

Scaling a Geometric Construction-Heuristic Framework for the Euclidean TSP to $n = 20,000$: Engineering Notes and Empirical Findings

Abhijit Bhattacharjee

Independent Researcher · Uncle Droid
tsp.uncledroid.app · largetsp.uncledroid.app

Third in a series · September 2026

Abstract

The parent algorithm evaluated in this work — Abhijit's Algorithm, introduced in the first paper of this series [1] — is a family of geometric construction heuristics for the Euclidean Traveling Salesman Problem, originally developed and validated at scales up to a few hundred cities. This paper documents a subsequent investigation into its behaviour at much larger scale, up to $n = 20,000$, carried out through a purpose-built research service that collected several thousand independent solves across controlled and randomised instance distributions. The investigation has two distinct outputs. First, porting the original strategies to this scale required a series of specific algorithmic and systems interventions — candidate-list restriction, incremental data structures to replace repeated full rescans, a safety-net mechanism for a class of error the restricted search cannot see by construction, and wall-clock budget management tuned against real, measured run durations rather than assumed worst cases. Each was motivated by a concrete failure observed at scale, and each is documented here with the failure, the diagnosis, and the fix, since this trajectory is likely to recur for others adapting $O(n)$ or $O(n \log n)$ constructive heuristics beyond toy sizes. Second, the resulting large-scale corpus — 212 controlled, same-instance, four-strategy comparisons at sizes from $n = 200$ to $n = 20,000$, spanning uniform, clustered, and disk-uniform point distributions — supports several empirical findings: a dominant strategy at scale that was not dominant at the small scales originally studied; a precisely located crossover point between two strategies as a function of n ; a refinement of that crossover showing distribution-dependence; and two clean negative results, reported in the interest of completeness rather than omitted for not confirming expectations.

Part I: Engineering — Making Construction Heuristics Work at $n = 20,000$

1. The core obstacle: everything in the original algorithm assumes $O(n^2)$ is affordable

The original algorithm's seeding strategies — hull-based insertion (OTR), shortest-edge-first chain construction (SSF), and nearest-neighbour growth (SPN, and its bidirectional variant SPNQ) — were all originally specified against a full distance matrix and, in several cases, exhaustive per-step scans over all remaining unplaced points. At $n = 100$ this is unremarkable; at $n = 20,000$ a full distance matrix alone is 400 million entries, and any step that scans “all remaining points” turns an $O(n)$ -step construction into $O(n^2)$ overall. This is the single problem every subsequent fix in this section is a response to in one form or another.

The general solution adopted throughout was candidate-list restriction: for each point, precompute a bounded set of geometrically plausible neighbours — its Delaunay triangulation neighbours [5], plus a small number of additional nearest neighbours (in practice, five, yielding an observed average of roughly 7.5 candidates per point across all distributions tested) — and require every subsequent operation to only ever consider edges between a point and its own candidate list, never a full scan. This turns each step from $O(n)$ into $O(k)$ where k is small and roughly constant. It is not, by itself, sufficient — several of the fixes below are specifically about what candidate-list restriction gets wrong by construction, not just about performance.

2. Porting OTR: the straightforward case

Hull-based insertion ported cleanly: build the convex hull, then insert remaining points via best-improvement insertion restricted to each point's candidate list. This required no further scaling intervention — it is, empirically, also the fastest of the four strategies at every scale tested, a fact later shown to be mechanistically connected to why it is also the weakest on final quality (Part II, Finding 6).

3. Porting SSF: two $O(n^2)$ bugs found by testing at scale, not assumed from small-scale correctness

SSF's original specification builds a tour by repeatedly taking the globally shortest available edge, Kruskal-style, subject to a degree- ≤ 2 constraint per point to keep the result a single chain. Two independent bugs surfaced only when tested at $n = 20,000$, despite the candidate-restricted port being verified quality-equivalent to the original algorithm at small and moderate n first.

Bug 1 — rebuilding auxiliary dictionaries from scratch on every insertion. The first working port rebuilt its bookkeeping structures from scratch on every single insertion — $O(n)$ work per insertion, $O(n^2)$ overall. This was invisible at a few hundred points but measured at 35 seconds for seeding alone at $n = 20,000$. The fix rewrote this using a neighbour-graph structure supporting $O(1)$ splice insertions, bringing the same step down to 4.6 seconds — roughly a $7.5\times$ improvement from a data-structure change alone, with zero change to the resulting tour.

Bug 2 — fallback stitching that rescans everything, every time. SSF's degree constraint occasionally leaves the graph disconnected into loose fragments requiring a final stitching pass. The number of fragments needing this fallback was observed to scale sharply with n — 1, 3, 7, 13, 38, and 720 fragments at successive size levels up to $n = 20,000$ — and the original fallback rescanned the entire remaining point set, and searched all pairs within it, on every single stitch. At 720 stitches this reproduced the exact $O(n^2)$ problem candidate restriction was meant to eliminate, and did not finish within 60 seconds at $n = 20,000$. The fix replaced the rescan with an incrementally-maintained set of unstitched point IDs and an $O(\text{remaining})$ nearest-valid-partner scan, bringing this bottleneck to 0.5 seconds. The general lesson — a fallback path that looks rare in small tests can dominate total runtime at scale if its own cost is not independently bounded — recurs in Section 6 in structurally identical form, for a different strategy, discovered independently.

4. The crossing bug: an error class the restricted search cannot see by construction

A separate, more subtle problem surfaced once both strategies above were seeding and refining quickly: solved tours at large n occasionally retained edge crossings through the full refinement process. Diagnosis on three real production instances found that of 11 actual crossings present, zero would ever have been examined by candidate-restricted 2-opt, because it only considers edge pairs whose endpoints are each other's own candidates — and a crossing between two long

edges connecting distant regions of the point cloud, by definition, does not meet that criterion. This is a structural blind spot of the restriction itself, present in every instance, not a rare failure triggered by unlucky data.

The fix, a full unrestricted crossing sweep, deliberately reintroduces an $O(n^2)$ check — but run once, as a safety net after fast candidate-restricted refinement has already converged, rather than on every iteration. If it finds and fixes a crossing, the fast loop re-runs to reconverge and the sweep repeats, bounded to at most ten outer rounds, until clean. This reduced observed crossings to zero across every tested instance, with a secondary benefit of 0.04–1.76% tour-length improvement from the crossings genuinely being missed. The pairing of a fast, structurally-limited method with an infrequent, expensive, structurally-complete correctness check recurs in Section 6's design and is, in retrospect, the closest thing this investigation has to a reusable principle.

5. Wall-clock budgets, and a boundary bug inside the mechanism meant to prevent exactly this kind of bug

Once individual solves at $n = 15,000$ – $20,000$ could take minutes, an unbounded refinement loop became operationally unacceptable for a scheduled batch process sharing a fixed nightly compute window. A wall-clock budget parameter was added, checked inside both the inner refinement loop and the outer safety-net loop of Section 4; the batch scheduler discards any non-converged result rather than logging it, on the principle that a genuinely incomplete tour is not useful research data.

This budget was tuned twice against real, measured data. It began at 600 seconds; production data then showed a genuine $n = 20,000$ convergence at 806 seconds, meaning the ceiling had been discarding attempts that would have succeeded, and it was raised to 900 seconds. A more interesting failure then appeared: a specific solve reported 806 seconds of measured runtime against what should have been a hard 900-second ceiling — structurally impossible on its face. Direct investigation found the gap: the safety-net sweep of Section 4, itself a real $O(n^2)$ cost (≈ 3.2 seconds at $n = 20,000$), sat between the inner loop's deadline check and the decision to begin another outer round, and was not itself deadline-checked at either boundary. A controlled test with a deliberately tight 7-second deadline, positioned to land inside an in-progress sweep, confirmed the fix bounded the overshoot to the duration of one sweep (0.111 seconds observed) rather than allowing a further, unprotected round. The general lesson: a deadline mechanism is only as complete as its enumeration of every code path capable of taking non-trivial time between checks — a safety-net added to fix one correctness problem can silently undermine a different mechanism's guarantee if it is not itself included in that mechanism's accounting.

The budget was raised a further time, to 1,500 seconds, once four-strategy same-instance comparisons (Section 6, Part II) made it possible to observe that SPN was consistently the slowest of the four strategies, and to project an empirical scaling ratio from its own measured $n = 15,000$ behaviour ($1.72\times$, $n = 15,000$ to $n = 20,000$) onto its slowest observed $n = 15,000$ sample — predicting a worst case near 1,375 seconds, which the 900-second budget could not accommodate. Where possible, a resource budget is best sized from an extrapolation grounded in the specific slowest real behaviour observed for the specific workload, not from a round number or the easiest case.

6. Porting SPN and SPNQ: the same fallback-cost lesson, in a worse form

Extending the framework to nearest-neighbour chain growth (SPN) and its bidirectional variant (SPNQ) followed the established candidate-restriction pattern directly, with one necessary addition: as a growing chain consumes more of the point set, a point's own candidate list

becomes increasingly likely to be entirely already visited, requiring a fallback search for the true nearest remaining point. For SPN this fallback was infrequent enough not to matter. For SPNQ — which grows from both ends simultaneously — instrumented measurement at $n = 20,000$ found the fallback firing on 19,092 of 40,000 growth-step calls, essentially one in two, because both simultaneously-advancing chain ends compete for the same shrinking local neighbourhood and exhaust it far faster than single-direction growth does. The cumulative cost — over 170 million point comparisons, a 12.86-second seed time at $n = 20,000$ — was the same class of bug as Section 3's SSF fallback, independently rediscovered in a structurally different strategy.

The fix generalises the design principle of Section 4: replace the linear fallback scan with a spatial index (a KD-tree) over the currently-remaining points, rebuilt lazily only once a meaningful fraction of previously-indexed points have since been consumed, with stale query hits skipped and the query radius widened before a full rebuild. This reduced SPNQ's seed time at $n = 20,000$ from 12.86 to 0.42 seconds — roughly $30\times$ — while producing output identical to the pre-fix version on every regression test, confirming the change was purely a performance fix.

7. A documented negative engineering result: MST-based construction

Beyond porting the original named strategies, one genuinely new construction method was designed, implemented, tested, and set aside within this investigation, reported here because a negative result obtained with the same rigour as a positive one is informative, not merely a dead end.

The idea, termed MSTD (“MST-Doubled”), was motivated by a specific structural observation about SSF: its degree- ≤ 2 constraint sometimes forces it to skip the true cheapest available edge purely because one endpoint already has degree 2. Since the exact minimum spanning tree was already available essentially for free — the Delaunay triangulation is provably guaranteed to contain the true MST as a subgraph [5] — MSTD instead builds the unconstrained MST directly, with no degree compromise during the build itself, and converts it to a tour via depth-first preorder traversal, the standard direct implementation of the classical double-tree approximation technique [6]. It was fully implemented, verified structurally correct, confirmed to never require its own disconnection fallback in practice, and confirmed safe at $n = 20,000$.

Its quality, however, was consistently disappointing: across five diverse test instances, MSTD ranked fourth of five available strategies every time — ahead only of OTR. The most likely mechanism: DFS-preorder's “shortcut” jumps between sibling subtrees occur in whatever order the tree's adjacency happens to produce, with no relationship to geometric distance, and can be long, arbitrary jumps that SSF's construction — which only ever selects genuinely short, locally cheap edges, even under a degree-constraint compromise — never produces. A more targeted follow-up, MSTG (“MST-Greedy”), always advancing to whichever tree-eligible node is geometrically nearest the current position rather than following blind DFS order, was built specifically to test this diagnosis, using the same KD-tree frontier design as Section 6. It performed worse on both dimensions it was meant to improve: final quality below MSTD's (427,305.6 versus 423,366.3 on an identical $n = 3,000$ instance) and refinement cost dramatically higher (1,239 passes versus 462). Greedy node-by-node selection, unlike DFS, does not commit to finishing one coherent branch before moving on; it readily abandons a partially-explored subtree to chase a slightly closer point elsewhere, producing a more fragmented, backtracking-heavy structure that costs more to untangle than the arbitrary-but-locally-coherent jumps it was designed to fix.

8. Operational scheduling: batch windows, guaranteed comparisons, and discard discipline

Converting a working large-N solver into a sustainable nightly research pipeline running unattended within a fixed compute window required several further systems-level decisions. Any run that does not converge within its budget is discarded, not logged — a genuinely incomplete tour would silently contaminate later analysis. Once four-strategy same-instance comparisons became the primary unit of research value, a guaranteed-comparison mechanism was added for the largest, slowest-converging size tier: the first attempt each night checks the database — not an in-memory flag, which would not survive each scheduled invocation being a fresh, stateless process — for whether a qualifying comparison has already landed since the window opened, and if not, forces one at a deliberately chosen, reliably-converging size rather than leaving it to chance. This check re-evaluates on every subsequent tick, so a first attempt that itself fails to converge does not silently leave the night without a guaranteed comparison; the mechanism retries until one succeeds. A related, smaller lesson: the numeric threshold defining what counts toward this guarantee was, at one point, found to have drifted from the constant defining the guarantee itself — an inconsistency caught only by testing the mechanism end-to-end against a real database, not by code inspection alone.

9. Strategies deliberately not ported, and why

Of the original algorithm's eight named strategies, four (OTR, SSF, SPN, SPNQ) were ported to the large-N framework; the remaining four were not, for reasons specific to each. SPX and SPXQ (farthest-neighbour growth) have a genuine structural tension with the candidate-list architecture underlying this entire investigation: their defining criterion is to connect to the farthest unvisited point, while the candidate lists this framework depends on for tractability are, by construction, lists of nearest neighbours — restricting a farthest-neighbour search to a nearest-neighbour candidate structure would not faithfully implement the strategy at all. SPEN and SPENQ (exhaustive multi-start variants, keeping the best result across every possible starting city) are fundamentally incompatible with scale in their original form: trying all n possible starts at $n = 20,000$ is a straightforward $O(n)$ multiplication of an already-substantial per-start cost, not a performance problem addressable by the techniques above, and any scalable version would need to sample a small subset of starts rather than exhaust them — a genuine redesign, not an optimisation.

Part II: Empirical Findings

Methodology

The findings below are drawn from a corpus collected via an automated nightly batch process, spanning both randomly-scheduled single-strategy runs and, of primary interest here, paired comparisons: all four large-N strategies run on the identical generated instance, allowing direct same-instance comparison uncontaminated by instance-to-instance variance. At the time of writing this yields 212 complete four-strategy comparisons, spanning $n = 200$ to $n = 20,000$ across three distribution families — uniform-random, clustered (Gaussian mixtures with randomised cluster count and spread), and uniform-in-disk. All comparisons report tour length as percentage above the instance's minimum spanning tree lower bound, a metric well-defined and near-instantaneous to compute at every scale, in place of the true optimal tour length, which is intractable to obtain at these scales for arbitrary instances.

Finding 1: SSF dominates the large-n regime, decisively and consistently

Across all 212 four-strategy comparisons, SSF was the best of the four strategies in 182 cases (85.8%). SPNQ was best in 18 cases (8.5%), SPN in 8 cases (3.8%), and OTR in 4 cases (1.9%). This dominance is stable across the full range of sizes and all three distribution families, and the margin did not narrow as data accumulated — an early subset of 12 comparisons showed a similar 67% SSF win rate, and the ranking sharpened toward its current figure as the sample grew, rather than reverting toward parity.

Table 1: Best-strategy outcome across all 212 same-instance, four-strategy comparisons.

Strategy	Wins	Share
SSF	182	85.8%
SPNQ	18	8.5%
SPN	8	3.8%
OTR	4	1.9%

Finding 2: a precisely located crossover point explains a pattern in the algorithm's earliest, small-scale benchmarks

SSF's dominance at $n \geq 100$ –200 stands in contrast to the algorithm's original small-scale validation [1], where standard TSPLIB benchmark instances (berlin52, $n = 52$; st70, $n = 70$; eil76, $n = 76$; kroA100, $n = 100$) had shown a mixed picture — OTR outperforming SSF on three of the four. A controlled experiment, using the original, unrestricted implementation of both strategies with twelve independently-seeded trials at each of ten instance sizes from $n = 10$ to $n = 300$, located the transition precisely: OTR wins clearly and consistently for $n = 10$ –30, the two strategies are close to indistinguishable around $n = 50$, and SSF is dominant and increasingly so from roughly $n = 75$ –100 onward. The four historical benchmark sizes sit almost exactly within this transition band — the “mixed” result obtained from those four instances was not an inconsistency in the original validation, but an accurate reflection of having sampled directly in the zone where the two strategies' relative advantage is genuinely still changing sign.

Finding 3: the crossover is not size-uniform — clustered instances sustain OTR's small-n competitiveness longer

A closer look at the rare instances in the large-n corpus where OTR either won or came within a fraction of a percentage point of winning shows a further, more specific pattern: these cases cluster disproportionately at small n (200–500) and the clustered distribution family specifically. Two of OTR's four outright wins in the full corpus were both $n = 200$, clustered instances, each by a substantial margin (≈ 10 percentage points above the corresponding SSF result); several additional $n = 500$ clustered comparisons show OTR and SSF essentially tied, the closest observed margin being 0.06 percentage points. No comparably close result appears among uniform or disk-distributed instances at the same sizes. This refines Finding 2 from a single scalar crossover point in n to a genuinely two-dimensional relationship between size and distribution shape — plausibly connected to a convex hull capturing comparatively more of a clustered instance's total structure than it does for a uniform or disk-distributed instance of the same size, offered as a plausible account of a robustly observed pattern rather than an independently verified mechanism.

Finding 4: SPN and SPNQ are legitimate minority strategies with no strong distribution-specific niche found to date

SPN and SPNQ together account for 12.3% of best-strategy outcomes in the full corpus — meaningfully more than OTR, meaningfully less than SSF, and not concentrated in any distribution family identified so far. An early, small-sample observation suggesting SPNQ performed specifically better on disk-distributed instances did not survive a larger subsequent sample, and is reported here as a retracted hypothesis rather than omitted, on the view that documenting a plausible-looking early pattern that did not hold up is itself useful to anyone extending this work.

Finding 5 (negative result): a geometrically well-motivated hypothesis about disk-shaped instances is not supported

A specific, mathematically grounded hypothesis was tested and not confirmed: since a disk's boundary is known, via Rényi–Sulanke asymptotic results [7], to accumulate convex-hull vertices at a faster rate, $O(n^{1/3})$, than a square's, $O(\log n)$, a hull-based strategy such as OTR was predicted to retain relatively more of its advantage on disk-distributed instances than on comparable uniform, square-domain instances. This was tested twice, independently: once via the large- n production corpus at $n = 1,500$ – $7,500$, where SSF's margin over OTR on disk instances was, if anything, slightly larger than on uniform instances; and once via a dedicated controlled small- n sweep spanning $n = 10$ – 50 , constructed to avoid confounding the geometric effect with the much larger, separately-established effect of instance size, where four of five tested cells showed the opposite of the predicted direction. The underlying geometric mathematics is not in question; what is not supported is that it translates into a measurable effect on final heuristic quality at any scale tested here.

Finding 6: a consistent quality-for-time tradeoff across strategies, with OTR anchoring the fast-but-weak end

Direct timing comparison across paired same-instance runs shows OTR consistently fastest, with SSF, SPN, and SPNQ costing on average 22%, 31%, and 26% more wall-clock time respectively in one representative twelve-comparison sample at large n . This is plausibly not independent of Finding 1: a hull-based seed that captures comparatively little of an instance's true structure settles into its own, weaker, local optimum with fewer refinement passes required to get there, while the other three strategies' stronger seeds require correspondingly more refinement work to reach their better final results. SPN is the most consistently expensive of the three non-OTR strategies at the largest sizes tested, concrete enough that it directly motivated the wall-clock budget increase in Part I, Section 5.

Finding 7 (negative/null result): static geometric instance features do not predict convergence difficulty for the strongest strategies

Two geometric summary statistics — elongation (point-cloud aspect ratio) and density coefficient of variation (local density unevenness) — were tested for correlation against per-point convergence time across roughly 90 large- n production runs, stratified by strategy. For the three best-performing strategies (SSF, SPN, SPNQ), both correlations were weak in every case ($|r| < 0.2$). A modest exception was found for OTR specifically, where density-of-variation showed a weak-to-moderate positive correlation ($r \approx 0.32$) with convergence time, plausibly connected to OTR's hull-based seed capturing comparatively less structure on instances with highly uneven local density — a secondary effect on the weakest strategy, not a general predictive tool. This extends an earlier, separately-obtained null finding that the same two

features fail to predict which strategy will win on a given instance; together, these results suggest a genuinely predictive measure of instance difficulty, if one exists, is unlikely to be found among simple static geometric summary statistics of the raw point set.

Finding 8: positioning against a canonical exact/near-exact solver

A detailed comparison against Concorde [2], incorporating Lin-Kernighan local search [4], on one representative $n = 1,000$ instance situates this family of heuristics within the broader landscape. Concorde's Lin-Kernighan implementation reached within 0.04–0.32% of the proven optimal tour; this investigation's best strategy (SSF, post the crossing-bug fix of Part I, Section 4) reached 4.08% above optimal on the same instance — a real, non-trivial gap. Concorde's simpler constructive heuristics, run without local search refinement, fared considerably worse on the same instance (15.97% above optimal for greedy edge construction, 27.92% for nearest-neighbour), indicating the gap to Lin-Kernighan reflects that method's specific, well-established strength rather than merely “any real solver beats any heuristic.” The comparison is offered not to claim competitiveness with Lin-Kernighan on raw quality, which this work does not achieve, but to characterise a deliberate tradeoff: this framework's candidate-restricted 2-opt and single-relocation refinement requires no linear-programming solver, no branch-and-cut infrastructure, and no Euclidean-specific cut-generation — requirements structurally more compatible with constrained-hardware settings than a full Lin-Kernighan implementation, whose own algorithmic core requires comparably sophisticated tour-representation structures regardless of whether the underlying cost structure is Euclidean or general.

Finding 9: independent, real-world validation of a newly added strategy

Beyond the controlled research corpus, the live public interface to this work recorded an unsolicited real visitor session in which SPNQ — a strategy added during this investigation, absent from the algorithm's original small-scale release [1] — was used on the classical TSPLIB instance berlin52 ($n = 52$, published optimal 7,542), returning a result of 7,544.37, or 0.03% above the published optimum. A single external data point cannot substitute for the controlled corpus above, but it is offered as independent, informal confirmation that the newly added strategies function correctly and competitively outside the research pipeline, on an instance and by a visitor entirely unconnected to this investigation's own testing.

Discussion and Limitations

The findings above are drawn from generated instances rather than real-world logistics or routing data, and the above-MST-bound metric, while tractable at every scale tested, is a lower bound rather than the true optimal tour length; the true optimality gap for any specific large instance in this corpus remains formally unknown, though the single available Concorde comparison (Finding 8) provides at least one calibration point. The mechanistic explanations offered for several findings — the hull-capture account of Finding 3, the seed-strength account of Finding 6, the density-unevenness account of Finding 7's OTR exception — are plausible and consistent with the observed data but not independently verified against a separate, targeted test in every case, and should be read as the most likely account currently available rather than an established mechanism. The negative results in Findings 4 and 5, and the MST-based methods of Part I, Section 7, are reported with the same care as the positive findings specifically because a negative result obtained through the same rigour as a positive one has genuine value, particularly for anyone considering the same hypotheses independently.

Future Work

The refinement-phase pass count required for full convergence at $n = 20,000$ (on the order of 1,000+ passes in some observed cases) remains an architectural bottleneck not addressed by any fix in Part I; don't-look bits, a standard technique in large-scale local search TSP implementations [8], were identified early in this investigation as likely relevant but unimplemented. The SPX/SPXQ and SPEN/SPENQ strategies excluded in Part I, Section 9, remain candidates for a genuinely redesigned, rather than directly ported, large- N treatment, particularly a sampled version of SPEN/SPENQ's multi-start approach. Finding 7's null result invites a search for convergence-difficulty predictors grounded in early refinement-process dynamics rather than static point-set geometry. The hardware-suitability argument in Finding 8 remains a reasoned architectural comparison rather than a measured one; an implementation and benchmark on constrained hardware would convert this from an argument into a result.

Conclusion

Adapting a family of construction heuristics originally developed and validated at $n \leq 100$ to operate correctly and efficiently at $n = 20,000$ required a specific, recurring pattern of engineering work: identify an operation implicitly assuming an affordable $O(n)$ or $O(n^2)$ cost, replace it with a candidate-restricted or incrementally-maintained equivalent, and, critically, pair every such restriction with an explicit, separately-tested accounting for what the restriction can no longer see or guarantee by construction. Every fix in Part I was motivated by a specific, measured failure at scale rather than anticipated from first principles — a trajectory this paper suggests is likely typical rather than particular to this codebase. The resulting corpus both confirms the parent algorithm's continued usefulness at scale, with a dominant strategy holding up under a very large sample, and meaningfully revises the small-scale picture that preceded it, locating and explaining a crossover effect the original validation had, in retrospect, sampled directly within without identifying. The negative results reported alongside these — a geometric hypothesis that does not hold up, and a theoretically well-motivated construction method that underperforms simpler greedy alternatives in practice — are included deliberately, in the view that a complete record of an empirical investigation includes what did not work as well as what did.

References

- [1] Bhattacharjee, A. Abhijit's Algorithm: A Geometric, Best-Improvement Heuristic for the Travelling Salesman Problem. Preprint, tsp.uncledroid.app/paper.pdf, 2026.
- [2] Bhattacharjee, A. Wind-Directed Solution Diversity in Abhijit's TSP Algorithm: A Novel Exploration Heuristic via Directional Cost Bias. Research Note, tsp.uncledroid.app/paper2.pdf, 2026.
- [3] Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). The Traveling Salesman Problem: A Computational Study. Princeton University Press.
- [4] Helsgaun, K. (2000). An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1), 106-130.
- [5] Shamos, M. I., & Hoey, D. (1975). Closest-point problems. *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, 151-162.
- [6] Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University.

- [7] Rényi, A., & Sulanke, R. (1963). Über die konvexe Hülle von n zufällig gewählten Punkten. Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete, 2(1), 75-84.
- [8] Johnson, D. S., & McGeoch, L. A. (1997). The traveling salesman problem: A case study in local optimization. In Local Search in Combinatorial Optimization, 215-310.