijoc.4.4.387.

[5] Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, USA, 1976. Reprinted in Operations Research Forum, 3, 20 (2022).

[6] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958. doi: 10.1287/opre.6.6.791.

[7] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014. doi: 10.1007/s00453-013-9801-4.

[8] Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956. doi: 10.1287/opre.4.1.61.

[9] Michael L. Fredman, David S. Johnson, Lyle A. McGeoch, and Gretchen Ostheimer. Data structures for traveling salesmen. *Journal of Algorithms*, 18(3):432–479, 1995. doi: 10.1006/jagm.1995.1018.

[10] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970. doi: 10.1287/opre.18.6.1138.

[11] Keld Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi: 10.1016/S0377-2217(99)00284-2.

[12] David S. Johnson and Lyle A. McGeoch. The traveling salesman problem: A case study in local optimization. In Emile H. L. Aarts and Jan Karel Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK, 1997.

[13] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. A (slightly) improved approximation algorithm for metric TSP. *Journal of the ACM*, 2021. doi: 10.1145/3406325.3451009. Conference version in STOC 2021, pp. 32–45.

[14] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

[15] Shen Lin. Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10):2245–2269, 1965. doi: 10.1002/j.1538-7305.1965.tb04146.x.

[16] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi: 10.1287/opre.21.2.498.

[17] Ilhan Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Northwestern University, Evanston, IL, USA, 1976.

[18] Christos H. Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. doi: 10.1016/0304-3975(77)90012-3.

[19] Gerhard Reinelt. TSPLIB—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. doi: 10.1287/ijoc.3.4.376.

[20] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977. doi: 10.1137/0206041.