Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Where do the gains come from? A component analysis of a spatial-decomposition memetic algorithm for large geometric traveling salesman problems

Minh-Khanh Vu

Minh-Khanh.Vu@etu.univ-grenoble-alpes.fr

Université Grenoble Alpes

Research Article

Keywords:

Posted Date: October 28th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11379553/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Where do the gains come from? A component analysis of a spatial-decomposition memetic algorithm for large geometric traveling salesman problems

Minh-Khanh Vu[1*]

[1*]Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, 38000, Grenoble, France.

Corresponding author(s). E-mail(s):
Minh-Khanh.Vu@etu.univ-grenoble-alpes.fr;

Abstract

Geometric traveling salesman problems (TSP) with thousands to hundreds of thousands of points arise in printed circuit board (PCB) drilling and VLSI layouts. We present a hierarchical spatial-decomposition memetic algorithm (HSD-MA) that combines Morton and nearest-neighbor seeding, a Lamarckian evolutionary core with order crossover, a three-tier local search (neighbor-list 2-opt, 2.5-opt node shift and segment-insertion 3-opt over 1-hop and 2-hop Delaunay candidates) and, for more than 5,000 points, a k-means decomposition whose clusters are solved in parallel. Distances are computed on the fly, so memory grows linearly with the instance size. We then ask which components matter, using controlled experiments in which all methods share the same compiled local search kernels. On ten instances with 30 runs each, HSD-MA is significantly better than genetic, particle swarm and ant colony hybrids that use 2-opt only, but not better than the same frameworks equipped with the three-tier local search. An iterated local search without population, given the same time per run, finds significantly shorter tours on the four instances with 1,173 or more points and is worse only on the two smallest; with 16 times more generations the difference shrinks but does not reverse. Given the same time, the iterated local search is also better than the decomposition branch on instances with 7,397 and 33,810 points; the decomposition pays off only on the largest instance tested, with 104,814 points. Seeding and the kick have no significant effect, and nearest-neighbor candidates work as well as Delaunay candidates, so the gains over 2-opt-based memetic algorithms come from the local search, not from the population. HSD-MA solves

all 102 Waterloo VLSI instances, up to 744,710 points, with a mean gap of **5.88%** on the 25 largest and a peak working set of 0.9 to 1.2 GiB on the largest. LKH-3, run on the same machine, finds much better tours in every configuration we tested, and GA-EAX does so on the small instances given a few seconds; the only advantage of HSD-MA is that it needs a third to a fifth of the wall-clock time of the quickest LKH-3 configuration on large instances. In a simulated drilling case study, HSD-MA shortens the cycle time by less than **1%** relative to a nearest-neighbor tour improved by 2-opt.

Keywords: Traveling salesman problem, Memetic algorithm, Iterated local search, Spatial decomposition, Component analysis, PCB drilling

1 Introduction

The traveling salesman problem (TSP) remains a central benchmark in discrete optimization and operations research [1, 2]. Geometric versions of the symmetric TSP arise in electronic manufacturing, notably in sequencing the holes drilled into printed circuit boards (PCBs) [3–6]. The drilling and programmed-logic-array instances of TSPLIB range from 198 (`d198`) to 85,900 (`pla85900`) points [7], and the `pcb442` drilling instance originates from an industrial case study [3], in which optimizing the sequence reduced the positioning moves of the plotting and drilling machines by about 10% to almost 90%. In multi-hole drilling more generally, non-cutting time, comprising repositioning and tool switching, can take up to 70% of the total process time according to an earlier survey quoted by Dewil et al. [8].

Exact branch-and-cut solvers such as Concorde [1] guarantee optimality, but their running time grows rapidly with instance size. Local improvement heuristics, notably 2-opt [9, 10], Or-opt [11] and Lin–Kernighan variants [12–15], are fast; well-implemented neighbor-list 2-opt ends about 5% above the Held–Karp bound on uniform random instances and roughly 10% above it on clustered instances [16, 17]. LKH [13, 14] produces near-best tours; without its partitioning options, its author reports that the original implementation becomes too expensive beyond about 100,000 nodes [18], whereas LKH-2 and LKH-3 provide tour and point partitioning schemes for very large instances [18, 19]. The strongest population-based methods for the TSP are memetic: genetic local search [20], the edge assembly crossover (EAX) genetic algorithm [21–24] and partition crossovers [25–27]. Evolutionary search without local improvement, in contrast, produces poor tours on the TSP; local optimization appears essential to obtain competitive results [16]. Neural combinatorial optimization based on pointer networks [28], attention models [29, 30] and learned divide-and-conquer schemes [31–34] provides fast inference once trained, but such models generalize poorly beyond their training sizes [35], and training on tens of thousands of points remains costly [36, 37].

This paper studies a memetic algorithm for large geometric instances from PCB drilling and VLSI layouts, with three practical goals: linear memory, so that instances with hundreds of thousands of points can be solved on a laptop; short runtimes of seconds to a few minutes; and a transparent evaluation against both simple baselines

and the established solvers LKH and GA-EAX. Storing an $N \times N$ double-precision distance matrix, as a straightforward implementation would, needs about 9.1 GB (8.5 GiB) for `pla33810` and about 4.4 TB for the largest Waterloo VLSI instance ($N = 744{,}710$); established solvers avoid this by computing distances from coordinates and keeping sparse candidate sets [1, 13], and HSD-MA follows the same principle.

1.1 Contributions

HSD-MA does not introduce a new local search move or a new decomposition principle. Its ingredients, namely Delaunay candidate sets [4, 38, 39], space-filling-curve tours [40, 41], k-means partitioning [19] and double-bridge kicks [42], are known, and several of them are options of LKH [19]. The contributions are an integration of these ingredients into a lightweight memetic algorithm and an empirical analysis of which of them matter:

1. A memetic algorithm whose three local search tiers (neighbor-list 2-opt, 2.5-opt node shift and segment-insertion 3-opt) draw candidates from 1-hop and 2-hop Delaunay neighbors truncated to 15 per vertex, combined with Morton and corner nearest-neighbor seeding and, for $N > 5{,}000$, a k-means decomposition whose clusters are solved in parallel; distances are computed on the fly, so memory grows linearly with N.
2. A controlled comparison in which all methods share the same compiled local search kernels, including two evolutionary frameworks with the full three-tier local search and an iterated local search without population run for the same time. It shows that the improvement over 2-opt-based memetic algorithms comes from the local search, that at the budgets studied the population does not outperform the iterated local search on instances with more than about 1,000 points, and that the decomposition pays off only on the largest instance tested; ablations of the candidate set, seeding, crossover and kick, a parameter study and diversity measurements support this conclusion.
3. A comparison with LKH-3 and GA-EAX on the same machine, results on all 102 Waterloo VLSI instances, and a simulated drilling case study, which together delimit where such a lightweight solver is useful: it returns tours several percent above the best known ones in a third to a fifth of the wall-clock time of the quickest LKH-3 configuration we tested, but LKH-3 and GA-EAX find much better tours.

2 Related work

Memetic algorithms, introduced by Moscato [43] and formalized by Radcliffe and Surry [44], combine population-based evolutionary search with individual improvement by local search. Under Baldwinian learning, local search is used only to evaluate fitness; under Lamarckian learning, the improved tour replaces the genotype. Lamarckian inheritance typically speeds up convergence [45], but powerful local search written back into the genotype can cause a rapid loss of diversity and premature convergence to a local optimum [45, 46]. For the TSP, Freisleben and Merz [20] combined a distance-preserving crossover with local search; EAX-based genetic algorithms [21, 22] reach state-of-the-art quality and scale, in parallel, to 100,000-city instances [23]; Tsai et al.

[24] combined EAX with Lin–Kernighan for large instances; and partition crossovers recombine local optima while preserving their common edges [25–27]. Recent memetic work also addresses the diversity of the population explicitly [47]. In PCB and multi-hole drilling, path optimization has been approached with particle swarm optimization [5, 6], hybrid genetic algorithms [48], the shuffled frog leaping algorithm [49], cuckoo search [50] and adaptive stochastic heuristics [51], mostly on instances of a few hundred holes; reviews are given by Dewil et al. [8] and Zainal Abidin et al. [52], and a recent drilling machine with integrated path optimization is described in [53].

Spatial decomposition scales heuristics to large instances. Karp [54] partitioned the plane for probabilistic analysis; Valenzuela and Jones [55] used evolutionary divide and conquer; Ding et al. [56] clustered large instances and solved the resulting clustered problem with a two-level genetic algorithm; Mulder and Wunsch [57] solved a million-city instance by clustering; POPMUSIC [58] and its linearithmic variant tested on billions of cities [59] optimize parts of a solution independently; and tour merging combines several complete tours by a branch decomposition [60]. LKH-3 offers partitioning into subproblems by k-means, Karp, Delaunay or space-filling-curve schemes, optionally re-optimizing the borders between subproblems [19]. Learning-based divide-and-conquer methods partition large instances and refine the result with learned or classical local search [31–34]. HSD-MA partitions the points rather than an existing tour and does not re-optimize cluster borders before the final global local search; Section 7 quantifies the resulting loss. Rocki and Suda [61] parallelized 2-opt and 3-opt on GPUs.

Regarding candidate sets, Bentley [10] showed that restricting k-opt moves to k-nearest neighbors reduces the search cost with little loss of solution quality, and Johnson and McGeoch [16, 17] analyzed neighbor lists and don't-look bits in detail. Helsgaun [13] introduced α-nearness candidates in LKH, which also offers Delaunay, nearest-neighbor and quadrant candidate sets [19]. Reinelt [4, 38] and Jünger, Reinelt and Rinaldi [39] used the Delaunay graph to derive candidate sets for fast heuristics on large geometric and PCB instances, noting that the Delaunay graph alone can be too small and is best complemented by further edges [39]. The Delaunay triangulation [62] contains the Euclidean minimum spanning tree and the nearest-neighbor graph [63]; nearly all edges of optimal TSPLIB tours (97% to 100%) are Delaunay edges [64], and on TSPLIB instances with fewer than 1,500 cities, restricting the tour to Delaunay edges loses only about 0.3% on average (0.1% when shortcuts are allowed) [65]. Delaunay-based neighborhoods have also been used in variable neighborhood search for large colored TSPs [66]. Space-filling-curve tours, which sort the points along a curve in $\mathcal{O}(N \log N)$ time and $\mathcal{O}(N)$ memory, go back to Bartholdi and Platzman [40, 41].

3 Problem formulation

3.1 Graph-theoretic model of geometric TSP

Let $V = \{v_1, v_2, \ldots, v_N\}$ represent N target coordinates in the plane $\mathbb{R}^2$, where each vertex $v_i = (x_i, y_i)$ corresponds to a hole location. The problem is defined on a complete undirected graph $G = (V, E)$, with edge set $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$. A tour is a cyclic permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ of the vertices. The objective is to minimize

the total length

$$\min_{\pi \in \Pi} L(\pi) = \sum_{i=1}^{N-1} d(v_{\pi_i}, v_{\pi_{i+1}}) + d(v_{\pi_N}, v_{\pi_1}), \tag{1}$$

where Π is the set of the $(N-1)!/2$ distinct undirected Hamiltonian cycles and d is the edge length. Following TSPLIB, HSD-MA uses the Euclidean length rounded to the nearest integer (`EUC_2D`); for the two `pla` instances, whose official metric rounds up (`CEIL_2D`), the tours found by HSD-MA are scored with the official metric [7].

3.2 Euclidean length and machine traverse time

Biaxial CNC machines move two independent orthogonal axes simultaneously, so the traverse time between two holes is determined by the slower axis [50]. For constant axis speed this time is proportional to the Chebyshev norm (L_∞), which TSPLIB provides as the `MAX_2D` edge-weight type [7]:

$$d_{L_\infty}(v_i, v_j) = \max\left(|x_i - x_j|, |y_i - y_j|\right), \qquad d_{L_2}(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{2}$$

Proposition 1 (Norm equivalence bounds) *For any two points $v_i, v_j \in \mathbb{R}^2$,*

$$\frac{1}{\sqrt{2}} d_{L_2}(v_i, v_j) \le d_{L_\infty}(v_i, v_j) \le d_{L_2}(v_i, v_j), \tag{3}$$

with equality on the right for axis-parallel moves and on the left for diagonal moves ($|\Delta x| = |\Delta y|$). Consequently, for any tour π, $L_2(\pi)/\sqrt{2} \le L_\infty(\pi) \le L_2(\pi)$, and a tour that is optimal for L_2 has an L_∞ length at most $\sqrt{2}$ times the optimal L_∞ length.

Proof Write $a = |x_i - x_j|$ and $b = |y_i - y_j|$. Then $\max(a,b)^2 \le a^2 + b^2 \le 2\max(a,b)^2$, which gives Eq. (3). Summing over the edges of a tour gives the tour-level bounds. If π_2 is L_2-optimal and π_∞ is L_∞-optimal, then $L_\infty(\pi_2) \le L_2(\pi_2) \le L_2(\pi_\infty) \le \sqrt{2}\, L_\infty(\pi_\infty)$. □

A single edge can thus be up to 29.3% shorter under L_∞ than under L_2. HSD-MA optimizes the Euclidean benchmark metric; Section 6 evaluates the resulting tours under a kinematic time model and tests whether optimizing the time objective directly changes the result.

4 The HSD-MA algorithm

Algorithm 1 outlines the execution flow of HSD-MA.

4.1 Spatial seeding

Random tours are very poor: on 1,000 uniform points they average about 2,150% above the Held–Karp bound [16], and our random initializations started on average 1,295% to 4,460% above the BKS (Table 5). Initializing all individuals with construction heuristics, however, reduces diversity [67]. HSD-MA therefore mixes a few constructed

Algorithm 1: Hierarchical spatial-decomposition memetic algorithm (HSD-MA)

Input: Point coordinates $V = \{v_1, \ldots, v_N\} \subset \mathbb{R}^2$; population size P; generations $T_{\max}$; crossover rate P_c; mutation rate P_m; stagnation threshold τ.
Output: Best tour π_{best} found.

$\mathcal{G}_{\mathcal{DT}} \leftarrow \text{DelaunayCandidates}(V, k_{\max} = 15)$; `// 1-hop + 2-hop, nearest` $k_{\max}$
if $N > 5{,}000$ `// Spatial decomposition for large graphs` **then**
  $K \leftarrow \lceil N/250 \rceil$;
  $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\} \leftarrow \text{MiniBatchKMeans}(V, K, \text{batch_size} = 2048, \text{n_init} = 3)$;
  $\pi_{\text{macro}} \leftarrow \text{MemeticCore}(\{\mu_1, \ldots, \mu_K\}, P, T_{\max})$; `// tour over cluster centroids`
  for $k \leftarrow 1$ **to** K ***in parallel*** **do**
    $\pi_k \leftarrow \text{MemeticCore}(\mathcal{C}_k, 30, 20)$; `// own Delaunay graph of` $\mathcal{C}_k$
  $\pi_{\text{stitched}} \leftarrow \text{BoundaryStitching}(\pi_{\text{macro}}, \{\pi_1, \ldots, \pi_K\})$;
  $\pi_{\text{best}} \leftarrow \text{ThreeTierLocalSearch}(\pi_{\text{stitched}}, \mathcal{G}_{\mathcal{DT}})$; `// global refinement`
else
  $\pi_{\text{best}} \leftarrow \text{MemeticCore}(V, P, T_{\max})$;
return π_{best};

Function $\text{MemeticCore}(V, P, T_{\max})$**:**
  $\mathcal{P}_0 \leftarrow \text{SmartSpatialSeeding}(V, P, \rho = 0.20)$; `// Algorithm 2`
  $\pi^* \leftarrow \arg\min_{\pi \in \mathcal{P}_0} L(\pi)$; $\pi_{\text{best}} \leftarrow \pi^*$; stag $\leftarrow 0$;
  for $t \leftarrow 0$ **to** $T_{\max} - 1$ **do**
    $\mathcal{P}_{t+1} \leftarrow \{\pi^*\}$; `// Elitism`
    while $|\mathcal{P}_{t+1}| < P$ **do**
      $p_1 \leftarrow \text{Tournament}(\mathcal{P}_t, 3)$; $p_2 \leftarrow \text{Tournament}(\mathcal{P}_t, 3)$;
      $c \leftarrow (\text{rand}() < P_c)$? $\text{BitmaskOX1}(p_1, p_2) : p_1$;
      if $rand() < P_m$ **then** $c \leftarrow \text{InversionOrSwapMutation}(c)$;
      $\mathcal{P}_{t+1} \leftarrow \mathcal{P}_{t+1} \cup \{c\}$;
    foreach $c \in \mathcal{P}_{t+1} \setminus \{\pi^*\}$ **do** $c \leftarrow \text{ThreeTierLocalSearch}(c, \mathcal{G}_{\mathcal{DT}})$; `// Algorithm 3, in parallel`
    $\pi_{\text{curr}} \leftarrow \arg\min_{\pi \in \mathcal{P}_{t+1}} L(\pi)$;
    if $L(\pi_{curr}) < L(\pi^*)$ **then**
      $\pi^* \leftarrow \pi_{\text{curr}}$; stag $\leftarrow 0$;
    else
      stag $\leftarrow$ stag $+ 1$;
      if $stag \geq \tau$ **then**
        $\pi^* \leftarrow \text{ThreeTierLocalSearch}(\text{DoubleBridgeKick}(\pi^*), \mathcal{G}_{\mathcal{DT}})$; stag $\leftarrow 0$;
    if $L(\pi^*) < L(\pi_{best})$ **then** $\pi_{\text{best}} \leftarrow \pi^*$;
  return π_{best};

tours with random permutations (Algorithm 2). The first constructed tour follows a space-filling curve, a classical $\mathcal{O}(N \log N)$ heuristic [40, 41]; HSD-MA uses the Morton Z-order curve, which scales the coordinates to 16-bit integers and interleaves their bits into 32-bit codes:

$$\text{code}(x, y) = (\text{part1by1}(y) \ll 1) \mid \text{part1by1}(x) \tag{4}$$

Sorting the points by their codes gives a tour that preserves locality without any distance table. Earlier work used the Sierpiński curve [41]; Section 5.4 compares the Z-order curve with a Hilbert curve. Further constructed tours are nearest-neighbor walks started from the extreme points in the order leftmost ($\min x$), rightmost ($\max x$), bottommost ($\min y$) and topmost ($\max y$). Each walk moves to the nearest unvisited vertex of the candidate graph $\mathcal{G}_{\mathcal{DT}}$ and scans the remaining unvisited vertices only when all candidates have been visited. The number of constructed tours is $M = \max(2, \lfloor P\rho \rfloor)$, capped at five: with $P = 20$ and $\rho = 0.20$, the population contains the Morton tour and the walks from the first three extreme points (20%), and the clusters of the decomposition branch ($P = 30$) contain the Morton tour and all four walks (16.7%). The remaining individuals are uniform random permutations.

Algorithm 2: Spatial seeding

Input: Point coordinates $V \subset \mathbb{R}^2$; Population size P; Smart seed ratio $\rho = 0.20$.
Output: Diverse initial population $\mathcal{P}_0$.

$\mathcal{P}_0 \leftarrow \emptyset$; $M \leftarrow \max(2, \lfloor P \cdot \rho \rfloor)$;
$\pi_{\text{morton}} \leftarrow \text{MortonZOrderTour}(V)$;
$\mathcal{P}_0 \leftarrow \mathcal{P}_0 \cup \{\pi_{\text{morton}}\}$;
corners $\leftarrow [\arg\min_i x_i, \arg\max_i x_i, \arg\min_i y_i, \arg\max_i y_i]$;
foreach $c \in$ *corners* **do**
    if $|\mathcal{P}_0| < M$ **then**
        $\pi_{\text{corner}} \leftarrow \text{FastNearestNeighbor}(V, \text{start} = c)$;
        $\mathcal{P}_0 \leftarrow \mathcal{P}_0 \cup \{\pi_{\text{corner}}\}$;
while $|\mathcal{P}_0| < P$ **do**
    $\pi_{\text{rand}} \leftarrow \text{RandomPermutation}(N)$;
    $\mathcal{P}_0 \leftarrow \mathcal{P}_0 \cup \{\pi_{\text{rand}}\}$;
return $\mathcal{P}_0$;

4.2 Lamarckian evolutionary core

Selection, crossover and mutation: Each generation keeps the incumbent π^* (elitism) and fills the remaining $P-1$ slots with offspring. Both parents are chosen by tournament selection of size 3. With probability $P_c = 0.9$, the child is produced by order crossover (OX1), which copies a random segment of p_1 and fills the remaining positions with the missing cities in the cyclic order of p_2; a boolean presence bitmask $\mathcal{B} \in \{0, 1\}^N$ makes the membership test constant-time, so the crossover runs in $\mathcal{O}(N)$. With probability

$P_m = 0.2$, the child then undergoes either a random segment inversion or a swap of two cities (each with probability 1/2). Every offspring is improved by the three-tier local search and written back into the population (Lamarckian inheritance); the offspring are improved in parallel threads.

Double-bridge four-opt kick perturbation: When the incumbent has not improved for $\tau = 5$ generations, HSD-MA applies a double-bridge four-opt kick [12, 42, 68] to it. The tour is partitioned into four blocks (A, B, C, D) via three random cuts and reconnected as (A, D, C, B), altering four edges without reversing segments. The kicked tour is re-optimized by the three-tier local search and becomes the new incumbent, which the next generation carries by elitism; the best tour found so far is stored separately and returned.

4.3 Three-tier local search

HSD-MA organizes local improvement into three tiers applied once in sequence, each run to its own local optimum (Algorithm 3); the result is therefore not guaranteed to be a joint local optimum of all three neighborhoods. All three tiers draw candidate edges from the same sparse graph $\mathcal{G}_{\mathcal{DT}}$: for every vertex, its 1-hop and 2-hop neighbors in the planar Delaunay triangulation [4, 38, 62], sorted by distance and truncated to the nearest $k_{\max} = 15$. Adding 2-hop neighbors follows the observation that the Delaunay graph alone can be too small as a candidate set [39]; Section 5.4 compares this choice with 1-hop Delaunay and k-nearest-neighbor candidates. Vertices that do not appear in the triangulation, for example duplicate coordinates, receive their $k_{\max}$ nearest neighbors instead. In the decomposition branch, each cluster uses the candidate graph of its own points and the final refinement uses that of the whole instance. Each tier processes a queue of active vertices with don't-look bits and applies the first improving move found [10, 16]. In Tier 1, neighbor-list 2-opt considers, for each vertex a and each of its two tour neighbors b, new edges (a, c) with $c \in \mathcal{G}_{\mathcal{DT}}(a)$ and $d(a, c) < d(a, b)$, and applies the reversal if $d(a, b) + d(c, c') - d(a, c) - d(b, c') > 0$, where c' is the corresponding tour neighbor of c. Since distances are rounded to integers, all tiers accept a move only if it shortens the tour by at least one unit. In Tier 2, the 2.5-opt node shift [10], the single-vertex case of Or-opt [11], removes a vertex u from between its tour neighbors and reinserts it next to a candidate $c \in \mathcal{G}_{\mathcal{DT}}(u)$ when this shortens the tour. Tier 3 implements one of the pure 3-opt reconnection types, the segment insertion (also called or-3opt) that removes the edges (a, b), (c, d) and (e, f), with b the successor of a, c the predecessor of d and f the successor of e, and adds (a, d), (e, b) and (c, f); this moves the segment $d \dots e$ in front of $b \dots c$ without reversing either, and generalizes Or-opt to segments of arbitrary length. The other 3-opt reconnection types, which involve segment reversals, are not used. Only $d \in \mathcal{G}_{\mathcal{DT}}(a)$ and $e \in \mathcal{G}_{\mathcal{DT}}(b)$ are considered, and the partial gains $d(a, b) - d(a, d)$ and $d(a, b) + d(c, d) - d(a, d) - d(e, b)$ must remain positive. Distances are computed from coordinates when needed, so no distance matrix is stored.

Remark 1 (Cost of the candidate-restricted tiers) As for any neighbor-list k-opt implementation [10, 16], truncating the candidate lists to $k_{\max}$ entries bounds the work of examining one active vertex: Tier 3 evaluates at most $k_{\max}$ choices of $d \in \mathcal{G}_{\mathcal{DT}}(a)$ and, for each, at most

Algorithm 3: Three-tier local search

Input: Tour permutation π; coordinates V; Delaunay candidate graph $\mathcal{G}_{\mathcal{DT}}$.
Output: Locally optimized tour π'.

$\pi' \leftarrow \text{NeighborList2Opt}(\pi, V, \mathcal{G}_{\mathcal{DT}})$; `// Tier 1: crossing elimination`
$\pi' \leftarrow \text{NodeShift25Opt}(\pi', V, \mathcal{G}_{\mathcal{DT}})$; `// Tier 2: node displacement`
$\pi' \leftarrow \text{Restricted3OptDelaunay}(\pi', V, \mathcal{G}_{\mathcal{DT}})$; `// Tier 3: segment exchange`
return π';

$k_{\max}$ choices of $e \in \mathcal{G}_{\mathcal{DT}}(b)$, that is at most $k_{\max}^2 = 225$ candidate moves, and Tiers 1 and 2 evaluate $\mathcal{O}(k_{\max})$ moves. Examining every vertex once therefore costs $\mathcal{O}(k_{\max}^2 N)$ evaluations. This does not bound the total running time of a tier: improving moves re-activate vertices, so the number of examinations is not bounded a priori, and applying a move costs time proportional to the length of the reversed or relocated segments, up to $\mathcal{O}(N)$ on the array representation used here. Building $\mathcal{G}_{\mathcal{DT}}$ takes $\mathcal{O}(N \log N)$ time for the triangulation plus $\mathcal{O}(\sum_v \deg(v)^2 \log N)$ for collecting and sorting 2-hop neighbors, which is near-linear when vertex degrees are bounded but can grow quadratically when a few vertices have very high Delaunay degree.

4.4 Complexity and memory

Table 1 summarizes the cost of the HSD-MA components, and Table 2 reports measured peak memory. Peak working set was read with `psutil` in a fresh process for one HSD-MA run per instance; it includes the Python interpreter, the imported libraries, the compiled Numba code and the loaded instance, which together occupied 145 to 183 MiB immediately before the call to the solver.

Table 1 Cost of the HSD-MA components (see Remark 1 for the local search tiers)

Component	Cost	Remark
Distance storage	$\mathcal{O}(N)$	Distances computed from coordinates; ≤ 15 candidates per vertex
Candidate graph $\mathcal{G}_{\mathcal{DT}}$	$\mathcal{O}(N \log N + \sum_v \deg(v)^2 \log N)$	Delaunay triangulation plus 2-hop collection
Morton seed	$\mathcal{O}(N \log N)$	Bit interleaving and sorting
Nearest-neighbor walk	$\mathcal{O}(N k_{\max})$ typical, $\mathcal{O}(N^2)$ worst case	Linear scan only when all candidates are visited
Order crossover (OX1)	$\mathcal{O}(N)$	Boolean presence bitmask
One examination of all vertices, Tier 3	$\leq k_{\max}^2 N = 225N$ evaluations	Moves cost up to $\mathcal{O}(N)$ each

Table 2 Measured peak working set of one HSD-MA run (seed 101) and the size a dense double-precision distance matrix would require

Instance	N	Peak working set (MiB)	Dense $N \times N$ matrix (MiB)
`pcb3038`	3,038	187.7	70.4
`pla33810`	33,810	240.1	8,721.3
`lrb744710`	744,710	968.4	4,231,208.7

5 Experimental evaluation and statistical analysis

5.1 Experimental setup and benchmark suites

All experiments ran on a laptop with an AMD Ryzen 9 6900HX processor (8 cores, 16 threads) and 24 GB of RAM under 64-bit Windows 11 Pro, using Python 3.13.15, NumPy 2.5.3, SciPy 1.18.1, scikit-learn 1.9.1 and Numba 0.68.0 [69]. The local search kernels of all methods are compiled with Numba. HSD-MA can improve offspring in parallel with Numba's `prange`; its results do not depend on the number of threads. The controlled comparison, the budget study and the comparison with LKH-3 and GA-EAX (Sections 5.2 and 5.6) use one thread for every method; the remaining experiments use 16 threads for HSD-MA. Runtimes are wall-clock times measured after a warm-up run, so they exclude just-in-time compilation. On this laptop, absolute runtimes varied between sessions: identical single-threaded HSD-MA runs, which produced the same tours, took 1.4 to 1.7 times longer in one batch than in another. We therefore compare times only within a batch, and runtimes reported in different tables are not directly comparable.

Unless stated otherwise, HSD-MA uses $P = 20$, $T_{\max} = 20$, $P_c = 0.9$, $P_m = 0.2$, $\tau = 5$, $\rho = 0.20$, tournament size 3 and $k_{\max} = 15$; for $N > 5{,}000$, clusters are solved with $P = 30$ and $T_{\max} = 20$. All seeded experiments use seeds 101–130.

Evaluations were performed across two standard benchmark collections. First, the TSPLIB suite [2] provides seven representative circuit and geometric instances spanning a broad scale spectrum: `lin318` ($N = 318$), `pcb442` ($N = 442$), `pcb1173` ($N = 1{,}173$), `pcb3038` ($N = 3{,}038$), `fnl4461` ($N = 4{,}461$), `pla7397` ($N = 7{,}397$), and `pla33810` ($N = 33{,}810$). Second, the Waterloo VLSI collection [70] encompasses all 102 circuit instances from the University of Waterloo benchmark library, ranging from $N = 131$ (`xqf131`) to $N = 744{,}710$ (`lrb744710`).

Solution quality is reported as the percentage gap relative to the Best Known Solution (BKS):

$$\text{Gap } (\%) = \frac{L(\pi) - \text{BKS}}{\text{BKS}} \times 100\% \tag{5}$$

5.2 Controlled comparison: does the population pay off?

Following statistical guidelines for evolutionary computation [71–73], we compared HSD-MA with nine baselines on 10 instances from TSPLIB and the Waterloo VLSI collection with N from 131 to 3,038 (`xqf131`, `xqg237`, `lin318`, `pma343`, `bcl380`, `pcb442`, `pcb1173`, `dca1389`, `bck2217`, `pcb3038`), with 30 seeded runs per method and instance

(3,000 runs). All methods ran single-threaded, all except ILS-3T in the same batch, and every 2-opt, 2.5-opt and 3-opt move of every method is executed by the same compiled kernels over the same Delaunay candidate graph, so that differences come from the algorithms rather than from implementation quality. The baselines are:

- Pure GA [74] (population 30, 35 generations) and Pure PSO [75] (30 particles, 35 iterations), both without local search; the PSO is our own permutation variant in which order crossover towards the personal and global best plays the role of the velocity update, in the spirit of discrete PSO for the TSP [76];
- Pure 2-Opt (a nearest-neighbor tour improved by 2-opt [9]);
- three hybrids with 2-opt on every offspring or particle: a GA-based memetic algorithm (MA-GA-2Opt, population 20, 20 generations) [43], the permutation PSO with 2-opt (H-DPSO-2Opt, 20 particles, 20 iterations), and MMAS-2Opt [77] (8–12 ants, 10–15 iterations, 2-opt on the iteration-best tour);
- MA-GA-3T and H-DPSO-3T, identical to MA-GA-2Opt and H-DPSO-2Opt except that every offspring receives the full three-tier local search of HSD-MA;
- ILS-3T, an iterated local search without population that starts from the best of the five constructed seeds of Section 4.1 after three-tier local search, then repeatedly applies a double-bridge kick followed by three-tier local search and accepts improvements. After each kick, the local search re-examines all vertices rather than only the endpoints of the changed edges, as standard iterated local search would [78], which makes ILS-3T slower than necessary. To match the time per run exactly, ILS-3T was run in a separate batch: for each instance and seed, single-threaded HSD-MA was run first, and ILS-3T was then given exactly the wall-clock time of that run. These HSD-MA runs reproduced the tours of the main batch in all 300 cases.

In the GA-based baselines, a child is passed to the swap mutation with probability 0.2 (MA-GA) or 0.25 (Pure GA), and the mutation operator itself swaps two cities with probability 0.2, so the effective swap-mutation rates are 0.04 and 0.05. No baseline was tuned. Because the baselines draw their random numbers in a fixed order, re-running them in a later session reproduced all 1,800 tour lengths of the six 2-opt-based and population-only baselines exactly, and the 300 HSD-MA results were identical with 1 and 16 threads.

In Tables 3 and 4, every method with the three-tier local search has a lower mean gap than every method with 2-opt only on every instance. In the rank test, HSD-MA is significantly better than MA-GA-2Opt, MMAS-2Opt and Pure 2-Opt (Holm $p \leq 0.004$), but its difference to H-DPSO-2Opt just misses significance (Holm $p = 0.055$), and among the four methods with the three-tier local search the rank test finds no difference (Holm $p = 1.00$).

We also compared HSD-MA with each baseline on each instance by a two-sided Wilcoxon signed-rank test on the 30 seed-paired costs, with Holm correction over the seven competitive baselines, and report the Vargha–Delaney effect size A_{12}, the probability that HSD-MA finds the shorter tour. HSD-MA is significantly better than each of the four 2-opt-based methods on all ten instances.

Against ILS-3T, which uses the same local search and the same time but no population, HSD-MA is significantly better on `xqf131` and `xqg237` ($A_{12} = 0.74$ and 0.86), not

Table 3 Mean gap to the BKS (%) over 30 runs. The first two columns have no local search; 2Opt and 3T denote 2-opt only and the three-tier local search. The last row gives the mean runtime over the ten instances (single-threaded). [a]ILS-3T was run in a separate batch, interleaved with HSD-MA, and each run was given exactly the time of the HSD-MA run with the same seed (Section 5.2).

Instance	N	GA	PSO	2-Opt	MA-GA 2Opt	H-DPSO 2Opt	MMAS 2Opt	MA-GA 3T	H-DPSO 3T	ILS 3T	HSD-MA
`xqf131`	131	465	445	6.74	1.05	0.80	4.46	0.06	0.06	0.17	**0.04**
`xqg237`	237	838	799	9.62	2.09	1.84	6.43	0.81	0.69	1.13	**0.50**
`lin318`	318	1,040	1,005	7.02	2.19	1.85	5.23	0.78	0.71	**0.68**	0.72
`pma343`	343	1,899	1,832	7.89	1.92	1.64	5.79	0.68	**0.45**	0.71	0.64
`bcl380`	380	1,208	1,174	7.90	1.66	1.63	6.51	0.71	0.73	0.75	**0.60**
`pcb442`	442	1,203	1,157	8.90	2.26	1.84	6.27	0.98	0.93	**0.80**	0.93
`pcb1173`	1,173	2,158	2,131	9.94	4.34	4.13	9.28	2.39	2.10	**1.68**	2.28
`dca1389`	1,389	2,964	2,918	7.75	4.92	4.29	7.75	2.54	**2.11**	2.19	2.63
`bck2217`	2,217	3,926	3,901	11.00	6.82	6.20	10.90	3.94	3.48	**2.80**	3.79
`pcb3038`	3,038	3,629	3,612	7.01	6.03	5.63	7.01	3.58	3.20	**2.30**	3.47
Time (s)		0.23	0.29	0.01	0.23	0.27	3.09	0.31	0.35	[a]	0.16

Table 4 Friedman ranks of the eight methods with local search (Pure GA and Pure PSO are excluded as non-competitive) and post-hoc Holm test with HSD-MA as control ($N = 10$ instances, $k = 8$)

Algorithm	Mean rank	Diff.	z	Unadj. p	Holm p	Sig. ($\alpha = 0.05$)
H-DPSO-3T	2.05	-0.25	-0.228	0.819	1.000	No
HSD-MA (control)	2.30	—	—	—	—	—
ILS-3T	2.30	0.00	0.000	1.000	1.000	No
MA-GA-3T	3.35	$+1.05$	0.959	0.338	1.000	No
H-DPSO-2Opt	5.00	$+2.70$	2.465	0.014	0.055	No
MA-GA-2Opt	6.00	$+3.70$	3.378	7.3×10^{-4}	3.7×10^{-3}	Yes
MMAS-2Opt	7.10	$+4.80$	4.382	1.2×10^{-5}	7.1×10^{-5}	Yes
Pure 2-Opt	7.90	$+5.60$	5.112	3.2×10^{-7}	2.2×10^{-6}	Yes

Friedman: $\chi^2_F = 63.04$ ($p = 3.7 \times 10^{-11}$); Iman-Davenport: $F_F = 81.54$ ($p = 3.9 \times 10^{-29}$).
Critical differences ($\alpha = 0.05$): Bonferroni-Dunn 2.95, Nemenyi 3.32.

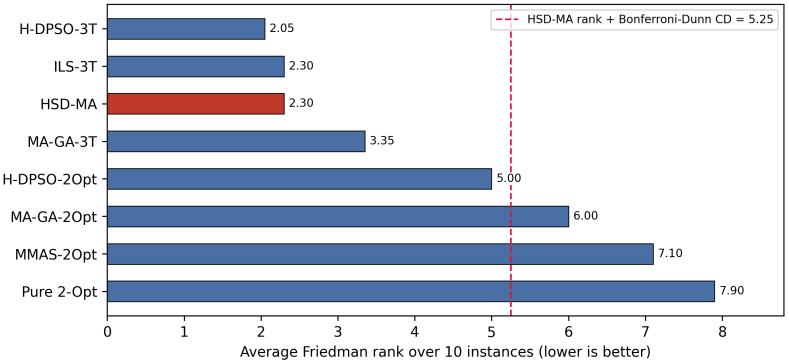

Fig. 1 Average Friedman ranks of the eight methods with local search over the 10 instances; the dashed line marks the HSD-MA rank plus the Bonferroni–Dunn critical difference.

significantly different on `lin318`, `pma343`, `bcl380` and `pcb442`, and significantly worse on the four instances with $N \geq 1{,}173$ ($A_{12} = 0.06$, 0.19, 0.02 and 0.00 on `pcb1173`, `dca1389`, `bck2217` and `pcb3038`). On `pcb3038`, ILS-3T found the shorter tour in all 30 seed pairs, with a mean gap 1.16 percentage points lower (95% bootstrap interval of the mean cost difference, relative to the mean HSD-MA cost, $[1.00\%, 1.25\%]$). Because ILS-3T starts from the same constructed tour for every seed, pairing by seed is arbitrary for this comparison; unpaired Mann–Whitney tests lead to the same conclusions on every instance. Against H-DPSO-3T, HSD-MA is better on `xqg237`, worse on `pma343`, `dca1389`, `bck2217` and `pcb3038`, and not different elsewhere, while H-DPSO-3T needs about twice the time. Against MA-GA-3T, HSD-MA is better on `xqg237` and not different on the other nine instances. The 2-opt baselines are degenerate in two respects that inflate their differences to HSD-MA: Pure 2-Opt is deterministic (a single tour length per instance across all seeds), and MMAS-2Opt returned the Pure 2-Opt tour in all 30 runs on `dca1389` and `pcb3038`. Without local search, Pure GA and Pure PSO end 445% to $3,926\%$ above the BKS.

These results identify the three-tier local search, not the evolutionary framework, as the source of the improvement over 2-opt-based memetic algorithms. Section 5.6 examines whether the population becomes useful when more time is available.

5.3 Population diversity

To examine whether spatial initialization and Lamarckian write-back cause premature convergence, we recorded population diversity inside the actual solvers on `pcb442` for 30 seeds and 20 generations, for HSD-MA, MA-GA-2Opt and Pure GA (the latter two with the same parameters and 2-opt kernel as in Section 5.2), using the normalized average pairwise edge difference:

$$\mathcal{D}(\mathcal{P}_t) = \frac{1}{\binom{P}{2}} \sum_{1 \leq i < j \leq P} \left(1 - \frac{|E(\pi_i) \cap E(\pi_j)|}{N}\right) \tag{6}$$

where $E(\pi)$ is the set of undirected edges in tour π. Values near 1.0 indicate distinct tours, whereas 0.0 indicates an identical population.

Figure 2 shows that all three algorithms lose diversity, but at different rates and with very different outcomes. Pure GA starts at $\mathcal{D} = 0.995$ and decays slowly to 0.053 after 20 generations, while its best tour stays more than $1,200\%$ above the BKS. MA-GA-2Opt starts from 2-opt-optimized random tours, which already share many edges ($\mathcal{D} = 0.334$, best gap 4.91%); its diversity falls to 0.054 by generation 10 and to 0.008 by generation 20, while its best gap improves to $2.26 \pm 0.78\%$. HSD-MA starts from a more diverse population ($\mathcal{D} = 0.974$), because 80% of it consists of random permutations that are not optimized before the first generation, and loses most of this diversity in the first generation (0.298), when every offspring is driven to a three-tier local optimum; at the same time its best gap falls from 30.4% (the best constructed seed) to 2.91%. From then on the two memetic algorithms lose diversity at similar rates, HSD-MA reaching 0.081 at generation 10 and 0.033 at generation 20, while its best gap keeps improving to $0.93 \pm 0.30\%$. This behavior is consistent with the known effects of Lamarckian learning, which speeds up convergence [45] but, combined with powerful local search,

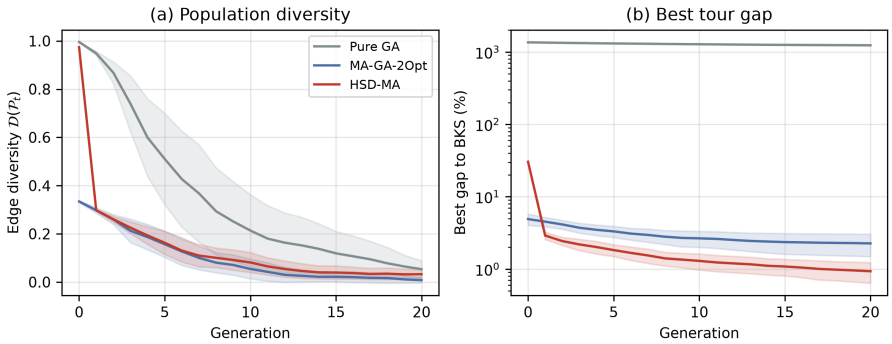

Fig. 2 Population dynamics on `pcb442` ($N = 442$), mean ± one standard deviation over 30 seeds: (a) pairwise edge diversity $\mathcal{D}(\mathcal{P}_t)$; (b) best tour gap to the BKS (log scale).

can cause a rapid loss of diversity [46]. In HSD-MA, the double-bridge kick of the incumbent after $\tau = 5$ stagnant generations re-injects new edges into the elite slot, but the measurements show that it does not maintain a diverse population; the continued improvement comes from the stronger local search rather than from preserved diversity.

5.4 Ablation and parameter study

To quantify the contribution of each component, we ran a two-part ablation on five TSPLIB instances (`lin318`, `pcb442`, `pcb1173`, `pcb3038`, `fnl4461`). The seeding part measures the gap of each seed type at $t = 0$; the random column averages 30 permutations. The local search part runs the complete HSD-MA 30 times per instance (seeds 101–130) with only Tier 1, with Tiers 1–2, and with all three tiers, all other settings unchanged; it also reports a single descent from the Morton tour without evolution.

Table 5 Seeding ablation: gap to the BKS (%) of the initial tours at $t = 0$

Instance	N	Random (mean ± SD)	Morton Z-order	Corner NN (best)	Corner NN (worst)
`lin318`	318	1,294.90 ± 33.92	85.11	22.84	35.36
`pcb442`	442	1,420.95 ± 34.70	86.31	30.41	39.09
`pcb1173`	1,173	2,375.45 ± 32.12	91.24	26.58	33.50
`pcb3038`	3,038	3,821.16 ± 26.30	93.06	24.41	29.92
`fnl4461`	4,461	4,459.82 ± 31.35	91.66	22.38	24.88

Tables 5–6 and Figure 3 support two observations. Regarding initialization, random permutations start 1,295% to 4,460% above the BKS on average and the Morton tour 85% to 93%, whereas the best of the four corner nearest-neighbor walks starts 22% to 30% above the BKS. The Morton tour therefore contributes structural variety rather than quality. Regarding local search depth, each additional tier lowers the final gap of the complete algorithm on every instance. Adding the 2.5-opt node shift to 2-opt reduces the mean gap by 0.68 to 1.61 percentage points, and adding Delaunay 3-opt

Table 6 Local search ablation: final gap to the BKS (%) of the complete HSD-MA, mean ± sd over 30 runs, with Tier 1 only (2-opt), Tiers 1–2 (+ 2.5-opt) and all tiers (+ Delaunay 3-opt); the last column gives the gaps of a single descent from the Morton tour with the three configurations

Instance	N	Tier 1	Tiers 1–2	Tiers 1–3	Single descent (1 / 1–2 / 1–3)
`lin318`	318	1.77 ± 0.48	1.09 ± 0.42	**0.72** ± 0.34	6.43 / 4.55 / 3.58
`pcb442`	442	2.11 ± 0.48	1.31 ± 0.35	**0.93** ± 0.30	8.06 / 6.52 / 5.26
`pcb1173`	1,173	4.34 ± 0.45	2.78 ± 0.39	**2.28** ± 0.29	8.54 / 6.54 / 5.19
`pcb3038`	3,038	5.77 ± 0.31	4.16 ± 0.30	**3.47** ± 0.29	9.73 / 7.67 / 6.23
`fnl4461`	4,461	5.35 ± 0.18	4.05 ± 0.14	**3.46** ± 0.12	9.34 / 6.97 / 5.81

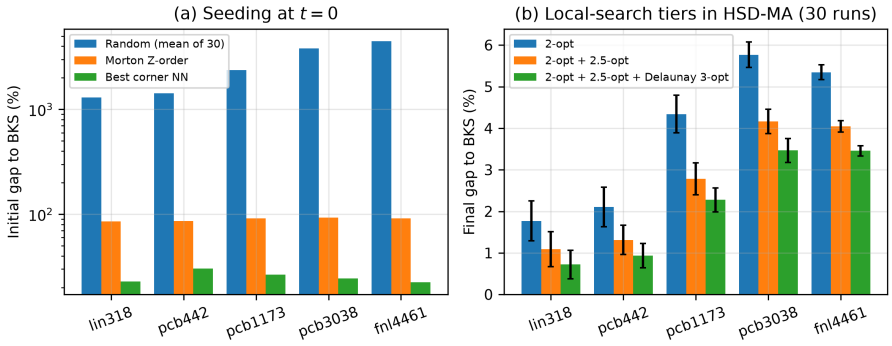

Fig. 3 Ablation results on five instances: (a) initial gaps at $t = 0$ of the seed types (log scale); (b) final gaps of the complete HSD-MA with one, two or three local search tiers (mean ± sd over 30 runs).

reduces it by a further 0.4 to 0.7 points. Both steps are significant on all five instances under paired Wilcoxon tests over the 30 seeds: $p \leq 8.0 \times 10^{-8}$ for adding 2.5-opt (except `lin318` and `pcb1173`, $p \leq 2.0 \times 10^{-6}$) and $p \leq 2.2 \times 10^{-3}$ for adding 3-opt, which improved the result in 22 to 30 of the 30 seed-paired runs. The single descents show the same ordering, so the gain of the higher tiers is not an artifact of the evolutionary loop.

Table 7 reports a further ablation of individual components on the same five instances (30 runs each, all variants run in the same batch). Removing crossover, so that offspring are copies of tournament winners modified only by mutation and local search, raises the mean gap from 2.17% to 3.07% and is significantly worse on all five instances. This does not isolate the value of recombination itself: a copy of a tournament winner is already a local optimum, so after a single swap mutation the local search has little to do, and the variant is also faster on four of the five instances. Restricting the candidates to 1-hop Delaunay neighbors raises the gap to 2.63% (significantly worse on four instances), which agrees with the observation that the Delaunay graph alone is too small as a candidate set [39]. Replacing the 1-hop and 2-hop Delaunay candidates by the 15 nearest neighbors gives the same mean gap (2.15% against 2.17%) and is significantly better on `fnl4461`, so the Delaunay structure brings no advantage over a plain k-nearest-neighbor list of the same size. Replacing the Morton curve by a Hilbert curve, removing the constructed seeds altogether, or disabling the kick changes the mean gap by at most 0.02 percentage points, with no significant difference on any

instance; since we did not run equivalence tests, small effects of these components cannot be excluded. With $\tau = 5$ and $T_{\max} = 20$, at most three kicks can occur in a run, so this ablation says little about the kick at longer budgets. The kick is indeed rarely used: over the 300 single-threaded HSD-MA runs of Section 5.2, it fired 185 times in 122 runs, produced a new best tour immediately in 3 cases and was followed by a new best tour before the next kick in 44 cases, while in 81 cases the next improvement of the incumbent merely returned to the cost of the best tour found before the kick.

Table 7 Component ablation on five instances (30 runs each, all variants in the same batch): mean ± standard deviation of the gap to the BKS (%). $^{+}/^{-}$: significantly better/worse than the default (paired Wilcoxon test over seeds, Holm correction over the six variants of an instance, $\alpha = 0.05$). The last row is the unweighted mean over the five instances

Instance	Default	1-hop Del.	k-NN ($k = 15$)	Hilbert	No crossover	No kick	No seeds
`lin318`	0.72 ± 0.34	1.12^{-} ± 0.40	0.66 ± 0.33	0.74 ± 0.39	1.69^{-} ± 0.42	0.74 ± 0.32	0.67 ± 0.41
`pcb442`	0.93 ± 0.30	0.89 ± 0.24	0.96 ± 0.30	0.86 ± 0.27	2.12^{-} ± 0.40	0.93 ± 0.29	0.91 ± 0.24
`pcb1173`	2.28 ± 0.29	3.03^{-} ± 0.43	2.20 ± 0.24	2.19 ± 0.24	3.67^{-} ± 0.30	2.28 ± 0.29	2.17 ± 0.30
`pcb3038`	3.47 ± 0.29	4.29^{-} ± 0.33	3.58 ± 0.24	3.49 ± 0.23	3.98^{-} ± 0.20	3.47 ± 0.29	3.56 ± 0.26
`fnl4461`	3.46 ± 0.12	3.81^{-} ± 0.14	3.35^{+} ± 0.12	3.47 ± 0.12	3.90^{-} ± 0.08	3.45 ± 0.12	3.53 ± 0.13
Mean	2.17	2.63	2.15	2.15	3.07	2.17	2.17

Table 8 varies one parameter at a time from the default on the five ablation instances (30 runs each); the default row is the default variant of Table 7. A larger population or more generations give better tours at a proportionally higher cost: the mean gap falls from 2.69% to 1.85% when P grows from 10 to 40, and from 3.13% to 1.47% when $T_{\max}$ grows from 5 to 80, while the mean runtime grows by a factor of 3.7. Shorter candidate lists lose quality (2.45% at $k_{\max} = 8$, 2.20% at $k_{\max} = 10$), while $k_{\max} = 20$ is slightly better than the default (2.11% against 2.17%) at a similar runtime, so the default $k_{\max} = 15$ is not optimal. The stagnation threshold has no effect (2.17% for $\tau = 3$ and $\tau = 10$), consistent with the rare use of the kick. The cluster size of the decomposition branch was varied on the two `pla` instances only, with no consistent trend: on `pla7397` larger clusters give better tours (6.00%, 5.08% and 3.77% for 150, 250 and 500 points per cluster), whereas on `pla33810` the default size of 250 gives the worst result (6.43%, 6.85% and 6.48%).

5.5 Results on the 102 Waterloo VLSI instances

To evaluate behavior on diverse industrial circuits, HSD-MA was run once (seed 101) on each of the 102 instances of the Waterloo VLSI collection, with the default parameters above. Gaps are computed against the tour lengths listed on the collection's summary page (optimal for the smaller instances, best known for the larger ones); all 102 values used were checked against that page. In addition, Table 9 reports 30 seeded runs on seven TSPLIB instances, including `pla7397` and `pla33810`, which use the decomposition branch.

Table 10 and Figure 4 summarize the 102 runs. On instances with $N < 1,000$, HSD-MA averages a 1.27% gap in 0.06 s and finds the optimal tour of `xqf131`. The

Table 8 Parameter sensitivity on `lin318`, `pcb442`, `pcb1173`, `pcb3038` and `fnl4461` (30 runs each, one parameter varied at a time): gap to the BKS (%) and runtime (s), each averaged over the five instances. Default values are marked with an asterisk

Parameter	Description	Value	Mean gap	Mean time
P	Population size	10	2.69	0.217
		20*	2.17	0.376
		40	1.85	0.871
$T_{\max}$	Generations	5	3.13	0.245
		10	2.64	0.308
		20*	2.17	0.376
		40	1.80	0.579
		80	1.47	0.914
$k_{\max}$	Candidates per vertex	8	2.45	0.376
		10	2.20	0.381
		15*	2.17	0.376
		20	2.11	0.390
τ	Stagnation threshold	3	2.17	0.391
		5*	2.17	0.376
		10	2.17	0.392

Table 9 HSD-MA on seven TSPLIB instances: gap to the optimal tour length (%) and runtime over 30 runs (seeds 101–130, 16 threads); tours of the `pla` instances are scored with their official `CEIL_2D` metric

Instance	N	Branch	Mean gap	SD	Best	Worst	Mean time (s)
`lin318`	318	memetic	0.72	0.34	0.15	1.68	0.02
`pcb442`	442	memetic	0.93	0.30	0.47	1.76	0.03
`pcb1173`	1,173	memetic	2.28	0.29	1.66	2.89	0.07
`pcb3038`	3,038	memetic	3.47	0.29	2.93	4.11	0.22
`fnl4461`	4,461	memetic	3.46	0.12	3.21	3.68	0.34
`pla7397`	7,397	decomposition	5.08	0.44	4.31	6.11	0.62
`pla33810`	33,810	decomposition	6.85	0.31	6.27	7.51	2.46

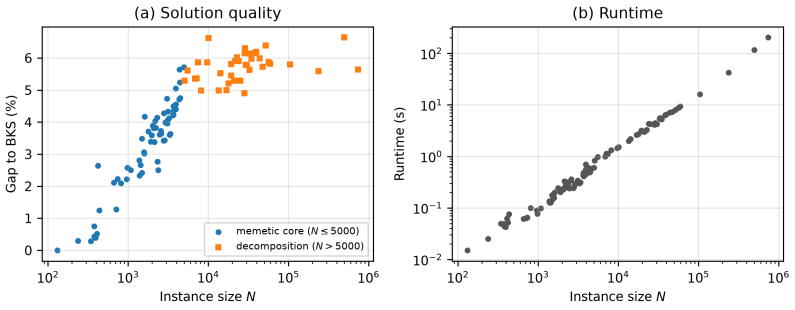

Fig. 4 HSD-MA on all 102 Waterloo VLSI instances ($N = 131$ to $N = 744,710$), one run each: (a) gap to the BKS; (b) runtime.

mean gap grows with instance size, to 3.87% for $1,000 \le N < 5,000$, 5.50% for $5,000 \le N < 20,000$ and 5.88% for the 25 instances with $N \ge 20,000$, where the per-instance gaps lie in a narrow band from 4.90% to 6.64%. All 102 instances were solved in 533.6 s in total; only `lra498378` (117.2 s) and `lrb744710` (203.3 s, gap 5.64%) took longer than one minute. These figures come from a single run per instance; the 30-run TSPLIB results in Table 9 indicate a run-to-run standard deviation of 0.1 to 0.5 percentage points.

Table 10 HSD-MA on the 102 Waterloo VLSI instances (one run each), grouped by size

Size range	Count	Mean gap	Min gap	Max gap	Mean time (s)	Max time (s)
$N < 1{,}000$	15	1.27%	0.00%	2.64%	0.06	0.10
$1{,}000 \le N < 5{,}000$	48	3.87%	2.34%	5.72%	0.32	0.70
$5{,}000 \le N < 20{,}000$	14	5.50%	4.98%	6.63%	1.79	3.20
$N \ge 20{,}000$	25	5.88%	4.90%	6.64%	19.68	203.34

For reference, exact branch-and-cut solvers need hours to days of CPU time even for TSPLIB circuit instances with a few thousand points; for example, Concorde took about 22 CPU hours on `pcb3038` and about 5 CPU days on `pla7397`, the latter with the optimal value supplied and tuned settings, on 1999 hardware [79]. Section 5.6 compares HSD-MA with LKH-3 on the same machine.

5.6 Longer budgets and comparison with LKH-3 and GA-EAX

Does the population pay off with more time? Population-based search is often expected to profit from longer runs. We therefore ran single-threaded HSD-MA with $T_{\max} = 20$, 80 and 320 generations on `pcb442`, `pcb1173` and `pcb3038` (seeds 101–110) and, for every run, ILS-3T for exactly the wall-clock time that this run took (Table 11); 16 times more generations took 7.3 to 8.4 times more time. With more time both methods improve and the difference shrinks, but it never turns in favor of HSD-MA: on `pcb442` the two are indistinguishable at all budgets, and on `pcb1173` and `pcb3038` ILS-3T is significantly better at the two shorter budgets, whereas at the longest budget the difference is no longer significant (on `pcb3038`, ILS-3T still found the shorter tour in 8 of 10 seed pairs). At the shortest budget on `pcb442`, ILS-3T needed 0.035 s instead of 0.028 s, because building and optimizing its five constructed tours takes longer than the budget. Within the tested range, the population therefore does not outperform repeated kicks and local search from a single tour, although with 10 seeds small differences at the longest budget cannot be resolved.

Does the decomposition pay off? The studies above use instances with at most 4,461 points, where HSD-MA runs its memetic core on the whole instance. To test the decomposition branch, we ran single-threaded HSD-MA on `pla7397`, `pla33810` and `sra104815` (seeds 101–110) and, for every run, ILS-3T on the whole instance, without decomposition, for exactly the same time (Table 12). On the two `pla` instances, ILS-3T finds the shorter tour in all ten seed pairs, with mean gaps 2.6 and 1.5 percentage points lower. On `sra104815`, in contrast, HSD-MA is better in all ten pairs, by 1.1

Table 11 HSD-MA with $T_{\max}$ generations versus ILS-3T run for the same wall-clock time (single thread, 10 seeds); mean gaps to the BKS (%), number of seed pairs in which ILS-3T found the shorter (better) or longer (worse) tour, and two-sided Wilcoxon p, unadjusted and with Holm correction over the nine tests

Instance	$T_{\max}$	Time (s)	HSD-MA	ILS-3T	ILS better	ILS worse	p	Holm p
pcb442	20	0.03	0.90	1.32	5/10	5/10	0.695	1.000
pcb442	80	0.07	0.46	0.54	3/10	7/10	0.193	0.773
pcb442	320	0.23	0.33	0.34	4/10	6/10	0.695	1.000
pcb1173	20	0.11	2.27	1.71	9/10	1/10	0.004	0.023
pcb1173	80	0.26	1.40	1.14	10/10	0/10	0.002	0.018
pcb1173	320	0.85	0.87	0.90	5/10	5/10	0.750	1.000
pcb3038	20	0.39	3.41	2.31	10/10	0/10	0.002	0.018
pcb3038	80	0.92	2.25	1.73	10/10	0/10	0.002	0.018
pcb3038	320	2.83	1.31	1.16	8/10	2/10	0.049	0.244

percentage points on average. Up to about 34,000 points, the decomposition therefore costs quality relative to a global iterated local search given the same time; among the three instances tested, it pays off only on the largest one. We did not test where between these sizes the advantage changes sides.

Table 12 HSD-MA with spatial decomposition versus ILS-3T on the whole instance, run for the same wall-clock time (single thread, 10 seeds): mean gap to the BKS (%), mean time and number of seed pairs in which ILS-3T found the shorter tour; all differences are significant (paired Wilcoxon test, Holm-corrected $p = 0.006$)

Instance	N	Time (s)	HSD-MA	ILS-3T	ILS better
pla7397	7,397	0.74	5.23	2.65	10/10
pla33810	33,810	3.74	6.78	5.23	10/10
sra104815	104,814	13.25	5.39	6.52	0/10

Comparison with LKH-3 and GA-EAX. We ran LKH-3 (version 3.0.13) [15] and GA-EAX [22, 80], in the original implementations of their authors, on the same machine and with one thread. For each of the ten instances of Section 5.2, each solver was run with its default settings and with a budget equal to the mean runtime of single-threaded HSD-MA on that instance: for LKH-3, this budget was its search time limit (`TIME_LIMIT`), which excludes the candidate-set preprocessing; for GA-EAX (population 100 and 30 offspring, as in the usage example of the code, whose documentation recommends a population of 300 only for $N > 10{,}000$ [80]), it was a wall-clock limit on the run, which we added to the termination test of the code; it is checked once per generation, after the initial population has been built. LKH-3 was run with seeds 101–110 and GA-EAX with ten trials of its built-in seed sequence; Table 13 reports mean gaps and internal solver execution times. To make GA-EAX read three TSPLIB files whose headers omit the space before the colon, the headers were reformatted; the coordinates are unchanged.

With their default settings, both established solvers find optimal or near-optimal tours on all ten instances (mean gaps at most 0.084% for LKH-3 and 0.025% for GA-EAX), at the cost of 0.15 to 78.9 s and 0.24 to 14.5 s, respectively, against 0.02 to

Table 13 HSD-MA and ILS-3T versus LKH-3 and GA-EAX on the ten instances of the controlled study (single thread). "Equal budget": LKH-3 search time limit or GA-EAX wall-clock limit equal to the HSD-MA time; all runtimes are internal solver execution times excluding instance file input/output and operating-system process initialization (measured via `Time.total` for LKH-3, internal trial timer for GA-EAX, and in-process timer after warm-up for HSD-MA and ILS-3T). Mean gap to the BKS (%) over 30 runs (HSD-MA, ILS-3T) or 10 runs (LKH-3, GA-EAX)

					Equal budget				Default settings			
		HSD-MA		ILS-3T	LKH-3		GA-EAX		LKH-3		GA-EAX	
Instance	N	Gap	Time (s)	Gap	Gap	Time (s)	Gap	Time (s)	Gap	Time (s)	Gap	Time (s)
`xqf131`	131	0.04	0.02	0.22	0.00	0.03	1.93	0.02	0.000	0.15	0.000	0.34
`xqg237`	237	0.50	0.02	0.91	0.23	0.06	2.85	0.06	0.000	0.78	0.000	1.00
`lin318`	318	0.72	0.04	0.78	0.24	0.14	3.43	0.06	0.084	0.73	0.024	0.57
`pma343`	343	0.64	0.05	0.69	2.43	0.12	3.20	0.06	0.000	3.30	0.000	0.84
`bcl380`	380	0.60	0.04	0.79	0.09	0.14	2.77	0.07	0.000	1.43	0.000	0.24
`pcb442`	442	0.93	0.05	0.71	0.03	0.18	3.03	0.08	0.000	1.76	0.000	0.65
`pcb1173`	1,173	2.28	0.17	1.68	0.09	1.14	5.68	0.36	0.002	5.03	0.004	3.84
`dca1389`	1,389	2.63	0.21	2.01	0.21	1.57	5.14	0.49	0.026	22.91	0.024	4.48
`bck2217`	2,217	3.79	0.39	2.65	0.29	4.17	6.54	0.91	0.024	68.03	0.025	7.76
`pcb3038`	3,038	3.47	0.64	2.26	0.09	8.15	5.91	1.65	0.000	78.88	0.000	14.49

0.64 s for HSD-MA. Under the equal budget, LKH-3 is still far better than HSD-MA on nine instances (gaps 0.00% to 0.29%, against 0.04% to 3.79%; on `pma343`, three of the ten LKH-3 runs ended above 4% and the mean was 2.43%), and its internal runtime including candidate generation is about 1.5 to 12.7 times that of HSD-MA (0.03 to 8.15 s against 0.02 to 0.64 s). GA-EAX, in contrast, cannot use such short budgets: building its initial population does not fit into them (measured times 0.02 to 1.65 s against limits of 0.02 to 0.64 s), and with the limit it ends 1.9% to 6.5% above the BKS, worse than HSD-MA on every instance.

Figure 5 places these results on a common time axis for three instances: HSD-MA and ILS-3T occupy the sub-second to few-second range with gaps of about 0.3% to 3.5%, and LKH-3 reaches gaps below 0.1% in up to ten times longer wall-clock time.

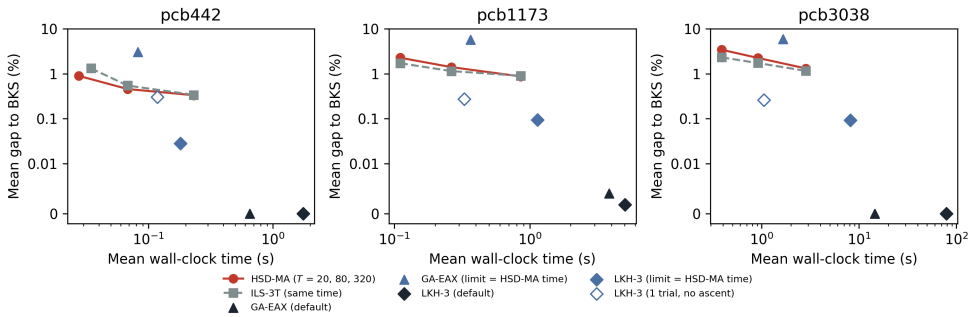

Fig. 5 Mean gap to the BKS versus mean wall-clock time (single thread) on three instances: HSD-MA with $T_{\max} = 20$, 80, 320 and ILS-3T with the same times (10 seeds), and LKH-3 and GA-EAX with their default settings and with a budget equal to the time of HSD-MA with $T_{\max} = 20$ in the controlled study (10 runs), and LKH-3 with one trial and without subgradient ascent (10 runs, interleaved with HSD-MA). The vertical axis is logarithmic above 0.01%.

On the large instances, we ran single-threaded HSD-MA once (seed 101) and LKH-3 once per configuration, all in one batch. Because the default α-nearness candidates of LKH-3 require an expensive subgradient ascent whose cost grows quickly with N [19, 58], LKH-3 was evaluated in three configurations: with the subgradient ascent and without it, each with a search time limit equal to the HSD-MA time, and without the ascent with a single trial and no time limit. Table 14 also reports the peak working sets, measured with `psutil`. HSD-MA ends 5.5% to 7.0% above the BKS. All LKH-3 configurations are far better: 0.33% to 2.05% with the ascent, 0.25% to 1.47% without it and 0.32% to 1.32% with a single trial. LKH-3 also uses less memory on every instance. Its preprocessing, however, makes it slower: on each instance, even the quickest of the two configurations without the ascent needs 2.9 to 4.6 times the wall-clock time of HSD-MA, and the configuration with the ascent 4.8 to 6.6 times. HSD-MA optimizes the rounded Euclidean distance also on the two `pla` instances, whose official `CEIL_2D` metric LKH-3 optimizes directly, which slightly disadvantages HSD-MA there. We did not run GA-EAX on these instances: its documentation states that the Normal version needs about 5.3 GB of memory at $N = 25{,}000$ and provides a separate version for large instances [80], which we did not test. We also did not test Chained Lin–Kernighan [68], a common fast baseline in this time range.

Table 14 Single-threaded HSD-MA (seed 101) versus single runs of LKH-3 (seed 101) on large instances: gap to the BKS (%), wall-clock time (s) and peak RAM working set (MiB). "Ascent" and "No ascent": search time limit equal to the HSD-MA time, with and without subgradient ascent; "1 trial": one trial without subgradient ascent and without time limit.

		HSD-MA			LKH-3						
					Ascent		No ascent		1 trial		
Instance	N	Gap	Time	RAM	Gap	Time	Gap	Time	Gap	Time	RAM
`pla7397`	7,397	5.50	0.7	228	1.12	4.7	0.25	3.3	0.32	3.5	116
`pla33810`	33,810	7.01	3.6	264	2.02	18.4	1.03	12.8	0.79	15.0	36
`sra104815`	104,814	5.80	14.1	324	0.33	72.4	0.53	44.8	0.60	40.7	98
`ara238025`	238,025	5.59	33.1	459	1.48	175.1	1.37	104.0	0.88	107.7	215
`lra498378`	498,378	6.64	75.1	759	1.86	358.9	1.47	225.9	1.32	236.8	444
`lrb744710`	744,710	5.64	108.5	1242	2.05	543.3	0.65	331.6	0.62	324.6	650

The same holds on the ten small instances. Run interleaved with single-threaded HSD-MA on seeds 101–110, LKH-3 with one trial and without the ascent found shorter tours than HSD-MA on nine of the ten instances (mean gaps 0.09% to 0.59% against 0.05% to 3.87%) and needed 2.7 to 4.2 times its runtime (0.04 to 1.05 s against 0.01 to 0.39 s). In summary, LKH-3 dominates HSD-MA in solution quality in every configuration we tested, and GA-EAX does so given a few seconds; the only advantage of HSD-MA is that it returns tours several percent above the BKS in about a third to a fourth of the runtime of the quickest LKH-3 configuration.

6 Case study: drilling cycle time under a machine kinematic model

Tour length is only a proxy for the quantity that matters on the shop floor, the machine cycle time. This case study therefore evaluates HSD-MA tours under a simple kinematic model of a biaxial CNC drilling table, using the same model for the optimized tours and for an unoptimized reference sequence.

6.1 Kinematic model and assumed machine parameters

CNC drilling tables position the spindle with orthogonal X and Y feed drives [81]. Production PCB drilling machines reach X/Y feedrates of up to 60 m/min [82], whereas PCB prototyping plotters specify X/Y travel speeds of 100–150 mm/s [83–85]. This case study assumes a prototyping-class machine with $V_{\text{rapid}} = 120$ mm/s (7.2 m/min), within that range, and an assumed maximum axis acceleration $a_{\max} = 2,500$ mm/s^2 (0.255 g). TSPLIB does not specify units. We assume that the coordinates are in mils (1 mil = 0.0254 mm), since 92% of the `pcb442` coordinates are multiples of 50 and 86% multiples of 100, consistent with component pitches of 0.05 and 0.1 inch, and the resulting board of 76 mm × 97 mm is of realistic size; displacements are converted via $\Delta x_{\text{mm}} = 0.0254 \cdot \Delta x$.

Because the two axes move simultaneously, the time to travel between two holes is the longer of the two axis times [50], which corresponds to the Chebyshev norm (L_∞). Each axis move consists of an acceleration phase, an optional constant-velocity phase and a deceleration phase [50, 86]. Let $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ be two consecutively visited holes, with physical axis displacements $\Delta x = 0.0254 \cdot |x_i - x_j|$ and $\Delta y = 0.0254 \cdot |y_i - y_j|$ in millimetres. With acceleration ramp duration $t_{\text{ramp}} = V_{\text{rapid}}/a_{\max}$ and displacement threshold $d_{\text{ramp}} = V_{\text{rapid}}^2/a_{\max} = 5.76$ mm, the X-axis transit time is

$$t_x = \begin{cases} 2\sqrt{\frac{\Delta x}{a_{\max}}}, & \text{if } \Delta x < d_{\text{ramp}}, \\ \frac{\Delta x}{V_{\text{rapid}}} + \frac{V_{\text{rapid}}}{a_{\max}}, & \text{if } \Delta x \geq d_{\text{ramp}}, \end{cases} \tag{7}$$

with the same formulation for t_y. Displacements shorter than d_{ramp}, such as standard electronic pin pitches of 100 mil (2.54 mm) and 50 mil (1.27 mm), remain entirely within the parabolic acceleration–deceleration regime and never reach V_{rapid}. The rapid traverse time between consecutive holes is

$$t_{\text{traverse}}(p_i, p_j) = \max(t_x, t_y). \tag{8}$$

Table 15 Simulated CNC PCB drilling kinematics under Chebyshev L_∞ servo travel (1 mil = 0.0254 mm, $V_{\text{rapid}} = 120$ mm/s, $a_{\max} = 2{,}500$ mm/s^2)

Instance	Sequence	Tour (mil)	Tour (m)	Trav. (s)	T_{total} (0.15s)	T_{total} (0.60s)
pcb442	File order	221,440	5.625	66.5	2.21 min	5.53 min
	Nearest Neighbor	68,591	1.742	32.6	1.65 min	4.96 min
	NN + 2-Opt	54,095	1.374	29.7	1.60 min	4.92 min
	HSD-MA	51,466	1.307	29.2	1.59 min	4.91 min
	HSD-MA + time 2-opt	51,444	1.307	29.2	1.59 min	4.91 min
	LKH-3 (default)	50,778	1.290	29.1	1.59 min	4.90 min
pcb3038	File order	295,793	7.513	165.2	10.35 min	33.13 min
	Nearest Neighbor	173,613	4.410	134.6	9.84 min	32.62 min
	NN + 2-Opt	147,654	3.750	128.5	9.74 min	32.52 min
	HSD-MA	142,003	3.607	126.7	9.71 min	32.49 min
	HSD-MA + time 2-opt	143,261	3.639	126.4	9.70 min	32.49 min
	LKH-3 (default)	137,694	3.497	125.3	9.68 min	32.47 min

The total cycle time of a drilling sequence $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$ is

$$T_{\text{total}} = \sum_{i=1}^{N} t_{\text{traverse}}(p_{\pi_i}, p_{\pi_{(i \bmod N)+1}}) + N \cdot t_{\text{drill}}, \tag{9}$$

where t_{drill} represents the Z-axis spindle plunge, dwell and retraction cycle. Two values are analyzed: $t_{\text{drill}} = 0.60$ s, obtained by reading the drilling speed of 100 strokes per minute specified for a prototyping plotter [84] as a pure Z cycle, which is an upper bound if that rate includes positioning, and an assumed faster cycle of $t_{\text{drill}} = 0.15$ s.

6.2 Cycle-time results

Table 15 compares six drilling sequences under this model: the TSPLIB file order, a nearest-neighbor (NN) tour started at the first hole, the NN tour improved by neighbor-list 2-opt, the HSD-MA tour (seed 101), the HSD-MA tour further improved by a 2-opt that minimizes the traverse time of the kinematic model directly, and the LKH-3 tour obtained with default settings, which is optimal for both instances. Relative to the file order, all optimized sequences reduce the rapid traverse time substantially, from 66.5 s to about 29 s on pcb442 and from 165.2 s to between 125 and 135 s on pcb3038. Relative to the simple NN + 2-opt sequence, however, the gains of HSD-MA are small: the traverse time falls by 0.5 s (1.7%) on pcb442 and by 1.8 s (1.4%) on pcb3038, which changes the total cycle time by at most 0.03 minutes (about 0.3% to 0.6% with $t_{\text{drill}} = 0.15$ s), and the optimal LKH-3 tour saves only a further 0.1 to 1.4 s. Optimizing the traverse time directly instead of the Euclidean length changes it by less than 0.3%, consistent with Proposition 1. With realistic board dimensions the holes are close together and the drill cycle dominates: with $t_{\text{drill}} = 0.60$ s, rapid positioning accounts for less than 10% of the cycle on pcb442 and about 7% on pcb3038. For these instances, the choice among reasonable sequencing heuristics therefore affects the simulated cycle time only marginally.

These results are simulations under assumed machine parameters and are not measurements on a physical machine. The case study uses a single tool: the TSPLIB

instances carry no hole diameters, whereas real drilling programs group holes by tool, which can be modeled as a clustered traveling salesman problem in which all holes of a class are visited contiguously [87–89].

7 Discussion and limitations

This section discusses where the improvements of HSD-MA come from, its limitations, and its position relative to established solvers.

7.1 Losses at cluster boundaries

For $N > 5{,}000$, HSD-MA partitions the instance with MiniBatchKMeans, solves every cluster independently with the memetic core and connects the cluster tours along a macro-tour over the centroids. Each cluster is optimized as a closed tour, which is opened at its vertex nearest to the exit vertex of the previous cluster and traversed in the direction whose exit lies closer to the next centroid; the edge removed when opening the tour is therefore arbitrary, whereas solving each cluster as a path with fixed endpoints, as in the subproblem schemes of LKH [19], would avoid this loss. Edges of a good global tour that cross cluster boundaries cannot be found during the cluster phase, and the stitching adds detours at every boundary. On the 39 VLSI instances solved by decomposition, the stitched tours were on average 14.0% above the BKS (between 12.1% and 17.4%); the subsequent global three-tier refinement over the Delaunay candidate graph reduced this to 5.75% on average. Boundary effects therefore account for most of the quality loss of the decomposition branch, and the refinement recovers most, but not all, of it. This explains why the mean gap rises from 3.87% for $1{,}000 \leq N < 5{,}000$ to 5.50%–5.88% for the decomposed instances. A stronger refinement, for example re-optimizing windows that straddle cluster boundaries, could narrow the remaining gap.

7.2 Where the improvement comes from

The controlled comparison, the ablations and the budget study point to the same conclusion. The three-tier local search is what separates HSD-MA from 2-opt-based memetic algorithms: every method with the three-tier local search outperforms every method with 2-opt only (Table 3), and within HSD-MA each additional tier lowers the final gap on all five ablation instances (Table 6). This agrees with the long-standing observation that 2.5-opt and 3-opt moves improve substantially on 2-opt local optima [10, 16]. The population, in contrast, does not add value in our setting: with the same local search and the same time, an iterated local search from a single tour is better than HSD-MA on the four instances with $N \geq 1{,}173$ and indistinguishable on four smaller ones, and with 16 times more generations the difference becomes non-significant but does not reverse (Table 11). The same holds for the decomposition branch: given the same time, the iterated local search on the whole instance is better up to 33,810 points, and the decomposition wins only on the instance with 104,814 points (Table 12). On `pcb442`, population diversity falls to 0.08 within ten generations (Figure 2), the kick is rarely triggered, and the constructed seeds and the choice of space-filling curve

have no significant effect. Within the population framework, crossover still helps, since removing it raises the gap; but order crossover does not preserve edges, and the strongest memetic algorithms for the TSP rely on edge-preserving recombination such as EAX or partition crossover [22, 25, 26]. Our measurements with GA-EAX, which finds optimal or near-optimal tours on all ten controlled instances within 15 s, are consistent with this. A population would therefore have to be combined with such a recombination operator, at the cost of more time per generation, to pay off at the budgets considered here. Finally, the Delaunay candidate structure itself brings no advantage over 15 nearest neighbors in our ablation, so the local search tiers, not the specific candidate graph, are the effective ingredient. We did not test whether these findings differ between rectilinear layouts and uniformly random point sets.

7.3 Positioning relative to established solvers

The core routines of HSD-MA and the local search kernels shared with the baselines are compiled with Numba [69], so the quality differences in Section 5.2 are not due to implementation. Compared with exact branch-and-cut solvers such as Concorde, HSD-MA gives up optimality guarantees for short runtimes: all 102 VLSI instances were solved in 533.6 s in total and the largest in 203 s, whereas Concorde required hours to days on TSPLIB circuit instances with a few thousand points [79]. LKH-3, however, finds much better tours than HSD-MA on every large instance in every configuration we tested and uses less memory, although its preprocessing makes even its quickest configuration 2.9 to 4.6 times slower in wall-clock time (Table 14), and with default settings LKH-3 and GA-EAX find optimal or near-optimal tours on all instances of the controlled study within about a minute (Table 13). HSD-MA is therefore useful only when a tour a few percent above the best known one is needed within a fraction of a second to a few minutes and a lightweight Python implementation is preferable; when tour quality matters and a few seconds to minutes are available, LKH-3 is the better choice. The drilling case study adds a further caution: on realistic board dimensions, the drill cycle dominates the machine time, and HSD-MA shortens the simulated cycle time by less than 1% relative to a nearest-neighbor tour improved by 2-opt.

7.4 Candidate graphs in higher dimensions

The candidate graph of HSD-MA relies on the planar Delaunay triangulation, which has $\mathcal{O}(N)$ edges, hence fewer than six neighbors per vertex on average, and can be computed in $\mathcal{O}(N \log N)$ time [62, 63, 90]. This does not carry over to higher-dimensional routing problems, such as three-dimensional path planning in additive manufacturing or multi-axis machining: in $\mathbb{R}^d$ the worst-case size of the Delaunay triangulation grows as $\mathcal{O}(N^{\lceil d/2 \rceil})$ [91, 92], that is, quadratically already for $d = 3$. Since k-nearest-neighbor candidates performed as well as Delaunay candidates in our ablation, they are the natural choice in such settings.

8 Conclusions and future work

This study developed the hierarchical spatial-decomposition memetic algorithm (HSD-MA) for large geometric traveling salesman problems from PCB drilling and VLSI layouts and examined, in controlled experiments, which of its components matter. HSD-MA combines Morton and corner nearest-neighbor seeding, a Lamarckian evolutionary core with elitism, tournament selection, order crossover and double-bridge kicks, a three-tier local search (neighbor-list 2-opt, 2.5-opt node shift and segment-insertion 3-opt) over 1-hop and 2-hop Delaunay candidates, and, for $N > 5{,}000$, a k-means decomposition whose clusters are solved in parallel. Distances are computed on the fly, so memory grows linearly with N.

In a comparison on ten instances with 30 runs each, in which all methods share the same compiled local search kernels and run single-threaded, HSD-MA was significantly better than every memetic, swarm and ant-colony baseline that uses 2-opt only. It was not better than the same evolutionary frameworks or an iterated local search equipped with the same three-tier local search: given the same time per run, the iterated local search found significantly shorter tours on the four instances with $N \geq 1{,}173$, and 16 times more generations reduced but did not reverse this difference. Given the same time, the iterated local search was also better than the decomposition branch on `pla7397` and `pla33810`, and worse only on the instance with 104,814 points. The improvement over 2-opt-based memetic algorithms therefore comes from the local search, not from the population. Over 30 runs, HSD-MA reached mean gaps of 0.72% to 3.47% on five TSPLIB instances and 5.08% and 6.85% on `pla7397` and `pla33810`; on the 102 Waterloo VLSI instances, single runs gave a mean gap of 5.88% for the 25 instances with $N \geq 20{,}000$, with a peak working set of 0.9 to 1.2 GiB on the largest instance ($N = 744{,}710$). LKH-3 found much better tours in every configuration we tested, and GA-EAX on the small instances given a few seconds; HSD-MA needed a third to a fifth of the wall-clock time of the quickest LKH-3 configuration on large instances. In a simulated drilling case study, HSD-MA reduced the cycle time by less than 1% relative to a nearest-neighbor tour improved by 2-opt.

Future work will follow three directions. First, the population layer should be combined with edge-preserving recombination such as EAX or partition crossover [22, 26] and tested against the iterated local search baseline introduced here; without such a test, an iterated local search with the three-tier local search is the simpler and, at the budgets studied, better choice. Second, the loss at cluster boundaries could be reduced by re-optimizing windows that straddle cluster boundaries, as LKH-3 does with its border subproblems [19]. Third, the drilling model should include tool sequencing as a clustered traveling salesman problem [87, 88] and be validated on real board data with measured machine parameters.

Declarations

Funding. The author received no specific funding or financial support for this research.

Competing interests. The author declares that there are no financial or non-financial competing interests directly or indirectly related to the work submitted for publication.

Data availability. The benchmark datasets analyzed during the current study (TSPLIB and Waterloo VLSI collections) are openly available in public repositories at https://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ and https://www.math.uwaterloo.ca/tsp/vlsi/. Complete experimental logs and numerical result summaries are available from the corresponding author upon reasonable request.

Code availability. The source code implementation of the HSD-MA framework is available from the corresponding author upon reasonable request.

Author contributions. Minh-Khanh Vu is the sole author of this manuscript and conducted the theoretical formulation, algorithm design, software implementation, empirical benchmarking, statistical analysis, and manuscript preparation.

References

[1] Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, Princeton, NJ, USA (2006)

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

[3] Grötschel, M., Jünger, M., Reinelt, G.: Optimal control of plotting and drilling machines: A case study. Zeitschrift für Operations Research **35**(1), 61–84 (1991) https://doi.org/10.1007/BF01415960

[4] Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science, vol. 840. Springer, Berlin, Heidelberg (1994). https://doi.org/10.1007/3-540-48661-5

[5] Zhu, G.-Y.: Drilling path optimization based on swarm intelligent algorithm. In: 2006 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 193–196. IEEE, Kunming, China (2006). https://doi.org/10.1109/ROBIO.2006.340357

[6] Onwubolu, G.C., Clerc, M.: Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. International Journal of Production Research **42**(3), 473–491 (2004) https://doi.org/10.1080/00207540310001614150

[7] Reinelt, G.: TSPLIB 95. Documentation, Universität Heidelberg, https://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/tsp95.pdf. Accessed 6 October 2026 (1995)

[8] Dewil, R., Küçükoğlu, İ., Luteyn, C., Cattrysse, D.: A critical review of multi-hole drilling path optimization. Archives of Computational Methods in Engineering **26**(2), 449–459 (2019) https://doi.org/10.1007/s11831-018-9251-x

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

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

[11] Or, İ.: Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. PhD thesis, Northwestern University, Evanston, IL, USA (1976)

[12] Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research **21**(2), 498–516 (1973) https://doi.org/10.1287/opre.21.2.498

[13] Helsgaun, K.: An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research **126**(1), 106–130 (2000) https://doi.org/10.1016/S0377-2217(99)00284-2

[14] Helsgaun, K.: General k-opt submoves for the Lin–Kernighan TSP heuristic. Mathematical Programming Computation **1**(2–3), 119–163 (2009) https://doi.org/10.1007/s12532-009-0004-6

[15] Helsgaun, K.: An extension of the Lin–Kernighan–Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Technical report, Roskilde University, Roskilde, Denmark (December 2017). https://forskning.ruc.dk/en/publications/an-extension-of-the-lin-kernighan-helsgaun-tsp-solver-for-constra

[16] Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: A case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) Local Search in Combinatorial Optimization, pp. 215–310. John Wiley & Sons, Chichester, UK (1997)

[17] Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A.P. (eds.) The Traveling Salesman Problem and Its Variations, pp. 369–443. Kluwer Academic Publishers, Boston, MA, USA (2002). https://doi.org/10.1007/0-306-48213-4_9

[18] Helsgaun, K.: An effective implementation of K-opt moves for the Lin–Kernighan TSP heuristic. Writings on Computer Science 109, Roskilde University, Roskilde, Denmark (2006). Revised November 20, 2007

[19] Helsgaun, K.: Short Description of the Parameters to LKH-3. LKH-3 documentation, webhotel4.ruc.dk/~keld/research/LKH-3. Accessed 7 October 2026 (2017)

[20] Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 616–621 (1996). https://doi.org/10.1109/ICEC.1996.542671

[21] Nagata, Y.: New EAX crossover for large TSP instances. In: Parallel Problem Solving from Nature – PPSN IX. Lecture Notes in Computer Science, vol. 4193, pp. 372–381 (2006). https://doi.org/10.1007/11844297_38

[22] Nagata, Y., Kobayashi, S.: A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. INFORMS Journal on Computing **25**(2), 346–363 (2013) https://doi.org/10.1287/ijoc.1120.0506

[23] Honda, K., Nagata, Y., Ono, I.: A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs. In: 2013 IEEE Congress on Evolutionary Computation, pp. 1278–1285 (2013). https://doi.org/10.1109/CEC.2013.6557712

[24] Tsai, H.-K., Yang, J.-M., Tsai, Y.-F., Kao, C.-Y.: An evolutionary algorithm for large traveling salesman problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **34**(4), 1718–1729 (2004) https://doi.org/10.1109/TSMCB.2004.828283

[25] Whitley, D., Hains, D., Howe, A.: Tunneling between optima: Partition crossover for the traveling salesman problem. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09), pp. 915–922 (2009). https://doi.org/10.1145/1569901.1570026

[26] Tinós, R., Whitley, D., Ochoa, G.: A new generalized partition crossover for the traveling salesman problem: Tunneling between local optima. Evolutionary Computation **28**(2), 255–288 (2020) https://doi.org/10.1162/evco_a_00254

[27] Sanches, D., Whitley, D., Tinós, R.: Building a better heuristic for the traveling salesman problem: Combining edge assembly crossover and partition crossover. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17), pp. 329–336 (2017). https://doi.org/10.1145/3071178.3071305

[28] Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems (NeurIPS 2015), vol. 28, pp. 2692–2700 (2015)

[29] Kool, W., Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (ICLR 2019) (2019)

[30] Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: POMO: Policy

optimization with multiple optima for reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS 2020), vol. 33, pp. 21188–21198 (2020)

[31] Pan, X., Jin, Y., Ding, Y., Feng, M., Zhao, L., Song, L., Bian, J.: H-TSP: Hierarchically solving the large-scale traveling salesman problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, pp. 9345–9353 (2023). https://doi.org/10.1609/aaai.v37i8.26120

[32] Ye, H., Wang, J., Liang, H., Cao, Z., Li, Y., Li, F.: GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-24), vol. 38, pp. 20284–20292 (2024). https://doi.org/10.1609/aaai.v38i18.30009

[33] Zheng, Z., Zhou, C., Tong, X., Yuan, M., Wang, Z.: UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2024), pp. 6081–6125 (2024). https://doi.org/10.52202/079017-0197

[34] Zhou, S., Ding, Y., Zhang, C., Cao, Z., Jin, Y.: DualOpt: A dual divide-and-optimize algorithm for the large-scale traveling salesman problem. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, pp. 27178–27186 (2025). https://doi.org/10.1609/aaai.v39i25.34926

[35] Joshi, C.K., Cappart, Q., Rousseau, L.-M., Laurent, T.: Learning the travelling salesperson problem requires rethinking generalization. Constraints **27**(1–2), 70–98 (2022) https://doi.org/10.1007/s10601-022-09327-y

[36] Bengio, Y., Lodi, A., Prouvost, A.: Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research **290**(2), 405–421 (2021) https://doi.org/10.1016/j.ejor.2020.07.063

[37] Cappart, Q., Chételat, D., Khalil, E.B., Lodi, A., Morris, C., Veličković, P.: Combinatorial optimization and reasoning with graph neural networks. Journal of Machine Learning Research **24**(130), 1–61 (2023)

[38] Reinelt, G.: Fast heuristics for large geometric traveling salesman problems. ORSA Journal on Computing **4**(2), 206–217 (1992) https://doi.org/10.1287/ijoc.4.2.206

[39] Jünger, M., Reinelt, G., Rinaldi, G.: The traveling salesman problem. In: Network Models. Handbooks in Operations Research and Management Science, vol. 7, pp. 225–330. Elsevier, ??? (1995). https://doi.org/10.1016/S0927-0507(05)80121-5

[40] Bartholdi, J.J., Platzman, L.K.: An $O(N \log N)$ planar travelling salesman heuristic based on spacefilling curves. Operations Research Letters **1**(4), 121–125 (1982) https://doi.org/10.1016/0167-6377(82)90012-8

[41] Platzman, L.K., Bartholdi, J.J.: Spacefilling curves and the planar travelling salesman problem. Journal of the ACM **36**(4), 719–737 (1989) https://doi.org/10.1145/76359.76361

[42] Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the TSP incorporating local search heuristics. Operations Research Letters **11**(4), 219–224 (1992) https://doi.org/10.1016/0167-6377(92)90028-2

[43] Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech Concurrent Computation Program C3P Report 826, California Institute of Technology (1989)

[44] Radcliffe, N.J., Surry, P.D.: Formal memetic algorithms. In: Evolutionary Computing. Lecture Notes in Computer Science, vol. 865, pp. 1–16. Springer, Berlin, Heidelberg (1994). https://doi.org/10.1007/3-540-58483-8_1

[45] Whitley, D., Gordon, V.S., Mathias, K.: Lamarckian evolution, the Baldwin effect and function optimization. In: Parallel Problem Solving from Nature – PPSN III. Lecture Notes in Computer Science, vol. 866, pp. 5–15. Springer, Berlin, Heidelberg (1994). https://doi.org/10.1007/3-540-58484-6_245

[46] Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation **9**(5), 474–488 (2005) https://doi.org/10.1109/TEVC.2005.850260

[47] Jian, S.-J., Hsieh, S.-Y.: A niching regression adaptive memetic algorithm for multimodal optimization of the Euclidean traveling salesman problem. IEEE Transactions on Evolutionary Computation **27**(5), 1413–1426 (2023) https://doi.org/10.1109/TEVC.2022.3211954

[48] Al-Janan, D.H., Liu, T.-K.: Path optimization of CNC PCB drilling using hybrid Taguchi genetic algorithm. Kybernetes **45**(1), 107–125 (2016) https://doi.org/10.1108/K-03-2015-0069

[49] Dalavi, A.M., Pawar, P.J., Singh, T.P.: Tool path planning of hole-making operations in ejector plate of injection mould using modified shuffled frog leaping algorithm. Journal of Computational Design and Engineering **3**(3), 266–273 (2016) https://doi.org/10.1016/j.jcde.2016.04.001

[50] Lim, W.C.E., Kanagaraj, G., Ponnambalam, S.G.: PCB drill path optimization by combinatorial cuckoo search algorithm. The Scientific World Journal **2014**, 264518 (2014) https://doi.org/10.1155/2014/264518

[51] Dhouib, S., Zouari, A.: Adaptive iterated stochastic metaheuristic to optimize holes drilling path in manufacturing industry: The Adaptive-Dhouib-Matrix-3 (A-DM3). Engineering Applications of Artificial Intelligence **120**, 105898 (2023) https://doi.org/10.1016/j.engappai.2023.105898

[52] Zainal Abidin, N.W., Ab Rashid, M.F.F., Nik Mohamed, N.M.Z.: A review of multi-holes drilling path optimization using soft computing approaches. Archives of Computational Methods in Engineering **26**(1), 107–118 (2019) https://doi.org/10.1007/s11831-017-9228-1

[53] Mamdouh, M., Khalid, A., Mahmoud, R., Desouki, O., Abdellatif, S.O.: A fully automated drilling machine for printed circuit boards with superior path optimization. Scientific Reports **15**(1), 40367 (2025) https://doi.org/10.1038/s41598-025-25707-9

[54] Karp, R.M.: Probabilistic analysis of partitioning algorithms for the traveling-salesman problem in the plane. Mathematics of Operations Research **2**(3), 209–224 (1977) https://doi.org/10.1287/moor.2.3.209

[55] Valenzuela, C.L., Jones, A.J.: Evolutionary divide and conquer (I): A novel genetic approach to the TSP. Evolutionary Computation **1**(4), 313–333 (1993) https://doi.org/10.1162/evco.1993.1.4.313

[56] Ding, C., Cheng, Y., He, M.: Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. Tsinghua Science and Technology **12**(4), 459–465 (2007) https://doi.org/10.1016/S1007-0214(07)70068-8

[57] Mulder, S.A., Wunsch, D.C.: Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks. Neural Networks **16**(5–6), 827–832 (2003) https://doi.org/10.1016/S0893-6080(03)00130-8

[58] Taillard, É.D., Helsgaun, K.: POPMUSIC for the travelling salesman problem. European Journal of Operational Research **272**(2), 420–429 (2019) https://doi.org/10.1016/j.ejor.2018.06.039

[59] Taillard, É.D.: A linearithmic heuristic for the travelling salesman problem. European Journal of Operational Research **297**(2), 442–450 (2022) https://doi.org/10.1016/j.ejor.2021.05.034

[60] Cook, W., Seymour, P.: Tour merging via branch-decomposition. INFORMS Journal on Computing **15**(3), 233–248 (2003) https://doi.org/10.1287/ijoc.15.3.233.16078

[61] Rocki, K., Suda, R.: Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem. In: 2012 International Conference on High Performance Computing & Simulation (HPCS), pp. 489–495. IEEE, Madrid, Spain (2012). https://doi.org/10.1109/HPCSim.2012.6266963

[62] Delaunay, B.: Sur la sphère vide. Bulletin de l'Académie des Sciences de l'URSS, Classe des Sciences Mathématiques et Naturelles **7**(6), 793–800 (1934)

[63] Shamos, M.I., Hoey, D.: Closest-point problems. In: 16th Annual Symposium on Foundations of Computer Science (SFCS 1975), pp. 151–162. IEEE, ??? (1975). https://doi.org/10.1109/SFCS.1975.8

[64] Kawashima, A., Sugai, Y.: A method of using sub graph based on the characteristics of the optima for the TSPs. IEEJ Transactions on Electronics, Information and Systems **131**(2), 311–319 (2011) https://doi.org/10.1541/ieejeiss.131.311

[65] Letchford, A.N., Pearson, N.A.: Good triangulations yield good tours. Computers & Operations Research **35**(2), 638–647 (2008) https://doi.org/10.1016/j.cor.2006.03.025

[66] Xu, X., Li, J., Zhou, M.: Delaunay-triangulation-based variable neighborhood search to solve large-scale general colored traveling salesman problems. IEEE Transactions on Intelligent Transportation Systems **22**(3), 1583–1593 (2021) https://doi.org/10.1109/TITS.2020.2972389

[67] Paul, P.V., Moganarangan, N., Kumar, S.S., Raju, R., Vengattaraman, T., Dhavachelvan, P.: Performance analyses over population seeding techniques of the permutation-coded genetic algorithm: An empirical study based on traveling salesman problems. Applied Soft Computing **32**, 383–402 (2015) https://doi.org/10.1016/j.asoc.2015.03.038

[68] Applegate, D., Cook, W., Rohe, A.: Chained Lin–Kernighan for large traveling salesman problems. INFORMS Journal on Computing **15**(1), 82–92 (2003) https://doi.org/10.1287/ijoc.15.1.82.15157

[69] Lam, S.K., Pitrou, A., Seibert, S.: Numba: A LLVM-based Python JIT compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (LLVM '15), pp. 1–6. ACM, Austin, TX, USA (2015). https://doi.org/10.1145/2833157.2833162

[70] Cook, W.: VLSI Data Sets. https://www.math.uwaterloo.ca/tsp/vlsi/. Collection of 102 instances provided by André Rohe, based on VLSI data from the Forschungsinstitut für Diskrete Mathematik, Universität Bonn; accessed 6 October 2026

[71] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7**, 1–30 (2006)

[72] Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation **1**(1), 3–18 (2011) https://doi.org/10.1016/j.swevo.2011.02.002

[73] Eftimov, T., Korošec, P., Seljak, B.K.: A novel approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics. Information

Sciences **417**, 186–215 (2017) https://doi.org/10.1016/j.ins.2017.07.015

[74] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, USA (1989)

[75] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE, Perth, WA, Australia (1995). https://doi.org/10.1109/ICNN.1995.488968

[76] Clerc, M.: Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: Onwubolu, G.C., Babu, B.V. (eds.) New Optimization Techniques in Engineering. Studies in Fuzziness and Soft Computing, vol. 141, pp. 219–239. Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-39930-8_8

[77] Stützle, T., Hoos, H.H.: MAX–MIN ant system. Future Generation Computer Systems **16**(8), 889–914 (2000) https://doi.org/10.1016/S0167-739X(00)00043-1

[78] Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, pp. 320–353. Kluwer Academic Publishers, Boston, MA (2003). https://doi.org/10.1007/0-306-48056-5_11

[79] Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Concorde TSP Benchmarks (from 1999). https://www.math.uwaterloo.ca/tsp/concorde/benchmarks/bench99.html. Accessed 6 October 2026

[80] Nagata, Y.: GA-EAX: Genetic Algorithm with Edge Assembly Crossover for the Traveling Salesman Problem. https://github.com/nagata-yuichi/GA-EAX. Source code and README of GA_EAX_1.0, accessed October 2026

[81] Altintas, Y., Verl, A., Brecher, C., Uriarte, L., Pritschow, G.: Machine tool feed drives. CIRP Annals **60**(2), 779–796 (2011) https://doi.org/10.1016/j.cirp.2011.05.010

[82] Via Mechanics, Ltd.: ND-6Y220E HDI General-Purpose 6-Stations Drilling Machine for PCBs: Primary Specification. Product datasheet, https://en.viamechanics.com/wp-content/uploads/2014/10/nd-6y220e_low.pdf. Accessed 6 October 2026 (2014)

[83] LPKF Laser & Electronics: LPKF ProtoMat E44: Product Flyer. https://www.lpkf.com/fileadmin/mediafiles/user_upload/products/pdf/DQ/flyer_lpkf_protomat_e44_en.pdf. Accessed 6 October 2026

[84] LPKF Laser & Electronics: LPKF ProtoMat S64: Product Flyer. https://www.lpkf.com/fileadmin/mediafiles/user_upload/products/pdf/DQ/flyer_lpkf_protomat_s64_en.pdf. Accessed 6 October 2026

[85] Bungard Elektronik: CCD/ATC/XL CNC Machine: Technical Data. https://www.bungard.de/en/machines/cnc-machines/ccd-atc-xl. Accessed 6 October 2026

[86] Erkorkmaz, K., Altintas, Y.: High speed CNC system design. part I: Jerk limited trajectory generation and quintic spline interpolation. International Journal of Machine Tools and Manufacture **41**(9), 1323–1345 (2001) https://doi.org/10.1016/S0890-6955(01)00002-5

[87] Chisman, J.A.: The clustered traveling salesman problem. Computers & Operations Research **2**(2), 115–119 (1975) https://doi.org/10.1016/0305-0548(75)90015-5

[88] Ancău, M.: The optimization of printed circuit board manufacturing by improving the drilling process productivity. Computers & Industrial Engineering **55**(2), 279–294 (2008) https://doi.org/10.1016/j.cie.2007.12.008

[89] Kolahan, F., Liang, M.: Optimization of hole-making operations: A tabu-search approach. International Journal of Machine Tools and Manufacture **40**(12), 1735–1753 (2000) https://doi.org/10.1016/S0890-6955(00)00024-9

[90] Guibas, L., Stolfi, J.: Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics **4**(2), 74–123 (1985) https://doi.org/10.1145/282918.282923

[91] McMullen, P.: The maximum numbers of faces of a convex polytope. Mathematika **17**(2), 179–184 (1970) https://doi.org/10.1112/S0025579300002850

[92] Amenta, N., Attali, D., Devillers, O.: Complexity of Delaunay triangulation for points on lower-dimensional polyhedra. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 1106–1113. SIAM, New Orleans, LA, USA (2007)