

Beyond Endpoint Quality: A Phase-Aware Analysis of Search Dynamics in Triangle Reassembly

Damir Hasanspahić(✉), Adis Alihodžić, and Damir Hasić

University of Sarajevo, Sarajevo, Bosnia and Herzegovina
damir.hasanspahic@gf.unsa.ba, {adis.alihodzic,d.hasic}@pmf.unsa.ba

Abstract. This paper studies the reassembly of triangles into the unit square from scattered Delaunay triangulations. Each triangle is independently translated and rotated, all adjacency information is removed, and the objective is to reconstruct a feasible non-overlapping arrangement that maximizes coverage. We compare Variable Neighborhood Search, Iterated Local Search, Artificial Bee Colony, IterativeSort, and IterativeRand under the same permutation representation, geometric evaluation operator, feasibility checks, and a budget of 21 evaluation calls on instances with 94–994 triangles. The analysis tracks phase-end best-so-far values, improvement incidence and gains, within-phase score levels, and the point at which the run-best value is reached before stagnation. Variable Neighborhood Search gives the strongest overall phase profile and remains the most reliable default choice, especially on larger instances. Iterated Local Search concentrates improvements earliest; Artificial Bee Colony shows the most persistent late-phase improvement behavior; and IterativeSort serves as a good heuristic starting point, especially in the middle of the search. Comparable terminal quality can therefore conceal materially different phase-wise search dynamics.

Keywords: triangle reassembly · Delaunay triangulation · geometric reconstruction · computational geometry · combinatorial optimization · metaheuristics · search dynamics · phase-aware analysis

1 Introduction

Packing and geometric reassembly problems study how finitely many geometric objects can be placed inside a bounded region under containment and non-overlap constraints while optimizing a global objective. When the pieces are irregular and their original neighborhood structure is unavailable, the search space combines a continuous geometric component with a discrete combinatorial one, yielding the nonconvex and fragmented behavior typical of hard packing problems [3,6,7,13,16,22,23]. In evaluation-intensive settings, the trajectory of progress also becomes important: under a limited budget of expensive geometric

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026, LNCS 16767, pp. 130–143, 2027.
https://doi.org/10.1007/978-3-032-30491-9_9

evaluations, endpoint quality alone can conceal substantial differences in how methods use the available search budget.

We study this issue for a triangle reassembly problem induced by Delaunay triangulations of the unit square. A point set in the square is triangulated, each triangle is then independently scattered by an arbitrary translation and an arbitrary rotation about some point in the plane, and all adjacency relations are erased. The task is to reconstruct as much of the original tiling as possible without overlap. Although the pieces come from an exact tiling, the lost combinatorial structure must be recovered implicitly from geometry alone. The problem therefore belongs to a combinatorial–geometric family of NP-hard problems [3,6,13,16] and exhibits a reconstruction setting in which the order of geometric attempts is particularly consequential.

Beyond its role as a controlled benchmark, the Delaunay-based reconstruction setting abstracts a class of situations in which irregular pieces must be placed or reassembled when original placement or adjacency information is unavailable. Related constrained-placement problems arise in irregular two-dimensional stock cutting and nesting, where irregular pieces are arranged within stock material without overlap and, in many applications, under rotation restrictions [10]. The geometric handling of such layouts is commonly supported by no-fit polygon constructions, including robust Minkowski-sum-based procedures for complex cases with holes, slits, and lock and key positions [8]. Related geometry-driven reassembly problems occur in 3D fragment reassembly for archaeology and forensic investigation, including the computer-aided restoration of damaged artifacts and evidence, where template guidance is integrated with fracture-region matching to recompose fragmented objects [29]. The simplified Delaunay setting used here is therefore not a direct model of these applications; rather, it provides a controlled way to isolate permutation and feasibility effects that also arise in broader constrained-placement and reassembly problems.

The present paper builds on the reconstruction setting introduced in [5]. Earlier experiments there considered instances induced by at most 60 points, i.e., up to 114 triangles. Here we retain the same reconstruction perspective but study instances induced by 50, 100, 200, 300, and 500 points, i.e., 94–994 triangles, under a shared geometric evaluation operator. Rather than altering the geometric core, we ask how different permutation-search paradigms behave when the evaluation mechanism, feasibility logic, and evaluation-call budget are all held fixed.

Our empirical focus is the phase structure of progress over the evaluation budget. The budget is partitioned into three equal call segments, and the comparison is organized around phase-end best-so-far profiles, within-phase score levels, productive-run incidence, improvement counts, jump amplitudes, the evaluation index k_{best} at which the eventual run-best value is reached, and the remaining stall length thereafter. Final coverage is reported as a supporting summary, whereas the primary focus is on when progress is produced and how it is distributed across phases.

Contributions

- We extend the scattered-triangulation reconstruction setting of [5] from the earlier limit of 60 points (114 triangles) to instances induced by up to 500 points (994 triangles) under a common unit-square model.
- We present a controlled comparative evaluation of three canonical metaheuristics and two lightweight heuristic methods within a unified experimental framework, using the same permutation representation, geometric evaluation operator, length-compatibility prior, and evaluation budget.
- We structure the empirical analysis around phase dynamics and show that methods with similar terminal coverage can differ substantially in when and how they generate, preserve, and exhaust progress throughout the run.

2 Related Work

Irregular packing and nesting form a broad class of problems in combinatorial optimization and computational geometry [7,22,23]. Polygon packing has been studied from theoretical, heuristic, and industrial perspectives, including maximum-packing heuristics [26] and nesting procedures based on no-fit polygons and Minkowski-sum constructions [7,8,10]. Across these settings, geometric feasibility and combinatorial choice remain tightly coupled, which is precisely the structural backdrop of the present reassembly problem.

Classical packing variants such as packing identical simple polygons, bin packing, strip packing, rectangle packing, and specialized triangle packing are NP-hard [3,6,13,16]. More recently, broad classes of two-dimensional packing problems with rotations have been shown to be $\exists\mathbb{R}$-complete [2], indicating a level of geometric hardness that goes beyond standard discrete-complexity formulations. These results place the present reconstruction task within a difficult combinatorial–geometric family.

Within the triangle-packing literature, graph-theoretic formulations study the existence of vertex-disjoint triangles as well as packing and covering variants [11,14,18,21,27]. In geometric and application-oriented settings, exact branch-and-bound procedures [1] and heuristic methods [12] have also been proposed. All of these formulations typically assume arbitrary input pieces or settings in which structural relations remain directly available.

Our setting differs from these formulations in one key respect: the pieces are derived from a coherent Delaunay triangulation that exactly tiles the container, but the adjacency structure is erased before the search begins, so compatibility must be reconstructed from geometry alone. This reconstruction perspective was introduced for scattered triangulations in [5], where geometry-driven constructive strategies were studied on instances up to 60 points, i.e., 114 triangles. The present paper follows that line of work at a larger scale and shifts the empirical emphasis from endpoint quality alone to the fixed-budget phase structure of search progress under a shared geometric evaluation operator.

3 Methods

3.1 Problem Definition

We restate the model of [5], specialized to the unit square. Let $R = [0,1] \times [0,1]$ be the unit square, with area $A(R) = 1$. Let $X = \{x_1, \ldots, x_n\} \subset R$ contain the four corners of R, with the remaining $n-4$ points strictly in the interior and no three points collinear. Let $T = \{T_1, \ldots, T_N\}$ be the Delaunay triangulation of X; then T partitions R into non-overlapping triangles, and since the convex hull has four vertices, $N = 2n - 6$ [4,9,15]. Each triangle T_i is then scattered by an independent planar rigid motion, i.e., by an arbitrary translation and an arbitrary rotation about some point, producing an unordered set $T' = \{T'_1, \ldots, T'_N\}$ with all positional and adjacency information removed [5].

A feasible reconstruction consists of a subset $S \subseteq T'$ together with a placement mapping $f : S \to \mathbb{R}^2$ that assigns to each triangle a rigid motion, such that

$$f(T'_i) \cap f(T'_j) = \emptyset \quad \forall\, T'_i \neq T'_j \in S, \tag{1}$$

$$f(T'_i) \subseteq R \quad \forall\, T'_i \in S. \tag{2}$$

The objective is to maximize covered area. Since $A(R) = 1$, the percentage coverage is

$$P(S) = 100 \cdot \sum_{T'_i \in S} A(T'_i), \tag{3}$$

and, because the triangles of T tile R exactly, the optimum value is 100%.

3.2 Shared Geometric Evaluation Operator

All compared methods search in the same solution space, namely permutations of the triangle set, and every candidate permutation is assessed by the same deterministic *Geometric Evaluation Operator*. For a given permutation π, the operator processes triangles in the prescribed order and incrementally attempts to build a feasible layout inside the unit square; its output is the resulting coverage score $P(\pi)$. Because the evaluation operator is fixed for all methods, the comparison isolates search behavior rather than geometric processing or feasibility handling.

The first triangle in the permutation serves as an anchor and is placed at an arbitrary feasible position inside the unit square, after which subsequent triangles are tentatively aligned to exposed edges of the current partial layout. The operator follows an incremental edge-attachment principle in the spirit of the S2/S3 constructive schemes in [5]. A candidate placement is accepted only if it remains inside the container and does not overlap previously placed triangles. To control geometric branching on larger instances, candidate edge pairs are considered only when $\ell_{\max} \leq 1.5\,\ell_{\min}$, i.e., when the longer edge is at most 50% longer than the shorter one. This filter is applied uniformly to all methods.

To keep evaluation stable up to $N \approx 10^3$, the operator uses uniform-grid spatial hashing for localized overlap queries, prioritized inspection of the 3×3

neighboring cells around the candidate location, coarse filters such as axis-aligned bounding-box checks before exact intersection testing, and pair-skip caching for edge pairs that have already failed within the current partial layout. These mechanisms are shared by all methods and leave the objective function and admissibility conditions unchanged.

Each evaluation call returns one scalar score $P(\pi)$, and all phase-aware descriptors are defined on the resulting ordered sequence of evaluation calls.

3.3 Compared Algorithms

All methods considered in this study operate in the same permutation space and are evaluated by the shared Geometric Evaluation Operator introduced in Sect. 3.2. The comparison is therefore designed to isolate search policy under a fixed evaluation mechanism. To preserve fairness, all selected metaheuristics are used in minimal canonical implementations, and their control parameters retain their standard roles.

Variable Neighborhood Search (VNS). VNS follows the classical principle of systematic neighborhood change combined with local improvement [17,28]. The search state is a permutation, and the method traverses neighborhoods of increasing scope through standard permutation perturbations. Whenever an improving solution is found, the search returns to the first neighborhood, in accordance with the standard VNS scheme. In the present study, VNS serves as the variable-neighborhood search component under the common evaluation setting.

Iterated Local Search (ILS). ILS follows the standard perturb–improve–accept pattern [24,25]. Starting from a current permutation, the method applies a perturbation, performs local improvement on the perturbed solution, and then decides whether the resulting local optimum replaces the incumbent one. In this setting, ILS provides a compact trajectory-based intensification baseline for the shared permutation landscape.

Artificial Bee Colony (ABC). ABC follows the standard employed/onlooker/scout organization [19,20]. In the present setting, candidate permutations act as food sources, and their quality is determined by the score returned by the shared geometric evaluation operator. The employed and onlooker phases explore modified permutations around existing candidates, whereas the scout phase restores diversification when progress stalls. ABC therefore represents the population-based search component in the comparison.

IterativeSort. IterativeSort is a deterministic heuristic reference derived from the descending-area ordering principle used in the earlier S2-style construction strategy [5]. The triangles are sorted in non-increasing order of area, and the

resulting permutation is submitted directly for evaluation. Its role is to provide a simple structured ordering rule without adaptive search, population dynamics, or iterative refinement.

IterativeRand. IterativeRand is a low-information heuristic reference in which candidate solutions are generated as random permutations of the triangle set. By removing the explicit size-based ordering signal present in IterativeSort, it provides a reference level for performance obtainable without structured guidance. It is therefore useful for separating the contribution of deliberate search from the contribution of the fixed evaluation mechanism itself.

The selected metaheuristics were chosen so as to span complementary search paradigms within the same permutation-based framework. Their joint consideration provides a sufficiently broad, though not exhaustive, view of the behavior of different metaheuristic search paradigms.

4 Experimental Design

Instance Sizes. We consider five point-set sizes, $n \in \{50, 100, 200, 300, 500\}$, which induce triangulation sizes $N = 2n - 6 \in \{94, 194, 394, 594, 994\}$.

Protocol. All methods are evaluated under the same regime, with computational effort controlled exclusively by the number of calls to the Geometric Evaluation Operator. Each run therefore uses the same fixed budget $B = 21$ evaluation calls, partitioned into three equal segments $F1 = \{1, \ldots, 7\}$, $F2 = \{8, \ldots, 14\}$, and $F3 = \{15, \ldots, 21\}$. These segments define the three analysis phases used throughout the paper and place progress directly on consumed evaluation budget, making it visible whether gains are concentrated early, sustained through F2, or preserved into F3. The choice $B = 21$ is supported not only by the phase-analysis design but also by supplementary budget-extension checks on the smallest instance, where a threefold larger horizon did not yield systematic or proportionate coverage gains across methods. In view of the problem complexity and the execution time required by larger instances, the number of independent runs was adjusted stepwise according to instance size: 20 runs for $N \in \{94, 194\}$, 10 runs for $N \in \{394, 594\}$, and 5 runs for $N = 994$. Reported summaries aggregate method–instance–run observations either globally or by size class, depending on the corresponding table or figure.

Phase-Aware Descriptors. Let s_k denote the score returned by the k-th evaluation call, and let $b_k = \max_{1 \le t \le k} s_t$ denote the best-so-far value after k evaluations. From these sequences, we derive the following descriptors. The terminal run quality is defined as $P = b_{21}$. For each phase F_j, we record the phase-end best-so-far value $F_{end,j}$, the number of strict best-so-far improvements $N_{imp,j}$, the productive-run incidence $Inc_j = \Pr(N_{imp,j} \ge 1)$, and the largest best-so-far jump $D_{max,j}$. To characterize the level of sampled solutions within a phase, we also record the phase mean and phase maximum, denoted by $S_{mean,j}$ and

$S_{max,j}$. To describe where progress is ultimately realized along the evaluation budget, we further record the evaluation-call index at which the eventual run-best value is last attained, denoted by k_{best}, together with the subsequent stall length, i.e., the number of remaining evaluation calls after k_{best} without further strict improvement. For phases $F2$ and $F3$, we also report the aggregate phase gain, defined as $Gain_j = F_{end,j} - F_{end,j-1}$. Within each productive phase, we summarize the local positions of the first and last strict improvement, denoted by imp@k and lastImp@k, where local positions are counted from 1 to 7 inside the corresponding phase segment.

These descriptors separate three complementary aspects of behavior: attained best-so-far quality, within-phase score levels, and where improvements occur within the evaluation budget. Jump-amplitude analysis is reported for F2 and F3, the phases in which nonzero best-so-far jumps are observed.

For inferential support, we use a light testing layer consisting of global Kruskal–Wallis tests and selected pairwise contrasts with Holm adjustment.

Implementation and Execution Environment. All algorithms were implemented in Python 3.11 and executed in single-threaded mode. All experiments were run on one x86-64 workstation equipped with an Intel® Core i5-3360M processor at 2.80 GHz and 8 GB of RAM. Because every method uses the same preprocessing pipeline, feasibility logic, and shared geometric evaluation operator under the same hardware and software configuration, the reported differences are attributable to search behavior within the common evaluation framework rather than to variation in the computational environment.

5 Discussion

All score-like coverages are reported in percentages, so gains and gaps are interpreted in percentage points. Phase-resolved descriptors refer to positions within the sequence of evaluation calls.

Accordingly, the phase-resolved results are interpreted not merely as rankings of terminal coverage, but also as diagnostics of search roles: strong opening levels indicate robust initial construction, middle-phase gains indicate useful exploitation after the opening layout, and late gains indicate remaining search reserve under a tight evaluation budget.

The highest observed final coverages are attained on $N = 994$ for all methods: VNS reaches 92.92%, ILS 92.50%, IterativeSort 92.44%, ABC 92.43%, and IterativeRand 86.92%. Figure 1 shows that mean final coverage also converges as N grows; on $N = 994$, the four strongest methods lie within 0.29% points in mean P (VNS 92.51%, ILS 92.39%, IterativeSort 92.28%, ABC 92.23%). Mean runtime grows with size in the expected way, from about 6.5–8.0 s at $N = 94$ to about 1520–1640 s at $N = 994$. The main separation therefore lies in where best-so-far progress occurs across the 21 evaluation calls.

Table 1 shows that VNS attains the highest mean final coverage (88.76%), followed by IterativeSort (88.38%), while ILS reaches the final best-so-far value

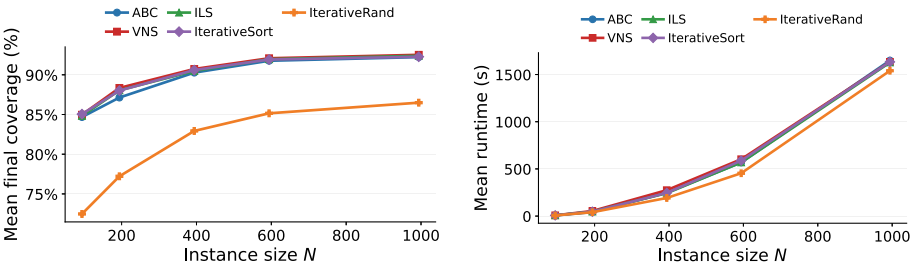

Fig. 1. Mean final coverage and mean runtime by instance size.

Table 1. Overall mean (std) final coverage and evaluation-call indices across all instances.

Method	P (%)	k_{best}	Stall
ABC	87.85 (2.99)	12.05	8.95
VNS	88.76 (2.92)	9.17	11.83
ILS	87.84 (2.99)	8.77	12.23
IterativeSort	88.38 (2.69)	10.38	10.62
IterativeRand	79.77 (5.24)	10.77	10.23

at the earliest mean evaluation-call index ($k_{best} = 8.77$). ABC reaches that point latest ($k_{best} = 12.05$) and therefore leaves the shortest trailing stall. These three descriptors already separate the main profiles: VNS is strongest in level, ILS is earliest in consolidation, and ABC preserves the largest post-opening reserve.

Opening Segment. Figure 2, Table 2, and Table 4 show that all methods remain productive in F_1, so the separation in the opening phase arises primarily through the level of the best-so-far trajectory rather than through the mere incidence of productive starts. Within that common setting, VNS builds the strongest opening level, with $F_{end,1} = 88.09\%$, and also leads the sampled-score descriptors in the first phase, with $S_{mean,1} = 85.01\%$ and $S_{max,1} = 88.09\%$. It then retains the highest phase-end value through F_2 and F_3. The same figure shows that VNS also leads $S_{mean,j}$ in all three phases, indicating that its advantage is not limited to isolated best-so-far updates. The same ordering is preserved by $S_{max,j}$, so within-phase maxima do not overturn the ranking established by $F_{end,j}$ and $S_{mean,j}$. IterativeSort (87.73%) and ILS (87.43%) remain close on $F_{end,1}$, which explains why the deterministic baseline already matches the leading metaheuristics after the first seven calls. ABC also reaches a competitive opening endpoint, with $F_{end,1} = 86.77\%$, but its markedly lower $S_{mean,1} = 77.46\%$ points to a wider gap between sampled solutions and the best-so-far path in the opening segment. IterativeRand records the highest early activity, with $N_{imp,1} = 2.81$, yet its opening score level remains clearly lower, at $F_{end,1} = 79.18\%$, showing that more frequent early improvements do not by themselves translate into

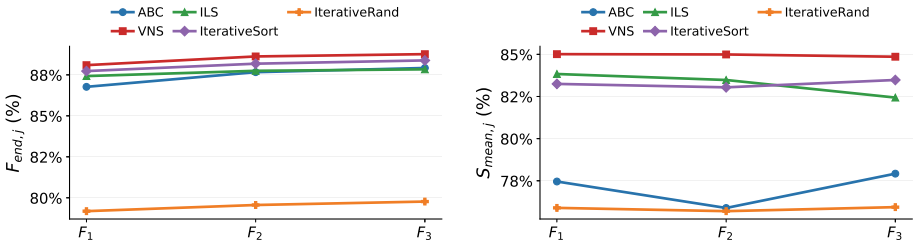

Fig. 2. Phase-end best-so-far levels and within-phase mean scores.

Table 2. Phase-end best-so-far levels and aggregate phase gains.

Method	$F_{end,1}$	$F_{end,2}$	$F_{end,3}$	$Gain_2$	$Gain_3$
ABC	86.77 (4.17)	87.66 (3.15)	87.92 (2.97)	0.89 (1.85)	0.26 (0.54)
VNS	88.09 (3.37)	88.62 (3.01)	88.76 (2.92)	0.53 (1.11)	0.14 (0.44)
ILS	87.43 (3.20)	87.75 (3.00)	87.84 (2.99)	0.32 (0.78)	0.09 (0.25)
IterativeSort	87.73 (3.09)	88.18 (2.86)	88.38 (2.69)	0.45 (0.98)	0.20 (0.73)
IterativeRand	79.18 (5.54)	79.56 (5.36)	79.77 (5.24)	0.38 (0.73)	0.21 (0.43)

stronger phase-end quality. Table 3 further refines the same picture: ILS closes productive F_1 progress earliest on average (lastImp@F_1 = 3.74), whereas IterativeSort remains active slightly deeper into the opening segment (4.07). The opening phase therefore separates methods both by attained level and by how quickly early progress is consolidated, while also revealing differences in score dispersion around the leading trajectory.

Table 3. Mean local positions of strict improvements on the phase-local scale 1–7.

Method	lastImp@F_1	imp@F_2	lastImp@F_2	imp@F_3	lastImp@F_3
ABC	3.91	3.79	3.97	5.37	5.57
VNS	3.90	3.03	4.00	3.65	4.00
ILS	3.74	3.56	4.03	4.46	4.92
IterativeSort	4.07	3.36	4.23	3.59	4.15
IterativeRand	4.28	3.64	4.06	3.77	4.05

Middle Segment. Table 5, Table 2, and Table 4 show that F_2 is the main point of behavioural separation among the methods. IterativeSort is the most reliable producer of additional F_2 updates, with the highest $N_{imp,2}$ (0.67) and the highest productive-run incidence Inc_2 (52.5%). ABC, however, converts productive F_2 runs into the largest gain and the largest jump amplitude, leading both $Gain_2$

Table 4. Within-phase score levels and phase maxima.

Method	$S_{mean,1}$	$S_{mean,2}$	$S_{mean,3}$	$S_{max,1}$	$S_{max,2}$	$S_{max,3}$
ABC	77.46	75.89	77.92	86.77	85.81	86.80
	(6.91)	(7.62)	(6.58)	(4.17)	(5.01)	(3.71)
VNS	85.01	84.99	84.86	88.09	88.13	88.01
	(5.77)	(5.69)	(5.66)	(3.37)	(3.37)	(3.37)
ILS	83.83	83.48	82.44	87.43	87.01	86.21
	(5.38)	(5.29)	(7.15)	(3.20)	(3.45)	(4.29)
IterativeSort	83.25	83.04	83.48	87.73	87.64	87.80
	(6.44)	(6.61)	(5.84)	(3.09)	(3.47)	(2.96)
IterativeRand	75.89	75.70	75.94	79.18	78.91	79.00
	(7.61)	(7.75)	(7.65)	(5.54)	(5.90)	(5.81)

(0.89%) and $D_{max,2}$ (0.88%). At the same time, VNS remains the strongest method in terms of absolute level in the middle phase, with the highest values of $F_{end,2} = 88.62\%$, $S_{mean,2} = 84.99\%$, and $S_{max,2} = 88.13\%$. Figure 3 shows where these updates occur within the phase. In F_2, ABC concentrates both first and last productive updates around local position 4, whereas IterativeSort spreads productive events more broadly across positions 3–5. This distinction explains why IterativeSort is the most reliable middle-phase updater, while ABC is the strongest middle-phase converter once an update occurs, with VNS nevertheless preserving the highest overall middle-phase quality level.

Table 5. Productivity, incidence (%), and jump descriptors by phase.

Method	$N_{imp,1}$	$N_{imp,2}$	Inc_2	$N_{imp,3}$	Inc_3	$D_{max,2}$	$D_{max,3}$
ABC	1.68	0.46	42.3	0.42	38.5	0.88	0.25
	(0.69)	(0.57)		(0.59)		(1.78)	(0.54)
VNS	2.46	0.61	44.0	0.24	20.2	0.46	0.13
	(0.99)	(0.81)		(0.51)		(0.94)	(0.40)
ILS	2.66	0.52	43.8	0.21	17.8	0.29	0.08
	(1.15)	(0.65)		(0.47)		(0.68)	(0.22)
IterativeSort	2.57	0.67	52.5	0.31	26.7	0.41	0.18
	(1.08)	(0.74)		(0.54)		(0.90)	(0.67)
IterativeRand	2.81	0.57	47.8	0.35	31.9	0.37	0.21
	(1.09)	(0.67)		(0.54)		(0.73)	(0.42)

Late Segment. The same descriptors show a clearer separation in F_3, where the late segment distinguishes residual improvement reserve from early saturation.

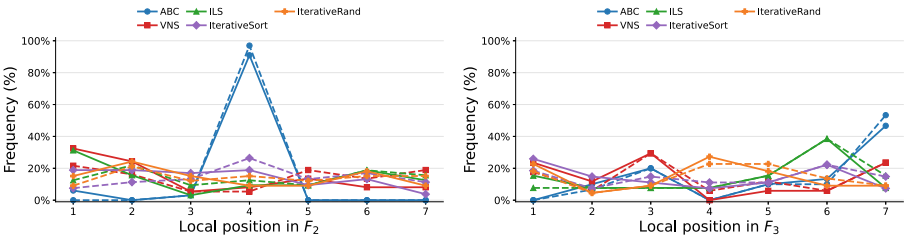

Fig. 3. Local-position frequencies of first (solid) and last (dashed) strict improvements in productive runs of F_2 and F_3.

Table 5, Table 2, and Table 4 show that ABC leads all late-phase productivity measures: $N_{imp,3} = 0.42$, $Inc_3 = 38.5\%$, $Gain_3 = 0.26\%$, and $D_{max,3} = 0.25\%$. Table 3 and Fig. 3 show that this late reserve is also deeper in the phase: ABC records imp@F_3 = 5.37 and lastImp@F_3 = 5.57, both later than the corresponding means of the other methods. VNS still keeps the highest late-phase level in Fig. 2 and remains strongest on the late sampled-score descriptors, with $S_{mean,3} = 84.86\%$ and $S_{max,3} = 88.01\%$, but its late productive incidence is much lower (20.2%). ILS reaches its final best-so-far value early in the budget and correspondingly retains the profile in which most progress is realized earlier, showing the smallest late productive incidence (17.8%). The late segment therefore distinguishes level preservation from residual improvement reserve: VNS preserves the highest level, whereas ABC retains the strongest capacity for additional best-so-far progress.

Instance Scale and Synthesis. Figure 4 shows that ABC's $Gain_2$ advantage is concentrated on the smaller and medium instances: 1.07% at $N = 94$ and 1.64% at $N = 194$, compared with only 0.03%–0.08% on $N = 594$ and $N = 994$. This explains why endpoint coverage converges on the largest instances even though the phase profiles remain distinct. On $N = 994$, mean final coverage is already tightly clustered among VNS, ILS, IterativeSort, and ABC, but Table 1, Table 5, and Table 3 still separate them by where progress is realized: VNS remains strongest in phase-end level, ILS reaches the final best-so-far value earliest, ABC retains the largest post-F_1 improvement reserve, and IterativeSort remains a good deterministic reference through reliable F_2 productivity.

Global Tests. Kruskal–Wallis tests detect between-method differences in phase-end quality ($F_{end,1}$–$F_{end,3}$: $H = 113.76$–130.29, all $p < 0.001$) and in the evaluation-call index at which the final best-so-far value is reached (k_{best}: $H = 12.67$, $p = 0.013$). For the selected phase-oriented pairwise contrasts, Holm-adjusted tests show that ABC and ILS differ both in late-phase gain ($Gain_3$: $p_{Holm} = 0.0060$) and in k_{best} ($p_{Holm} = 0.0060$). These findings remain consistent with the descriptive reading: the methods separate primarily by phase profile and by the evaluation-call indices at which progress is realized, rather than by terminal level alone.

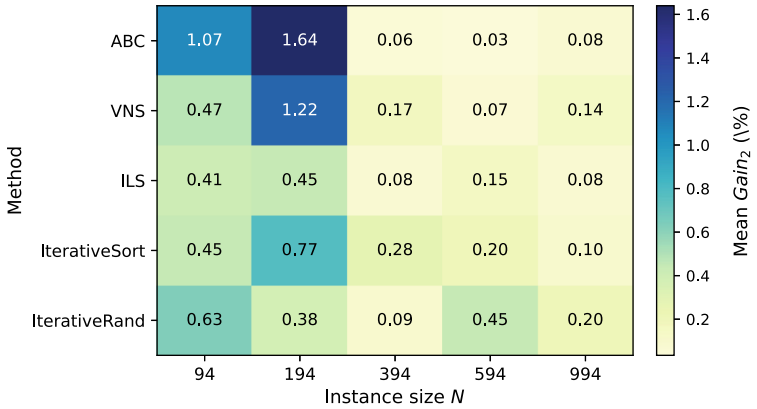

Fig. 4. Mean $Gain_2$ by instance size.

Robustness. A supplementary robustness check on the smallest instance ($N = 94$) under a threefold larger budget ($B = 63$) leads to the same practical conclusion. For the competitive methods, the mean gain realized after F1 remained modest: 0.04% points for ILS, 0.44 for VNS, 0.55 for ABC, and 0.67 for IterativeSort. Some individual runs still improved in F2 or F3, but the added calls did not produce a systematic or proportionate increase in coverage. This supports the use of $B = 21$ as a controlled phase-analysis budget rather than a budget chosen in a regime where substantially larger horizons would reliably transform the outcome.

6 Conclusion

We studied the reassembly of triangles into the unit square from scattered Delaunay triangulations through a controlled phase-aware comparison in which all methods shared the same permutation representation, geometric evaluation operator, admissibility logic, and evaluation-call budget. The benchmark extends the earlier reconstruction setting from instances with up to 114 triangles to instances with up to 994 triangles. Within this common setting, the results are most informative when viewed phase-wise, as the methods differ less in terminal quality than in how improvement unfolds over the evaluation budget. VNS delivers the strongest overall best-so-far profile and remains the most reliable default choice, especially on larger instances. IterativeSort stands out as a good heuristic starting point and the most reliable improver in F2. ILS exhibits the clearest early intensification, but also saturates sooner. ABC retains the strongest late-phase elasticity, with the highest residual productivity in F3 and the largest productive late moves among the competing methods.

Taken together, these findings show that similar terminal quality can conceal substantially different search dynamics.

A similar phase-structured analysis protocol could also provide a useful basis for examining broader families of baseline methods, adaptive hybrids, and related irregular reconstruction or polygon-packing settings where evaluation constitutes a substantial computational cost.

References

1. Abdelsadek, Y., Herrmann, F., Kacem, I., Otjacques, B.: Branch-and-bound algorithm for the maximum triangle packing problem. Comput. Industr. Eng. **81**, 147–157 (2015). https://doi.org/10.1016/j.cie.2014.12.006
2. Abrahamsen, M., Miltzow, T., Seiferth, N.: Framework for ∃ℝ-completeness of two-dimensional packing problems. TheoretiCS **3**, 11 (2024). https://doi.org/10.46298/theoretics.24.11
3. Adamaszek, A., Kociumaka, T., Pilipczuk, M., Pilipczuk, M.: Hardness of approximation for strip packing. ACM Trans. Comput. Theor. **9**(3), 14:1–14:7 (2017). https://doi.org/10.1145/3092026
4. Alihodžić, A.: Exploring Computational Geometry: Theory and Python Implementations. Springer, Cham (2026). https://doi.org/10.1007/978-3-032-06393-9
5. Alihodžic, A., Hasanspahic, D., Tuba, E., Hasic, D.: Triangle packing strategies within a rectangle. In: Proceedings of the IEEE International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA), Boracay, Philippines (2026, to appear). https://doi.org/10.1109/ACDSA67686.2026.11468118, https://ieeexplore.ieee.org/document/11468118
6. Allen, S.R., Iacono, J.: Packing identical simple polygons is NP-hard (2012). https://arxiv.org/abs/1209.5307
7. Bennell, J.A., Oliveira, J.F.: The geometry of nesting problems: a tutorial. Eur. J. Oper. Res. **184**(2), 397–415 (2008). https://doi.org/10.1016/j.ejor.2006.11.038
8. Bennell, J.A., Song, X.: A comprehensive and robust procedure for obtaining the nofit polygon using Minkowski sums. Comput. Oper. Res. **35**(1), 267–281 (2008). https://doi.org/10.1016/j.cor.2006.02.026
9. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77974-2
10. Burke, E.K., Hellier, R.S.R., Kendall, G., Whitwell, G.: Complete and robust no-fit polygon generation for the irregular stock cutting problem. Eur. J. Oper. Res. **179**(1), 27–49 (2007). https://doi.org/10.1016/j.ejor.2006.03.011
11. Caprara, A., Rizzi, R.: Packing triangles in bounded degree graphs. Inf. Process. Lett. **84**(4), 175–180 (2002). https://doi.org/10.1016/S0020-0190(02)00274-0
12. Chen, C.B., He, D.H.: A heuristic method for solving triangle packing problem. J. Zhejiang Univ. Sci. A **6**(6), 565–570 (2005). https://doi.org/10.1631/jzus.2005.A0565
13. Chou, A.: NP-hard triangle packing problems. Research Science Institute manuscript, January 2016. https://math.mit.edu/research/highschool/rsi/documents/2015Chou.pdf
14. Corrádi, K., Hajnal, A.: On the maximal number of independent circuits in a graph. Acta Mathematica Academiae Scientiarum Hungaricae **14**(3–4), 423–439 (1963). https://doi.org/10.1007/BF01895727
15. Delaunay, B.: Sur la sphère vide. A la mémoire de Georges Voronoï. Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles (6), 793–800 (1934)

16. Fowler, R.J., Paterson, M.S., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inf. Process. Lett. **12**(3), 133–137 (1981). https://doi.org/10.1016/0020-0190(81)90111-3
17. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001). https://doi.org/10.1016/S0377-2217(00)00100-4
18. Haxell, P.E.: Packing and covering triangles in graphs. Discret. Math. **195**, 251–254 (1999). https://doi.org/10.1016/S0012-365X(98)00183-6
19. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Tech. Rep. TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, October 2005. https://abc.erciyes.edu.tr/pub/tr06_2005.pdf
20. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J. Global Optim. **39**(3), 459–471 (2007). https://doi.org/10.1007/s10898-007-9149-x
21. Kirkpatrick, D.G., Hell, P.: On the completeness of a generalized matching problem. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, pp. 240–245 (1978). https://doi.org/10.1145/800133.804353
22. Leao, A.A.S., Toledo, F.M.B., Oliveira, J.F., Carravilla, M.A., Alvarez-Valdés, R.: Irregular packing problems: a review of mathematical models. Eur. J. Oper. Res. **282**(3), 803–822 (2020). https://doi.org/10.1016/j.ejor.2019.04.045
23. Lodi, A., Martello, S., Monaci, M.: Two-dimensional packing problems: a survey. Eur. J. Oper. Res. **141**(2), 241–252 (2002). https://doi.org/10.1016/S0377-2217(02)00123-6
24. 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 and Management Science, vol. 57, pp. 320–353. Springer, Boston, MA (2003). https://doi.org/10.1007/0-306-48056-5_11
25. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search: framework and applications, 2nd edn. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, pp. 363–397. Springer, New York (2010). https://doi.org/10.1007/978-1-4419-1665-5_12
26. Luo, C., Su, Z., Lü, Z.: A general heuristic approach for maximum polygon packing. In: 40th International Symposium on Computational Geometry, SoCG 2024. Leibniz International Proceedings in Informatics (LIPIcs), vol. 293, pp. 86:1–86:9 (2024). https://doi.org/10.4230/LIPIcs.SoCG.2024.86
27. Manić, G., Wakabayashi, Y.: Packing triangles in low degree graphs and indifference graphs. Discret. Math. **308**(8), 1455–1471 (2008). https://doi.org/10.1016/j.disc.2007.07.100
28. Mladenović, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**(11), 1097–1100 (1997). https://doi.org/10.1016/S0305-0548(97)00031-2
29. Zhang, K., Yu, W., Manhein, M., Waggenspack, W., Li, X.: 3D fragment reassembly using integrated template guidance and fracture-region matching. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 2138–2146 (2015). https://doi.org/10.1109/ICCV.2015.247