The 2D Shortest Superstring Problem

Dat Thanh Tran[a], Khai Quang Tran[a] and Van Khu Vu[a,*]

[a] VinUniversity, Hanoi, Vietnam

ARTICLE INFO

Keywords:
Shortest superstring problem
Two-dimensional strings
Genetic algorithm
Integer linear programming
Combinatorial optimization
Bounded-offset tree encoding

ABSTRACT

The Shortest Superstring Problem (SSP), finding the shortest string containing a given set of strings as substrings, is a classical NP-hard problem. We introduce its two-dimensional generalization (2D-SSP): given a collection of rectangular symbol arrays, arrange them on the integer lattice with symbol-consistent overlaps to minimize bounding-box area. While 1D-SSP is pure sequencing, 2D-SSP becomes a *simultaneous sequencing-and-packing* problem.

We prove NP-hardness for both the area and balanced-area objectives, and APX-hardness for the area objective via an L-reduction from 1D-SSP. We develop a *Bounded-Offset Tree Representation* that transforms this geometric problem into a finite combinatorial problem over spanning trees. A connectivity lemma guarantees that optimal solutions can be made 4-connected without increasing cost, ensuring the tree space covers all optima.

Exploiting this representation, we design a Tree-Based Genetic Algorithm (T-GA) with *locality-preserving crossover* that recombines subtrees rather than coordinates, preserving functional clusters across generations. Experiments verify near-optimality against exact ILP solvers (gap $\leq 2.6\%$) and show 6–12% improvement over greedy baselines on larger instances. The framework extends naturally to d dimensions.

1. Introduction

The Shortest Superstring Problem (SSP) is a classical NP-hard problem (Gallant, Maier and Storer, 1980): given a collection of strings, the goal is to construct the shortest string that contains each input string as a contiguous substring. SSP has been extensively studied, with known approximation algorithms (Blum, Jiang, Li, Tromp and Yannakakis, 1994; Mucha, 2013) and rich connections to combinatorial optimization.

This paper explores a largely uncharted territory at the intersection of two classical algorithmic domains: *stringology* (the study of 1D sequencing problems like SSP, where overlap is the key operation) and *bin packing* (the study of 2D geometric arrangement problems, where objects are placed without overlap). Standard packing algorithms fail when objects must overlap with symbol consistency; standard string algorithms fail when overlap can occur from four directions rather than two, and when a new object can fill an interior "hole" created by the arrangement of others. We introduce the *Two-Dimensional Shortest Superstring Problem* (2D-SSP) as a principled bridge between these domains, providing both the theoretical foundations and algorithmic machinery to handle the unique challenges that arise when sequencing meets geometry.

In 2D-SSP, the basic objects are rectangular 2D strings $T_1, \dots, T_n$ (finite 2D arrays over a finite alphabet), collected into a set $\mathcal{T}$. The goal is to place them in the plane with symbol-consistent overlaps so that all 2D strings are embedded while minimizing a bounding-box cost derived from the minimal axis-aligned bounding rectangle enclosing the occupied region. We consider two cost variants:

- the *area objective*, which minimizes the rectangle area $H \cdot W$, the natural 2D generalization of string length in the classical SSP;
- a *balanced-area objective*, which minimizes the side length $\max\{H, W\}$ of the smallest enclosing square, a secondary objective that penalizes extreme aspect ratios.

The primary motivation for 2D-SSP is *compression*: just as 1D-SSP asks how compactly a set of strings can be represented via a single superstring exploiting overlaps, 2D-SSP asks the same question for 2D patterns. This is the natural higher-dimensional generalization of a classical problem, and to our knowledge, the first systematic study

[*] Corresponding author

dat.tt3@vinuni.edu.vn (D.T. Tran); khai.tq@vinuni.edu.vn (K.Q. Tran); khu.vv@vinuni.edu.vn (V.K. Vu)
ORCID(s):

of SSP beyond one dimension. The problem exhibits a rich combinatorial structure—sitting at the intersection of stringology and geometric packing—that is absent from either domain alone and warrants study in its own right. Potential applications (DNA tile assembly, texture synthesis, 2D barcode design) may emerge where symbol-consistent overlap is physically meaningful, but our focus here is on establishing the theoretical and algorithmic foundations.

The transition from 1D to 2D introduces a fundamental *complexity leap*. In 1D-SSP, strings can only overlap from two directions (left/right), making it a pure sequencing problem. In 2D-SSP, strings can overlap from four directions, and, crucially, a new string can fill a "hole" created by the arrangement of other strings. This transforms the problem into a *simultaneous sequencing and packing* problem, where the optimal placement of one string depends not just on its neighbors but on the global geometric configuration. Our approach is the first to bridge these two domains by using a graph-based structure to handle the sequencing aspect and a grid-based canvas to handle the packing constraints.

Our central modelling innovation is a *relative-offset encoding* that shifts the search space from absolute Cartesian coordinates to placement trees. Rather than representing a solution as an explicit 2D array or a vector of (x, y) coordinates, we encode it as a *tree of relative offsets between 2D strings*. This representation achieves *symmetry-breaking* by collapsing the infinitely many translationally equivalent coordinate vectors into a single canonical form, and enforces structural connectivity by construction. Unlike coordinate-based representations, which suffer from high redundancy due to translational invariance, a solution at $(0, 0)$ is equivalent to one at $(1, 1)$, $(1, 2)$, etc., our tree-based encoding ensures that each individual represents a *unique relative arrangement*, significantly increasing search efficiency by eliminating redundant exploration of the coordinate space.

From a *theoretical* perspective, we develop a *Bounded-Offset Tree Representation* for 2D-SSP solutions. The key structural result (Theorem 4) shows that optimal placements can always be made 4-connected without increasing cost. This is intuitively similar to compaction arguments in VLSI floorplanning (Murata, Fujiyoshi, Nakatake and Kajitani, 1996a), but the proof is complicated by the *symbol-consistency* constraint: we must verify that sliding disconnected components together does not introduce symbol conflicts, a concern absent in traditional floorplanning where components are non-overlapping rectangles. Building on this, we prove an *Optimality-Preserving Equivalence* (Corollary 7): every connected placement can be represented by a spanning tree of a naturally defined placement graph (though the correspondence is many-to-one: multiple trees may encode the same placement, and not every tree yields a feasible placement when decoded). Crucially, at least one optimal solution always admits such a tree representation, so the tree search space is *complete* for optimization. Together, these results establish that *the search space can be reduced from an infinite Cartesian plane to a finite discrete combinatorial space*, the space of spanning trees with bounded edge labels. While this space remains exponential in the worst case, it transforms a geometric optimization problem into a structured combinatorial one amenable to both exact methods and structure-aware search exploiting this bounded-offset encoding. Our theoretical framework (the connectivity/compaction theorem and tree representation) relies only on lattice adjacency and translation, making it directly applicable to d-dimensional generalizations for any $d \geq 2$.

From an *algorithmic* perspective, the tree-based encoding enables a *locality-preserving crossover* operator that directly addresses the *building block hypothesis* (Goldberg, 1989): a subtree corresponds to a spatially coherent cluster, and crossover can transplant entire subtrees between parents, preserving beneficial local arrangements. Our Tree-Based GA (T-GA) consistently outperforms greedy baselines and matches ILP-optimal solutions on small instances while scaling to much larger problems.

2. Background and Related Work

2.1. Shortest Superstring Problem

In the classical SSP, the input is a set of strings

$$\mathcal{S} = \{s_1, \dots, s_n\}$$

over an alphabet Σ. A superstring is a string S in which each s_i appears as a substring. The objective is to minimize $|S|$. SSP is NP-hard, and there is a substantial literature on constant-factor approximations (e.g., greedy maximum-overlap merging, cycle-cover-based algorithms) and heuristic implementations used in practice.

2.2. 2D Covers and 2D Covering Sequences

Two-dimensional generalizations of string concepts appear in several areas:

- *2D covers and 2D strings.* Work on covers of 2D arrays considers how a small pattern can cover a larger 2D string with overlaps, generalizing the notion of a cover in 1D (Charalampopoulos, Pissis, Radoszewski, Waleń and Zuba, 2021).
- *Covering sequences and 2D covering sequences.* Recent research introduces covering sequences and covering 2D-sequences, where all $m \times n$ windows of a large 2D array form a covering code for patterns of that size up to a given radius. These provide natural sources of structured test instances (Yehezkeally and Schwartz, 2025).

These works focus on covering combinatorial spaces, whereas we focus on overlapping a *given finite set* of 2D strings with exact symbol consistency.

2.3. 2D Bin Packing and Cutting Stock

Classical two-dimensional bin packing problems ask how to place a collection of rectangles into one or more rectangular bins so as to minimize, for example, the number of bins used or the height of a single strip, under strict *non-overlap* constraints. The items are unlabeled shapes. A closely related problem is the *two-dimensional cutting stock problem* (Gilmore and Gomory, 1965; Lodi, Martello and Monaci, 2002), where the goal is to cut rectangular pieces from large stock sheets while minimizing waste. Both problems have been extensively studied using exact methods (branch-and-bound, column generation) and metaheuristics (genetic algorithms, simulated annealing) (Bennell and Oliveira, 2009).

Our setting is similar in that we also optimize a global bounding box for a family of rectangular pieces, but differs in two key ways. First, each string is a *discrete symbol array* rather than an unlabeled rectangle; second, *overlaps are allowed* as long as they are *symbol-consistent*. Thus a solution is not simply a packing of shapes, but a combinatorial “gluing” of patterns in which overlaps can reduce the effective occupied area, a phenomenon absent from standard 2D bin packing and cutting stock formulations. In the language of cutting stock, 2D-SSP allows “pieces” to share material when their patterns match, a constraint that transforms the problem from pure geometry to a hybrid of sequencing and packing.

2.4. Related Geometric and Assembly Problems

2D-SSP shares structural similarities with several problems in combinatorial optimization. In VLSI floorplanning, topological representations such as Sequence Pairs (Murata, Fujiyoshi, Nakatake and Kajitani, 1996b) and B*-trees (Chang, Chang, Wu and Wu, 2000) encode relative module positions; our placement trees adapt this approach to content-based adjacency. Patch-based texture synthesis (Efros and Freeman, 2001; Kwatra, Schödl, Essa, Turk and Bobick, 2003) places patches to minimize visual error in overlaps; 2D-SSP is the discrete, lossless limit requiring exact symbol consistency. The Tile Assembly Model (Winfree, Liu, Wenzler and Seeman, 1998) studies self-assembly of Wang tiles with edge-matching constraints; 2D-SSP asks the inverse question of finding the most compact configuration containing a given set of patterned tiles. Unlike jigsaw puzzle assembly (Sholomon, David and Netanyahu, 2013) or polyomino packing (Golomb, 1994), 2D-SSP permits symbol-consistent overlaps that enable compression beyond pure geometric packing.

3. Preliminaries

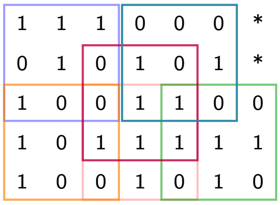
In this section, we formalize the Two-Dimensional Shortest Superstring Problem (2D-SSP), introduce the two objective variants, and develop a *Bounded-Offset Tree Representation* that enables structured algorithmic treatment. The central contribution of this section is showing that the search space can be reduced from an infinite coordinate space to a finite (though exponential) space of spanning trees with bounded edge labels.

Let Σ be a finite alphabet over which the 2D strings are defined. A *2D-string* over Σ is a finite $m \times n$ array $T \in \Sigma^{m\times n}$, for some $m, n \in \mathbb{N}$. For indices $1 \leq i \leq j \leq m$ and $1 \leq i' \leq j' \leq n$, we write $T[i..j,\ i'..j']$ for the corresponding subarray and call this a *2D-substring* of T.

We identify a 2D string T with a function on a finite index set $C_T \subset \mathbb{Z}^2$, its set of *local cell coordinates*. We write cells as pairs $(u, v) \in \mathbb{Z}^2$ and use the same coordinate system for both local and global positions: a translation by an offset $p(i) = (x_i, y_i)$ sends a local cell (u, v) of T_i to the global cell $p(i) + (u, v) = (x_i + u,\ y_i + v)$. The choice of which axis is drawn horizontally or vertically is irrelevant for our arguments; we only rely on coordinate-wise addition in $\mathbb{Z}^2$.

Let P be an $m' \times n'$ 2D-string. We denote the set of its occurrences in T by

$$\mathrm{Occ}(P, T) = \{(i, j) : T[i..i + m' - 1,\ j..j + n' - 1] = P\}.$$



1	1	1	0	0	0	*
0	1	0	1	0	1	*
1	0	0	1	1	0	0
1	0	1	1	1	1	1
1	0	0	1	0	1	0

Figure 1: A 2D-superstring S (5 rows × 7 columns) containing six 2D strings $T_1, \ldots, T_6$ as 2D-substrings (highlighted rectangles). The cells marked with "$*$" are *wildcard* (or *don't-care*) positions, cells within the bounding box that are not covered by any input string. The area is $|S|_{\text{area}} = 35$, and the balanced side length is $|S|_{\text{bal}} = 7$.

We will derive cost functions from the dimensions of the minimal axis-aligned bounding rectangle of a placement, and consider two variants: one that minimizes the area and one that minimizes the maximum side length (balanced-area objective).

Definition 1. Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a finite set of 2D strings over Σ, which we call *2D strings*. An $m \times n$ 2D-string S is a *2D-superstring* of $\mathcal{T}$ if each string T_i occurs as a 2D-substring of S, i.e.,

$$\mathrm{Occ}(T_i, S) \neq \emptyset \quad \text{for all } i \in \{1, \ldots, n\}.$$

We denote by

$$|S|_{\text{area}} := m \cdot n \quad \text{and} \quad |S|_{\text{bal}} := \max\{m, n\}$$

the *area* and the *balanced side length* of S, respectively. The area measure is the natural 2D analogue of string length in 1D-SSP, while the balanced measure constrains the aspect ratio. Both are derived from the minimal axis-aligned rectangle containing S and penalize any empty cells inside that rectangle.

Thus $|S|_{\text{area}}$ is the primary objective, the natural 2D analogue of superstring length (total cells in the bounding box), while $|S|_{\text{bal}}$ is a constrained variant that bounds the aspect ratio, ensuring neither dimension dominates.

Remark 1 (Wildcard characters and 2D holes)**.** The wildcard symbol "$*$" in Figure 1 represents a fundamental distinction between 1D-SSP and 2D-SSP. In 1D-SSP, every optimal superstring is *fully covered*: each position belongs to at least one input string. If any position were uncovered, we could delete that character to obtain a shorter superstring, contradicting optimality.

In 2D-SSP, this property fails. The two-dimensional geometry permits *holes*: cells within the bounding box that no input string covers. These holes cannot simply be "deleted" as in 1D, because removing a row or column would disrupt the geometric arrangement of strings in other parts of the superstring. The wildcard cells contribute to the bounding-box cost but carry no information, they can be filled with any symbol (or left undefined) without affecting which strings are embedded.

Formally, given a placement p, let $R(p) = \bigcup_{i=1}^{n} \mathrm{footprint}(T_i, p)$ denote the set of occupied cells, and let $B(p)$ denote the bounding box. The *hole set* is $B(p) \setminus R(p)$, cells inside the bounding box but not covered by any string. In Figure 1, this set contains two cells (the "$*$" positions). These holes represent "wasted" area that inflates the objective value, and minimizing them is part of the optimization challenge unique to 2D-SSP.

Definition 2. Given a finite set $\mathcal{T}$ of 2D strings over Σ, we define two variants of the Two-Dimensional Shortest Superstring Problem:

- *Area-based 2D-SSP* (2D-SSP$_{\text{area}}$): find a 2D-superstring S of $\mathcal{T}$ minimizing $|S|_{\text{area}}$. This is the primary variant, directly generalizing the 1D-SSP objective.
- *Balanced-area 2D-SSP* (2D-SSP$_{\text{bal}}$): find a 2D-superstring S of $\mathcal{T}$ minimizing $|S|_{\text{bal}}$. This variant constrains the aspect ratio and is useful in applications requiring near-square layouts.

Both objectives depend only on the minimal axis-aligned bounding rectangle of S and penalize all empty cells inside it.

3.1. Computational complexity

We establish that 2D-SSP is NP-hard. Both proofs are straightforward reductions; we state them for completeness.

Theorem 1 (NP-hardness of 2D-SSP$_{\text{area}}$)**.** *2D-SSP*$_{\text{area}}$ *is NP-hard, even for binary alphabets.*

Proof. Reduction from 1D-SSP (NP-hard for $|\Sigma| \geq 2$ (Gallant et al., 1980)). Given 1D strings $\mathcal{S} = \{s_1, \dots, s_n\}$, create height-1 2D strings $T_i = 1 \times |s_i|$. Any 2D placement with k rows and maximum row width W has area $\geq k \cdot W$. Concatenating rows yields a 1D superstring of length $\leq k \cdot W$. Conversely, the optimal 1D superstring gives a 1-row 2D placement. Thus $\text{OPT}_{\text{2D}} = \text{OPT}_{\text{1D}}$. □

Theorem 2 (NP-hardness of 2D-SSP$_{\text{bal}}$)**.** *2D-SSP*$_{\text{bal}}$ *is NP-hard.*

Proof. Reduction from minimum enclosing square packing (NP-hard (Leung, Tam, Wong, Young and Chin, 1990)). Given rectangles $R_1, \dots, R_n$, assign each a unique symbol σ_i from alphabet Σ with $|\Sigma| = n$. The resulting 2D strings cannot overlap (symbol conflict), so the problem reduces to non-overlapping rectangle packing. □

Remark 2 (Complexity summary and open problems)**.** *Area objective:* NP-hard and APX-hard for any $|\Sigma| \geq 2$ (inherited from 1D-SSP via L-reduction with $\alpha = \beta = 1$; see Appendix A.2).

Balanced objective: NP-hard, but Theorem 2 requires a large alphabet ($|\Sigma| = n$). This places the proof in the *packing regime* where no overlaps occur.

Open problem: *The complexity of 2D-SSP*$_{\text{bal}}$ *over a fixed constant-size alphabet (e.g., binary) remains open.* In the sequencing regime where overlaps are common, neither the 1D-SSP reduction nor the packing reduction applies directly.

3.1.1. Experimental scope

Our experiments focus on the *sequencing regime*: binary alphabets ($|\Sigma| = 2$) where overlap opportunities are common enough to matter but rare enough to be computationally challenging. This is the regime where 2D-SSP is most distinct from pure packing and where our tree-based methods provide the most value. See Appendix A.3 for extended discussion of the sequencing–packing spectrum and entropy effects.

Whenever the distinction between the two variants is not important, we simply refer to either as *2D-SSP*. Hereafter, we use *string* to mean *2D string* unless otherwise specified.

We assume throughout that strings are axis-aligned rectangles and cannot be rotated or reflected. Following standard 1D-SSP convention, we assume the input is *substring-free*: no string T_i is a 2D-substring of another string T_j. Redundant strings can be removed in polynomial-time preprocessing without affecting the optimal solution, since any 2D-superstring containing T_j automatically contains T_i.

Rather than working directly with the superstring S, we use placements on the integer grid.

Definition 3. A *placement* of $\mathcal{T}$ is a function

$$p : \{1, \dots, n\} \to \mathbb{Z}^2, \qquad p(i) = (x_i, y_i),$$

assigning an integer offset to each string T_i. A cell of T_i with local coordinates (u, v) (row and column indices) is mapped to global coordinates $(x_i + u,\ y_i + v) \in \mathbb{Z}^2$.

A placement p is symbol-consistent if for every global coordinate $(x, y) \in \mathbb{Z}^2$, all strings covering (x, y) under p write the same symbol. We denote by

$$R(p) := \{(x, y) \in \mathbb{Z}^2 : (x, y) \text{ is covered by some } T_i \\ \text{under } p\}$$

the union of occupied global cells, and let $B(p)$ be the minimal axis-aligned rectangle containing $R(p)$. Let $W(p)$ and $H(p)$ be the width and height of $B(p)$, and define

$$\text{cost}_{\text{area}}(p) := W(p) \cdot H(p), \qquad \text{cost}_{\text{bal}}(p) := \max\{W(p), H(p)\}.$$

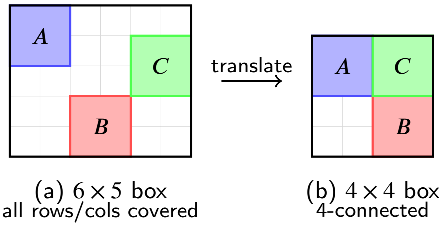


Figure 2: Row/column coverage is necessary but not sufficient for 2D optimality. (a) Both strings cover all rows and columns of the 6×5 bounding box, yet they are disconnected. (b) Translating B diagonally yields a 4-connected placement with a smaller 4 × 4 bounding box.

Restricting the symbol map to $B(p)$ yields an $m \times n$ array T_p; by construction T_p is a 2D-superstring of $\mathcal{T}$, and its area and balanced side satisfy

$$|T_p|_{\text{area}} = \text{cost}_{\text{area}}(p), \qquad |T_p|_{\text{bal}} = \text{cost}_{\text{bal}}(p).$$

In particular, empty cells of $B(p)$ that are not covered by any strings still contribute to both objectives. Thus every symbol-consistent placement defines a feasible solution to both 2D-SSP$_{\text{area}}$ and 2D-SSP$_{\text{bal}}$, with cost equal to the chosen bounding-box functional.

Remark 3. We minimize bounding box area $W \cdot H$ rather than union area $|R(p)|$ for two reasons: (i) it is the natural 2D generalization of 1D superstring length, and (ii) minimizing $|R(p)|$ would permit fragmented layouts with “holes,” whereas bounding-box cost penalizes such fragmentation and encourages compact, connected arrangements.

Remark 4. In 1D-SSP, every position in an optimal superstring must be covered by some input string. In 2D, the analogous condition, every row and column intersects some string, is necessary but not sufficient (Figure 2). The connectivity/compaction theorem (Theorem 4) provides the appropriate 2D analogue by ensuring optimal placements can be made 4-connected.

Conversely, let S be a 2D-superstring of $\mathcal{T}$ and fix an arbitrary occurrence $(i, j) \in \text{Occ}(T_k, S)$ for each string T_k. Placing T_k with offset (i, j) then yields a symbol-consistent placement whose induced array is S up to a global translation. Hence optimizing over 2D-superstrings is equivalent to optimizing over symbol-consistent placements, modulo a global shift of all coordinates. We therefore work with placements from now on.

A subset $R \subseteq \mathbb{Z}^2$ is *4-connected* if its adjacency graph under the 4-neighbourhood ($\|x - y\|_1 = 1$) is connected. Two cells $x, y \in \mathbb{Z}^2$ overlap if $x = y$.

Definition 4. Given the string set $\mathcal{T}$, the *placement graph* G^{pl} has vertex set $\{1, \dots, n\}$. Its edges are triples

$$e = (i, j, \delta) \quad \text{with} \quad i \neq j, \ \delta \in \mathbb{Z}^2,$$

where δ is a relative offset such that placing T_j at position $p(j) = p(i) + \delta$ yields symbol-consistent contact between T_i and T_j, meaning they either overlap with matching symbols, or are 4-adjacent (share a boundary edge). Multiple edges may exist for the same unordered pair $\{i, j\}$, corresponding to different valid offsets.

Intuitively, each edge (i, j, δ) in G^{pl} specifies a way of gluing T_j next to T_i. The placement graph is determined entirely by $\mathcal{T}$ and can be precomputed before searching for solutions.

Lemma 3. *Let T_i and T_j have bounding boxes of dimensions $w_i \times h_i$ and $w_j \times h_j$. If (i, j, δ) is an edge in G^{pl}, then $\delta = (\Delta x, \Delta y)$ satisfies:*

$$|\Delta x| \leq w_i + w_j - 1, \qquad |\Delta y| \leq h_i + h_j - 1.$$

Proof. For T_i and T_j to be in contact (overlap or 4-adjacent), their bounding boxes must intersect or share an edge. The x-projections $[0, w_i - 1]$ and $[\Delta x, \Delta x + w_j - 1]$ intersect or are adjacent if $|\Delta x| \leq w_i + w_j - 1$. The bound for Δy follows symmetrically. □

Remark 5 (Graph size vs. tree space)**.** Lemma 3 ensures that the *placement graph* G^{pl} has polynomial size: for each pair (T_i, T_j), there are $O((w_i + w_j)(h_i + h_j))$ candidate offsets to check. For uniform $w \times h$ strings, the placement graph has $O(n^2 wh)$ edges.

However, the *search space of spanning trees* remains exponential. By Cayley's formula, a complete graph on n vertices has n^{n-2} spanning trees; while our placement graph is typically sparser, the number of feasible placement trees can still grow exponentially with n. The significance of our reduction is not that it yields a polynomial search space, but that it transforms an *infinite continuous space* (absolute coordinates on $\mathbb{Z}^2$) into a *finite discrete combinatorial space* (spanning trees with bounded edge labels). This discretization enables exact enumeration for small instances and structured search exploiting the graph-based encoding for larger ones.

Remark 6 (Edge density)**.** The bound $O(n^2 wh)$ on $|E(G^{\mathrm{pl}})|$ is a worst-case geometric bound. The actual edge count depends on string entropy: high-entropy strings yield sparse graphs (dominated by adjacency edges), while low-entropy strings yield dense graphs (many symbol-consistent overlaps). See Appendix A.4 for detailed analysis.

Remark 7 (Optimized offset enumeration)**.** In the sequencing regime, practical performance improves by enumerating offsets in order of bounding-box increase, stopping at the first level with a symbol-consistent offset. See Appendix A.5 for details.

Definition 5. Let p be a symbol-consistent placement of $\mathcal{T}$. The *contact graph* $G^{\mathrm{ct}}(p)$ is the subgraph of G^{pl} induced by p: it has vertex set $\{1, \ldots, n\}$, and edge (i, j, δ) is present if $\delta = p(j) - p(i)$ and this edge exists in G^{pl}.

Equivalently, strings i and j are adjacent in $G^{\mathrm{ct}}(p)$ if they are in contact under p (overlapping or 4-adjacent).

The contact graph $G^{\mathrm{ct}}(p)$ records which edges of the placement graph are "realized" by a given placement. Different placements of the same instance may realize different subsets of edges.

Remark 8. Our cost functionals depend only on the bounding rectangle $B(p)$, not directly on the cardinality of $R(p)$. In particular, two placements with the same bounding box have the same cost for both 2D-SSP$_{\mathrm{area}}$ and 2D-SSP$_{\mathrm{bal}}$. The area-based objective directly generalizes the 1D shortest superstring objective (string length), while the balanced-area variant additionally constrains the aspect ratio. Both differ from geometric covering formulations that minimize the area of the union $R(p)$.

Assumption 1. In the structural discussion below we work with placements whose occupied region $R(p)$ is 4-connected. The following theorem shows this is without loss of optimality.

The following theorem shows that optimal solutions can be restricted to connected placements, a result analogous to compaction in VLSI floorplanning (Murata et al., 1996a).

Theorem 4 (Connectivity/compaction)**.** *Let p be an optimal symbol-consistent placement for 2D-SSP (under either* $\mathrm{cost}_{\mathrm{area}}$ *or* $\mathrm{cost}_{\mathrm{bal}}$*). If $R(p)$ is not 4-connected, then there exists an optimal symbol-consistent placement p' such that $R(p')$ is 4-connected and* $\mathrm{cost}(p') \leq \mathrm{cost}(p)$.

Proof. We prove existence of an optimal 4-connected placement by an extremal argument.

Canonical normalization and tie-breaking. Among all optimal symbol-consistent placements (for the chosen objective), pick one q as follows. First translate q so that the lower-left corner of its bounding box is at the origin, i.e., $B(q) = [0, W-1] \times [0, H-1]$ for some W, H. Among all such translated optimal placements, choose q that minimizes the pair (W, H) lexicographically, and subject to that minimizes

$$\Phi(q) := \sum_{(x,y)\in R(q)} x + \sum_{(x,y)\in R(q)} y.$$

This tie-breaking is well-defined because, after anchoring $B(q)$ at the origin, the set $R(q) \subseteq [0, W-1] \times [0, H-1]$ is finite.

Claim: $R(q)$ is 4-connected. Assume for contradiction that $R(q)$ has at least two maximal 4-connected components. Let these components be $C_1, \ldots, C_k$ with $k \geq 2$.

(1) Components are separated by distance at least 2. For distinct components $C_i \neq C_j$, we have $\mathrm{dist}(C_i, C_j) \geq 2$: if $\mathrm{dist}(C_i, C_j) = 1$ then there exist 4-adjacent cells across them, contradicting maximality of 4-connected components.

(2) Any unit shift of a component cannot create overlap. Fix a component C and a unit vector $e \in \{(\pm 1, 0), (0, \pm 1)\}$. If $(C + e) \cap (R(q) \setminus C) \neq \emptyset$, then there exist $c \in C$ and $d \in R(q) \setminus C$ with $c + e = d$, hence $\|c - d\|_1 = 1$ and thus $\mathrm{dist}(C, R(q) \setminus C) = 1$, contradicting (1). Therefore shifting any component by a unit step preserves non-overlap with all other components. Since the shift is rigid, symbol-consistency is preserved as well.

(3) Each component touches every side of the bounding box. Suppose some component C does not touch the left side of $B(q)$, i.e., every cell $(x, y) \in C$ satisfies $x \geq 1$. Then $C + (-1, 0) \subseteq B(q)$. By (2), translating C by $(-1, 0)$ yields a symbol-consistent placement q' whose occupied set stays inside the same box $B(q)$, so (W, H) does not increase. However, this translation strictly decreases Φ (every cell of C decreases its x-coordinate by 1), contradicting the choice of q. Thus every component touches the left side. The same argument applied to $(1, 0)$, $(0, -1)$, and $(0, 1)$ shows that every component touches the right, bottom, and top sides of $B(q)$.

(4) Two disjoint components cannot both touch all four sides. Fix two distinct components A and B. Since A is 4-connected and touches the left and right sides, it contains a 4-neighbour path in the grid from the left side of $B(q)$ to the right side. Similarly, since B is 4-connected and touches the top and bottom sides, it contains a 4-neighbour path from the top side to the bottom side. By the standard planar separation (crossing) property of the rectangular grid (see Lemma 8 in Appendix A.1), these two paths must share a grid cell. This contradicts that A and B are disjoint.

Therefore $k = 1$ and $R(q)$ is 4-connected.

Conclusion. Let $p' := q$. Then p' is optimal for the chosen objective and $R(p')$ is 4-connected. In particular, $\mathrm{cost}(p') = \mathrm{cost}(q) \leq \mathrm{cost}(p)$. □

Corollary 5. *For any instance of 2D-SSP, there exists an optimal placement whose occupied region is 4-connected. Consequently, Assumption 1 is justified for all instances.*

This connectivity/compaction theorem establishes that restricting attention to connected placements loses no optimal solutions. The assumption matches our experimental focus and enables the tree-based structural perspective developed in this section. It is not required for the correctness of the algorithms in Section 4, which operate on arbitrary symbol-consistent placements.

Corollary 6 (Connected occupied region implies connected contact graph)**.** *Let p be a symbol-consistent placement such that $R(p)$ is 4-connected. Then the contact graph $G^{\mathrm{ct}}(p)$ is connected.*

Proof. If $G^{\mathrm{ct}}(p)$ were disconnected, we could partition the strings into two nonempty sets A and B with no edge between them. For each string T_i, let R_i be the set of global cells covered by T_i under p. Define

$$R_A = \bigcup_{i \in A} R_i, \qquad R_B = \bigcup_{j \in B} R_j,$$

so that $R(p) = R_A \cup R_B$ and $R_A \cap R_B = \emptyset$. By construction there is no pair of 4-adjacent cells across R_A and R_B, which contradicts 4-connectivity of $R(p)$. □

By Theorem 4 and Corollary 6, every optimal connected placement has a connected contact graph that admits a spanning tree. We now formalize this tree representation.

3.2. Placement trees

A placement tree encodes a solution as a spanning tree with labeled edges specifying relative offsets between strings. The key insight is that edge labels are drawn from a *bounded* set determined by string dimensions, which, combined with the finite number of spanning tree topologies, yields a finite search space.

Definition 6. A *placement tree* for $\mathcal{T}$ is a rooted tree $F = (V, E)$ with vertex set $V = \{1, \dots, n\}$ together with, for each edge $\{i, j\} \in E$, a label $\delta_{ij} \in \mathbb{Z}^2$ interpreted as the relative offset from i to j. For each oriented edge (i, j) we store δ_{ij} and require $\delta_{ji} = -\delta_{ij}$.

By Lemma 3, valid edge labels are bounded: if T_i and T_j have dimensions $w_i \times h_i$ and $w_j \times h_j$, then any symbol-consistent contact requires $|\Delta x| \leq w_i + w_j - 1$ and $|\Delta y| \leq h_i + h_j - 1$. This bounds the number of candidate labels per edge to $O((w_i + w_j)(h_i + h_j))$—for uniform $w \times h$ strings, $O(wh)$ per edge—ensuring the search space of placement trees is finite.

The *realization* of F with root r and root position $p(r) \in \mathbb{Z}^2$ is the placement p_F defined by

$$p_F(i) \; = \; p(r) + \sum_{t=0}^{\ell-1} \delta_{v_t v_{t+1}},$$

where $r = v_0, v_1, \dots, v_\ell = i$ is the unique simple path from r to i in F.

We call F *feasible* if the realization p_F is symbol-consistent for some (equivalently, every) choice of root position.

Remark 9 (Placements vs. Trees)**.** Not every symbol-consistent placement corresponds to a connected tree. If the occupied region $R(p)$ is disconnected, the contact graph $G^{\mathrm{ct}}(p)$ is a forest of multiple components, not a single spanning tree. However, by Theorem 4, *at least one optimal placement* has a connected occupied region, and thus corresponds to a spanning tree. This is the crux of the search space reduction: we need not enumerate all placements (including disconnected ones), but only spanning trees of the placement graph.

Remark 10. Feasibility is a global property: even if all adjacent pairs (i, j) are locally consistent, collisions may occur between distant parts of the tree when their footprints overlap after summing the offsets. In particular, closed walks in the underlying contact graph induce non-trivial "loop-closure" constraints on the offsets. We treat feasibility algorithmically: given a placement tree F, we realize it via p_F and explicitly check symbol-consistency.

The following theorem establishes that optimal solutions can always be represented as placement trees, a crucial fact that justifies restricting our search to tree-based encodings.

Corollary 7 (Tree representation of optimal placements)**.** *The following statements hold:*

(i) ***(Existence of tree-optimal solutions.)*** *For any instance of 2D-SSP, there exists an optimal placement p^* that corresponds to a feasible placement tree. Specifically, by Theorem 4 we may assume $R(p^*)$ is 4-connected; any spanning tree F of $G^{\mathrm{ct}}(p^*)$, equipped with edge labels $\delta_{ij} := p^*(j) - p^*(i)$, is then a feasible placement tree whose realization coincides with p^* up to global translation.*

(ii) ***(Completeness.)*** *Conversely, any feasible placement tree F induces a symbol-consistent placement p_F that is a valid solution to 2D-SSP.*

Proof. (i) Optimal Placement → Tree. Let p^* be any optimal placement. By Theorem 4, there exists an optimal placement p with the same cost such that $R(p)$ is 4-connected. By Corollary 6, $G^{\mathrm{ct}}(p)$ is connected, so it admits a spanning tree F. For any edge (i, j) of F, we have $p(j) = p(i) + \delta_{ij}$ by definition of the labels. For any vertex i with path $r = v_0, \dots, v_\ell = i$ from root r:

$$p(i) = p(r) + \sum_{t=0}^{\ell-1} \delta_{v_t v_{t+1}} = p_F(i).$$

Thus p equals the realization p_F (for root position $p(r)$), which is symbol-consistent, so F is feasible and optimal.

(ii) Tree → Placement. By feasibility, p_F is symbol-consistent. Every string T_i is placed exactly once, so T_i occurs as a 2D-substring of the induced array T_{p_F}, making it a 2D-superstring of $\mathcal{T}$. □

The significance of this corollary is that *no optimal solution is lost* by restricting to the Bounded-Offset Tree Representation. While not every placement corresponds to a tree, disconnected placements correspond to forests of multiple trees, Theorem 4 ensures that at least one optimal solution has a connected contact graph, and hence corresponds to a single spanning tree. Combined with the bounded edge labels (Lemma 3), this establishes that the search space is both *finite* (bounded labels, finite tree topologies) and *complete* (contains at least one optimum). This justifies designing algorithms that search exclusively over placement trees.

Remark 11. All results in this section concern only symbol-consistency, connectivity, and combinatorial structure. They apply verbatim to both 2D-SSP$_{\mathrm{area}}$ and 2D-SSP$_{\mathrm{bal}}$.

Remark 12 (Extension to higher dimensions)**.** The theoretical framework (connectivity lemma, bounded offsets, tree representation) extends directly to d-dimensional SSP for any $d \geq 1$, with $2d$-adjacency replacing 4-adjacency. See Appendix A.6 for details.

Table 1
Summary of main notation.

Symbol	Meaning
Σ	Alphabet
$\mathcal{T} = \{T_1, \dots, T_n\}$	Input set of strings
$C_T \subset \mathbb{Z}^2$	Local cell coordinates of string T
P	Pattern 2D-string (for occurrences)
$\mathrm{Occ}(P, T)$	Set of occurrences of P in T
S	2D-superstring of $\mathcal{T}$
$\lvert S\rvert_{\mathrm{area}}$	Area of minimal bounding box of S
$\lvert S\rvert_{\mathrm{bal}}$	Balanced side length $\max\{m, n\}$ of S
$p(i) = (x_i, y_i)$	Placement (offset) of string T_i
$R(p) \subset \mathbb{Z}^2$	Union of occupied cells under placement p
$B(p)$	Minimal axis-aligned bounding box of $R(p)$
$W(p), H(p)$	Width and height of $B(p)$
$\mathrm{cost}_{\mathrm{area}}(p)$	Area-based cost $W(p)H(p)$
$\mathrm{cost}_{\mathrm{bal}}(p)$	Balanced-area cost $\max\{W(p), H(p)\}$
$G^{\mathrm{ct}}(p)$	Contact graph induced by placement p
G^{pl}	Placement graph of symbol-consistent offsets
$F = (V, E)$	Placement tree on $\{1, \dots, n\}$
$\delta_{ij} \in \mathbb{Z}^2$	Relative offset from i to j in F
p_F	Realization (tree-induced placement) of F

3.3. Algorithmic implications

Corollary 7 has two key algorithmic consequences. First, it guarantees that *optimal solutions are always reachable* via tree-based search: since at least one optimal placement corresponds to a feasible placement tree, any algorithm that exhaustively searches over placement trees is guaranteed to find an optimum. Second, combined with the bounded offset property (Lemma 3), it yields a *finite discrete search space*: while the number of spanning trees can be exponential in n (see Remark 5), this is a dramatic improvement over the infinite space of absolute coordinate vectors. For small instances, this enables exact ILP enumeration; for larger instances, it provides a well-structured combinatorial space amenable to structure-aware search.

Remark 13 (Existence vs. search reachability)**.** Corollary 7 establishes *existence* of an optimal placement tree, but our GA searches only "greedily-completable" trees. Empirically, the GA matches or closely approaches ILP solutions on small instances, suggesting that this subspace contains near-optimal solutions. The theoretical gap remains open; see Appendix A.7 for detailed analysis.

The tree structure has three properties making it suitable for structure-aware search: (1) *locality preservation*—subtrees correspond to spatially coherent clusters; (2) *incremental realizability*—trees can be grown one string at a time; and (3) *symmetry breaking*—relative offsets factor out global translation, yielding a non-redundant search space.

These properties motivate our choice to design a genetic algorithm whose individuals are placement trees and whose crossover operates on subtrees, rather than using a more conventional coordinate-based representation.

4. Algorithms

Having established the structural foundation linking placements and trees, we present three algorithmic approaches for solving 2D-SSP. These methods span the spectrum from exact to structure-aware, offering different trade-offs between solution quality and computational cost:

- An *exact ILP formulation* (Section 4.1) that enumerates all candidate placements on a discrete grid and optimizes over them using mixed-integer linear programming. This approach guarantees optimal solutions but is limited to small instances.
- A *merge-based greedy heuristic* (Section 4.2) adapted from the classical 1D Shortest Superstring algorithm. This baseline repeatedly merges pairs of partial superstrings with maximum overlap.

- A *tree-growing greedy heuristic* (Section 4.3) that builds a placement tree incrementally, motivated by Corollary 7. At each step it attaches a new string to minimize the bounding-box cost.
- A *tree-based genetic algorithm* (Section 4.4) that represents individuals as placement trees and uses crossover operators that recombine subtrees. By exploiting the relative-offset encoding, this approach aims to combine the quality of exact methods with the scalability of structure-aware search.

All three methods can be instantiated with either of the two bounding-box cost variants, 2D-SSP$_{\text{area}}$ or 2D-SSP$_{\text{bal}}$.

The genetic algorithm uses placement trees rather than the more obvious coordinate-based representation (where each individual is a vector of (x, y) coordinates for each string). This choice is motivated by several considerations. First, in the coordinate representation, an offspring produced by crossover rarely inherits good local structure from its parents: if parent 1 places strings A and B in a well-overlapping configuration, and parent 2 places strings B and C similarly, a crossover that takes A from parent 1 and C from parent 2 will likely place them far apart, destroying both favorable overlaps. In contrast, our tree-based crossover transplants entire subtrees, preserving the relative offsets among all strings in the subtree. Second, the coordinate representation is highly redundant: any global translation of a placement yields the same objective value, so the search space contains infinitely many equivalent solutions. The tree representation eliminates this redundancy by encoding only the pairwise offsets that matter. Third, the tree structure aligns naturally with the connectivity requirement: a spanning tree automatically ensures that all strings are geometrically connected, whereas the coordinate representation requires additional constraints or repair operators to enforce connectivity.

4.1. Exact Verification via ILP

To validate the solution quality of our heuristic approaches, we formulate a direct grid-based mixed-integer linear program (Wolsey, 1998). We emphasize that this formulation is *not* intended as a scalable solver for general instances, but strictly as a *ground-truth oracle* for small-scale verification ($N \leq 10$). This allows us to measure exactly how close our genetic algorithm comes to the global optimum.

The model works with discrete candidate placements of each 2D string on a finite grid: it enumerates a finite set of allowed origins $\mathcal{O}_i$ for each 2D string T_i, uses binary decision variables $b_{io} \in \{0, 1\}$ to choose exactly one origin per string, precomputes pairwise conflict indicators $\kappa_{ijoo'}$ to forbid symbol-inconsistent placements, and minimizes the bounding-box cost via big-M constraints that track the enclosing rectangle. For the area objective, we linearize the bilinear term $W \cdot H$ using the McCormick envelope (McCormick, 1976); for the balanced objective, we simply minimize $L = \max\{W, H\}$ via linear constraints.

The formulation incorporates several refinements to improve tractability: (i) greedy-based grid bounds that accommodate all optimal aspect ratios while remaining much smaller than naïve worst-case bounds; (ii) symmetry breaking by fixing the first string at the origin; and (iii) instance-adaptive big-M constants that tighten the LP relaxation. The complete mathematical formulation, including all decision variables, constraints, and objective functions for both the balanced and area variants, is provided in Appendix B.

Despite these optimizations, the number of variables and conflict constraints grows quickly with the number of strings. Consequently, this ILP is practically limited to small instances ($N \leq 10$) and serves exclusively as a verification tool to certify the optimality gap of our heuristic methods.

4.2. Merge-Based Greedy Heuristic

Before presenting our tree-growing heuristic, we describe a natural baseline adapted from the classical 1D Shortest Superstring Problem: the *merge-based greedy* algorithm.

In the 1D setting, the Greedy Superstring Algorithm (Blum et al., 1994) is remarkably effective: it repeatedly merges the pair of strings with maximum overlap until a single superstring remains. This simple strategy achieves a 4-approximation for 1D-SSP (Blum et al., 1994), later improved to 2.5 (Kaplan and Shafrir, 2005). Empirically, the algorithm performs far better than these worst-case bounds suggest: extensive experiments on both random and biological sequences show approximation ratios consistently below 1.05 (Cazaux and Rivals, 2018), making it a strong practical baseline.

Role in our evaluation. We include merge-greedy not as a strawman but for two principled reasons. First, it provides *scalability comparison*: merge-greedy runs in polynomial time and scales to arbitrarily large instances, whereas our exact ILP baseline is limited to $n \leq 10$ strings. Second, its near-optimal performance on 1D-SSP ($< 5\%$ gap) makes it a

Algorithm 1 Merge-Based Greedy for 2D-SSP

```
Require: Set of 2D strings $\mathcal{T} = \{T_1, \dots, T_n\}$
Ensure: 2D superstring $S$
$\mathcal{S} \leftarrow \{T_1, \dots, T_n\}$    ▷ Set of partial superstrings
while $|\mathcal{S}| > 1$ do
    $(S_i^*, S_j^*, \delta^*) \leftarrow \arg\max_{S_i, S_j \in \mathcal{S}, \delta} \mathrm{overlap}(S_i, S_j, \delta)$    ▷ Find best symbol-consistent overlap
    $S_{\text{merged}} \leftarrow \textsc{Merge}(S_i^*, S_j^*, \delta^*)$    ▷ Create merged superstring
    $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{S_i^*, S_j^*\}) \cup \{S_{\text{merged}}\}$
end while
return the single element of $\mathcal{S}$
```

calibration baseline: if our 2D algorithms cannot substantially outperform a method that is nearly optimal in 1D, then the 2D structure is not being exploited. Our primary validation of solution quality, however, comes from comparison against exact ILP solutions on tractable instances (Table 4), where convergence to the global optimum is verifiable.

We adapt this approach to the 2D setting as follows. Given a set $\mathcal{T}$ of 2D strings, we maintain a set of *partial superstrings* (initially, each string is its own partial superstring). At each step, we identify the pair (S_i, S_j) of partial superstrings that can be merged with maximum symbol-consistent overlap, merge them into a single partial superstring, and repeat until only one remains.

Crucially, the overlap function measures *geometrical overlap*, the area of the overlapping region, rather than the number of matching non-wildcard characters:

$$\mathrm{overlap}(S_i, S_j) = \max_{\delta \in \mathcal{D}_{ij}} |R_i \cap (R_j + \delta)|,$$

where R_i, R_j are the cell sets of S_i, S_j, the offset δ ranges over all translations that yield symbol-consistent overlap, and $|\cdot|$ denotes the cardinality of the intersection (i.e., the number of overlapping cells, regardless of whether those cells contain alphabet symbols or wildcards).

Geometrical vs. character overlap. In the 1D setting, maximizing geometrical overlap and maximizing character overlap are equivalent: every overlapping position contains exactly one character. In the 2D setting, however, these objectives diverge. Consider two 3×3 strings that can overlap in two ways: (a) a 2×2 region with 4 matching characters, or (b) a 1×3 strip with 3 matching characters. A character-based criterion would prefer (a), but this choice might force the merged component into an unfavorable shape that increases the final bounding box.

We use geometrical overlap because it directly relates to the bounding-box objective: maximizing the area of overlap is equivalent to minimizing the area increase when merging. Formally, if S_i has area A_i and S_j has area A_j, then the merged component has area

$$A_{\text{merged}} = A_i + A_j - |R_i \cap (R_j + \delta)|,$$

so maximizing geometrical overlap minimizes A_{merged}. This observation motivates our choice of overlap function. The merge operation places S_j at the offset δ^* achieving this maximum, creating a new partial superstring whose bounding box contains both.

Hypothesis: 1D vs. 2D performance. We hypothesize that the merge-based greedy performs well when the input strings are *nearly one-dimensional* (i.e., have aspect ratios close to $1 \times m$ or $m \times 1$), since this regime closely resembles the classical 1D setting where the algorithm has provable guarantees. However, as strings become *genuinely two-dimensional* (aspect ratios closer to 1), the merge-based approach may struggle: the "best overlap" criterion optimizes locally for overlap size but ignores how the merge affects the global bounding-box shape. In contrast, our tree-growing heuristic (Section 4.3) explicitly optimizes the bounding-box cost at each step, which we expect to yield better solutions for 2D instances.

The procedure is summarized in Algorithm 1.

We test this hypothesis experimentally in Section 5: Table **??** compares merge-based greedy against our methods on instances with 1×8 strings (nearly 1D), while Tables 2 and **??** evaluate performance on 3×3 strings (genuinely 2D).

4.3. Tree-Growing Greedy Heuristic

Motivated by Corollary 7, which establishes that every connected placement corresponds to a spanning tree of relative offsets, we design a greedy heuristic that directly constructs a *placement tree*. Rather than merging pairs of partial superstrings as in the merge-based approach, we grow a single tree by iteratively attaching strings to the current structure.

The algorithm maintains:

- A *placement tree* $F = (V_F, E_F)$ with $V_F \subseteq \{1, \dots, n\}$ representing the strings placed so far;
- A *canvas* of occupied global cells with their symbols;
- The bounding-box coordinates $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$ of the current placement.

At each iteration, we select an unplaced string $T_j \notin V_F$ and attach it to some string $T_i \in V_F$ via an edge (i, j, δ) from the placement graph G^{pl}, choosing the pair (i, j, δ) that minimizes the resulting bounding-box cost while maintaining symbol-consistency. This directly mirrors the structural insight of Corollary 7: we are constructing an optimal placement by "growing" a spanning tree one edge at a time.

Crucially, by Lemma 3, we need only consider offsets δ within the bounded window $|\Delta x| \leq w_i + w_j - 1$ and $|\Delta y| \leq h_i + h_j - 1$. This transforms what might seem like an unbounded search into a tractable enumeration: for each candidate parent T_i in the current tree and each unplaced string T_j, we examine $O((w_i + w_j)(h_i + h_j))$ candidate attachment positions.

The current width and height are

$$w(C) = x_{\max} - x_{\min} + 1, \qquad h(C) = y_{\max} - y_{\min} + 1,$$

and we define either

$$\mathrm{size}_{\mathrm{bal}}(C) = \max\{w(C), h(C)\} \quad \text{or} \quad \mathrm{size}_{\mathrm{area}}(C) = w(C) \cdot h(C)$$

depending on the chosen objective.

For any candidate attachment of a string, we can (i) check whether overlapping cells agree symbolically, and (ii) compute the resulting bounding-box cost.

For a given target side length L in the balanced-area case, we enumerate translations $(\Delta x, \Delta y)$ of a string T that would result in a bounding box of side length exactly L when combined with the current canvas. Intuitively:

- if the canvas is empty, we place the first string so that the bounding box matches its own width/height;
- otherwise, when L equals the current size s, we slide the string in all ways that keep the bounding box within a virtual $s \times s$ square;
- when $L > s$, we consider placements that extend this square along one of its four sides so that the new size becomes exactly L.

Symbol-consistency of these candidate placements is checked against the canvas. For the area-based variant we similarly enumerate candidate positions and evaluate them according to the area-based cost.

4.3.1. Deterministic Tree-Growing

The deterministic variant starts from a chosen root string T_r (placed at the origin) and iteratively attaches the remaining strings. At each step, it considers all candidate edges (i, j, δ) where T_i is already in the tree and T_j is not, and selects the edge that:

(i) achieves the smallest possible increase in the chosen bounding-box cost, and
(ii) among those, maximizes the *character overlap*, the number of overlapping cells containing matching non-wildcard symbols, as a tie-breaker.

Rationale for the two-level criterion. The primary criterion (minimizing bounding-box cost increase) is equivalent to maximizing geometrical overlap: as shown in Section 4.2, if the current canvas has area A and the new string has area A_j, then

$$\Delta A = A_j - |\text{geometrical overlap}|,$$

so minimizing ΔA is the same as maximizing geometrical overlap. The secondary criterion (character overlap) serves as a tie-breaker when multiple attachments achieve the same bounding-box cost. This design reflects a key insight: in the 2D setting, controlling the bounding-box shape is the primary concern, but among equally good placements (in terms of bounding box), preferring those with more character matches encourages compression and may improve solution quality.

The algorithm terminates when all n strings have been added to the tree. By construction, it always increases the cost threshold until a consistent attachment exists for some string, and therefore returns a complete placement tree spanning all strings.

This procedure directly implements the "growing" perspective of Corollary 7: we construct a spanning tree of the contact graph by adding one vertex at a time, always maintaining a valid partial placement.

4.3.2. Stochastic Tree-Growing

The stochastic variant follows the same tree-growing structure but introduces randomization in the selection among equally good candidate edges. Instead of deterministically picking the single best attachment, it uses *roulette wheel selection* among all candidates that achieve the minimal cost increase, with selection probabilities proportional to their *character overlap* (number of matching non-wildcard symbols). Specifically, for candidates $\{(i_1, j_1, \delta_1), \dots, (i_k, j_k, \delta_k)\}$ all achieving cost c^*, the probability of selecting candidate ℓ is

$$P(\ell) = \frac{\text{char\_overlap}(i_\ell, j_\ell, \delta_\ell)}{\sum_{m=1}^{k} \text{char\_overlap}(i_m, j_m, \delta_m)},$$

where char_overlap(i, j, δ) denotes the number of overlapping cells containing matching non-wildcard symbols when T_j is placed at offset δ relative to T_i. This weighting biases the selection toward attachments with more character matches (favoring compression) while still allowing exploration of alternative placements.

This stochastic heuristic often yields slightly worse single solutions than the deterministic variant, but it produces a diverse set of placement trees, which is useful for initializing the genetic algorithm population. The diversity arises because different runs may grow the tree in different orders, exploring distinct regions of the solution space.

The tree-growing procedure is summarized in Algorithm 2.

4.4. Tree-based genetic algorithm

We now describe the tree-based genetic algorithm (GA) used in our experiments. More precisely, our approach is a *Memetic Algorithm* (Moscato, 1989), a hybrid that couples global evolutionary search with a problem-specific local completion operator. This design reflects a modern consensus in combinatorial optimization: "pure" GAs rarely compete with state-of-the-art methods, but GAs hybridized with domain-specific heuristics consistently achieve top performance on structured problems (Merz and Freisleben, 2000). In our setting, the evolutionary process explores the *high-level topology*, which strings cluster together and how subtrees combine, while the greedy completion operator handles the *low-level boundary repair*, attaching the small fraction of strings that cannot be consistently inherited from parents. This division of labor is deliberate: the expensive evolutionary search focuses on the critical structural backbone of the placement tree, while trivial leaf attachments are delegated to fast, deterministic (or stochastic) greedy. Recall that a solution is represented as a *placement tree* $F = (V, E, r)$ whose vertices are strings, whose directed edges $(u \to v)$ are annotated by integer offsets $(\Delta x, \Delta y)$, and whose root r is the string placed at the origin. Decoding such a tree yields a concrete placement and an objective value.

As a preprocessing step we build a *placement graph* $G = (V, E_G)$ on the strings, whose directed edges encode all locally valid relative placements between string pairs. For each ordered pair of distinct strings (u, v) we enumerate translations of v relative to u within the bounded search window established by Lemma 3: $|\Delta x| \le w_u + w_v - 1$ and $|\Delta y| \le h_u + h_v - 1$. We collect all offsets within this window that yield symbol-consistent overlaps or 4-adjacent contact. This bounded enumeration is crucial for efficiency: for uniform $w \times h$ strings, we examine $O(wh)$ candidate offsets per pair rather than an unbounded search space.

Algorithm 2 Tree-Growing Greedy for 2D-SSP

```
Require: Set of 2D strings $\mathcal{T} = \{T_1, \dots, T_n\}$, root index $r$, objective cost $\in \{\text{area}, \text{bal}\}$, mode $\in \{\textsc{Det}, \textsc{Stoch}\}$
Ensure: Placement tree $F$ with placement $p_F$
$V_F \leftarrow \{r\}$; $E_F \leftarrow \emptyset$; $p_F(r) \leftarrow (0,0)$
$canvas \leftarrow$ cells of $T_r$ at origin
while $|V_F| < n$ do
    $candidates \leftarrow \emptyset$
    for each $i \in V_F$, each $j \notin V_F$, each valid offset $\delta$ do
        if attaching $T_j$ at $p_F(i) + \delta$ is symbol-consistent with $canvas$ then
            $c \leftarrow \text{cost}(canvas \cup \text{cells of } T_j \text{ at } p_F(i) + \delta)$
            $candidates \leftarrow candidates \cup \{(i, j, \delta, c)\}$
        end if
    end for
    $c^* \leftarrow \min\{c : (i, j, \delta, c) \in candidates\}$
    $best \leftarrow \{(i, j, \delta) : (i, j, \delta, c^*) \in candidates\}$
    if mode = $\textsc{Det}$ then
        $(i^*, j^*, \delta^*) \leftarrow$ element of $best$ maximizing overlap
    else
        // Roulette wheel selection weighted by overlap
        $w_\ell \leftarrow \text{overlap}(i_\ell, j_\ell, \delta_\ell)$ for each $(i_\ell, j_\ell, \delta_\ell) \in best$
        $(i^*, j^*, \delta^*) \leftarrow$ sample from $best$ with $P(\ell) \propto w_\ell$
    end if
    $V_F \leftarrow V_F \cup \{j^*\}$; $E_F \leftarrow E_F \cup \{(i^*, j^*, \delta^*)\}$
    $p_F(j^*) \leftarrow p_F(i^*) + \delta^*$
    Update $canvas$ with cells of $T_{j^*}$ at $p_F(j^*)$
end while
return $(F, p_F)$
```

To evaluate a tree $F = (V, E, r)$ we traverse it from the root r, assign absolute coordinates $p_F(i)$ to every 2D string T_i by summing the edge offsets along the unique path from r to i, and construct the global canvas, rejecting any edge that would create a symbol conflict. The fitness is then the bounding-box cost of the resulting placement, either

$$\text{cost}_{\text{area}}(p_F) = W(p_F) \cdot H(p_F) \quad \text{or} \quad \text{cost}_{\text{bal}}(p_F) = \max\{W(p_F), H(p_F)\},$$

depending on which objective variant is being optimized.

Each individual in the initial population is obtained by running a greedy constructive heuristic from a given start string, producing a full placement with absolute coordinates; we then extract a spanning tree of relative offsets. Thus every individual faithfully encodes the relative structure of a greedy solution.

We consider two variants that differ in their use of deterministic versus stochastic greedy:

- **T-GA** (Tree-based GA): Uses *deterministic* tree-growing greedy (DET mode) for both population initialization and greedy completion. Since deterministic greedy produces the same tree for a given root, initial population members differ *only* in their choice of starting root. This limits initial diversity to n distinct individuals.

- **ST-GA** (Stochastic Tree-based GA): Uses *stochastic* tree-growing greedy (STOCH mode) for both population initialization and greedy completion. Each initial individual is generated by an independent stochastic run, introducing diversity at the population level. Greedy completion during crossover is also stochastic, allowing different repair trajectories for incomplete offspring.

Our *locality-preserving crossover* operator combines two parent trees by *alternating* their local tree structures while maintaining geometric feasibility. This design directly implements the building block hypothesis: rather than mixing raw coordinates, we transplant entire subtrees, thereby preserving the *schema*, the pattern of relative offsets that makes a cluster of strings fit together well. Starting from the root, we expand a child tree by re-using parent edges

Algorithm 3 Tree-Based Genetic Algorithm for 2D-SSP

```
Require: Strings $\mathcal{T}$, population size $N$, generations $G$, crossover rate $\rho$, elite fraction $\epsilon$, mode $\in \{\textsc{Det}, \textsc{Stoch}\}$
Ensure: Best placement tree $F^*$
// Initialization
for $k = 1$ to $N$ do
    $r_k \leftarrow$ random root from $\{1, \dots, n\}$
    $P[k] \leftarrow \textsc{TreeGrowingGreedy}(\mathcal{T}, r_k, \text{mode})$        ▷ Det for T-GA, Stoch for ST-GA
end for
// Main loop
for $g = 1$ to $G$ do
    Evaluate fitness $f[k] \leftarrow \text{cost}(P[k])$ for all $k$
    Sort population by fitness (ascending)
    $P' \leftarrow$ copy of top $\lfloor \epsilon N \rfloor$ individuals        ▷ Elitism
    while $|P'| < N$ do
        if rand() $< \rho$ then
            Select parents $F_1, F_2$ via tournament selection
            $F_{\text{child}} \leftarrow \textsc{TreeCrossover}(F_1, F_2, \text{mode})$
        else
            $F_{\text{child}} \leftarrow$ copy of tournament-selected individual
        end if
        $P' \leftarrow P' \cup \{F_{\text{child}}\}$
    end while
    $P \leftarrow P'$
end for
return $\arg\min_{F \in P} \text{cost}(F)$
```

whenever they can be realized without conflicts on the canvas. If some strings cannot be connected using parent edges alone, we perform a final *greedy completion* step to attach all remaining strings. The greedy completion ensures that every crossover produces a complete tree containing all strings, even when parental structures are incompatible.

The full GA is summarized in Algorithm 3, and the crossover operator in Algorithm 4. We maintain a population of trees, initialized from greedy placements with different starting strings (T-GA) or independent stochastic runs (ST-GA). In each generation we decode all individuals, rank them by fitness, copy the best few (elitism (De Jong, 1975)), and fill the remaining slots by crossover or by copying fit parents.

Our GA deliberately omits a dedicated mutation operator. This design choice is motivated by two considerations. First, when crossover is carefully designed to recombine meaningful building blocks, as our subtree-based crossover does, explicit mutation often provides diminishing returns or can even be counterproductive by disrupting well-structured solutions. This observation aligns with findings in other combinatorial optimization domains where problem-specific crossover operators dominate the search dynamics (Sholomon et al., 2013). Second, maintaining feasibility under traditional mutation is non-trivial in our setting: a random perturbation of edge offsets in a placement tree can easily introduce symbol conflicts or break connectivity.

However, we observe that the *greedy completion* step in our crossover operator (Algorithm 4, lines 18–22) implicitly serves as a *feasibility-preserving mutation mechanism*. When crossover produces an incomplete offspring, one that does not contain all strings, the greedy completion step attaches the missing strings using fresh, locally-optimal placements that differ from both parents. In the stochastic variant (GA (Stochastic)), this completion step uses *stochastic greedy* placement, which randomly samples among equally-good candidate positions rather than deterministically choosing one. This randomization introduces genuine variation: even when the same set of strings must be completed, different runs may attach them at different positions, exploring alternative regions of the solution space.

This design offers several advantages over traditional mutation:

1. *Guaranteed feasibility.* Unlike random perturbations of tree edges, greedy completion always produces symbol-consistent placements by construction.

Algorithm 4 Locality-Preserving Tree Crossover with Greedy Completion

```
Require: Parent trees $F_1 = (V, E_1, r_1)$, $F_2 = (V, E_2, r_2)$, mode $\in \{\textsc{Det}, \textsc{Stoch}\}$
Ensure: Child tree $F_c$
$r_c \leftarrow r_1$                                                    ▷ Inherit root from first parent
$V_c \leftarrow \{r_c\}$; $E_c \leftarrow \emptyset$; $p_c(r_c) \leftarrow (0, 0)$
$canvas \leftarrow$ cells of $T_{r_c}$ at origin
$Q \leftarrow [r_c]$                                                    ▷ Queue for BFS expansion
while $Q \neq \emptyset$ do
    $i \leftarrow Q.\textsc{Dequeue}()$
    for each child $j$ of $i$ in $F_1$ or $F_2$ with $j \notin V_c$ do
        $\delta \leftarrow$ offset of edge $(i, j)$ in the parent tree
        if attaching $T_j$ at $p_c(i) + \delta$ is symbol-consistent with $canvas$ then
            $V_c \leftarrow V_c \cup \{j\}$; $E_c \leftarrow E_c \cup \{(i, j, \delta)\}$
            $p_c(j) \leftarrow p_c(i) + \delta$
            Update $canvas$; $Q.\textsc{Enqueue}(j)$
        end if
    end for
end while
// Greedy completion for missing strings
if $|V_c| < n$ then
    $missing \leftarrow V \setminus V_c$
    for each $j \in missing$ do
        $(i^*, \delta^*) \leftarrow$ best attachment via greedy (mode)          ▷ Det or Stoch
        $V_c \leftarrow V_c \cup \{j\}$; $E_c \leftarrow E_c \cup \{(i^*, j, \delta^*)\}$
        $p_c(j) \leftarrow p_c(i^*) + \delta^*$; update $canvas$
    end for
end if
return $F_c = (V_c, E_c, r_c)$
```

2. *Adaptive intensity.* The amount of "mutation" adapts to the compatibility of the parents: when parent structures are highly compatible, few strings require completion and offspring closely resemble parents; when parents are incompatible, many strings are completed afresh, introducing substantial variation.
3. *Local optimization.* Newly attached strings are placed greedily, ensuring that the mutated portion of the solution is locally reasonable rather than random.

Table 5 confirms that crossover dominates solution construction (95–97.5% of placements), with greedy completion providing lightweight repair for 3–5% of placements. See Appendix A.8 for detailed analysis of population dynamics.

Our experimental analysis (Section 5, Table 5) shows that approximately 3–5% of string placements arise from greedy completion, providing a consistent but moderate level of exploration that complements the exploitation performed by crossover.

In all experiments we fix the crossover rate $\rho = 0.7$ and an elite fraction of 10% of the population. Unless otherwise stated, the objective used in selection is the area-based cost of the decoded placement. The same GA can be run with the balanced-area cost by simply changing the fitness function to cost_{bal}, and we report results for both objective variants.

5. Experiments

This section presents our experimental evaluation. We first describe the experimental setup (Section 5.1), then report results comparing algorithms across different problem scales and string geometries (Section 5.2), and finally analyze the internal dynamics of the genetic algorithm (Section 5.3).

5.1. Experimental Setup

Hardware and implementation. All experiments were conducted on a system with an Intel Core i5-13400F processor and 32 GB RAM. The ILP formulation was solved using IBM ILOG CPLEX 22.1. All heuristic algorithms were implemented in Python.

Instance generation. For each configuration, we generated 10 synthetic instances consisting of random binary 2D strings (alphabet $\Sigma = \{0, 1\}$), where each cell is assigned 0 or 1 uniformly at random with probability 0.5. The same 10 instances were used across all algorithms within each configuration to ensure fair comparison.

Table 2 presents results for configurations mixing 1D-like and 2D string shapes with the *area* objective. Table 3 reports results for genuinely 2D strings with the *balanced-area* objective.

Experimental configurations. We designed four experimental configurations to systematically evaluate algorithm performance across different scales and string geometries:

1. **1D-like instances** (`1d`): Strings with extreme aspect ratios (1 × 2, 1 × 4, 1 × 8) that closely resemble classical 1D strings. Instance sizes: $n \in \{10, 20, 50, 100, 200, 300\}$ strings. This configuration uses the *area* objective ($H \cdot W$) and tests whether merge-based greedy retains its 1D effectiveness. GA parameters: population size 150, 300 generations.
2. **Small instances** (`small`): Genuinely 2D strings (3 × 3, 2 × 4, 5 × 5, 4 × 6) with $n \in \{6, 8, 10\}$ strings. This configuration includes CPLEX as a baseline (time limit: 300 s) to validate heuristic quality against optimal solutions. Uses the *balanced-area* objective ($\max\{H, W\}$). GA parameters: population size 100, 200 generations.
3. **Medium instances** (`medium`): Same string shapes as `small`, but with $n \in \{20, 30, 50\}$ strings. CPLEX is excluded due to computational intractability. Uses the *balanced-area* objective. GA parameters: population size 150, 300 generations.
4. **Large instances** (`large`): Same string shapes, with $n \in \{60, 80, 100\}$ strings for scalability testing. Uses the *balanced-area* objective. GA parameters: population size 200, 400 generations.

Algorithms compared. We evaluate five heuristic algorithms plus an exact solver:

- `CPLEX`: Exact ILP solver (small instances only, 300 s time limit).
- `M-Greedy`: Merge-based greedy (Algorithm 1).
- `T-Greedy`: Tree-growing greedy with deterministic tie-breaking.
- `ST-Greedy`: Tree-growing greedy with stochastic tie-breaking.
- `T-GA`: Genetic algorithm using deterministic `T-Greedy` for both population initialization and greedy completion. Initial population members differ only in the choice of starting root.
- `ST-GA`: Genetic algorithm using stochastic `ST-Greedy` for both population initialization and greedy completion. This introduces diversity at both stages: each initial individual is generated by an independent stochastic greedy run, and incomplete offspring are completed stochastically.

All GA variants use tournament selection with size 3 and crossover probability 0.7.

Performance metrics. For each algorithm and configuration, we report the objective value (mean ± standard deviation over 10 instances) and mean runtime. Best results per configuration are shown in bold.

5.2. Results

Key observations (area objective, Table 2).

- Tree-based methods outperform merge-greedy by 9–15% on genuinely 2D instances.
- Merge-greedy matches tree-based methods on 1D-like (1 × 8) instances.
- GA variants improve over greedy by 6–12% across all 2D configurations.
- Stochastic tie-breaking (`ST-GA`) consistently achieves best objective values.

Config	M-Greedy	T-Greedy	ST-Greedy	T-GA	ST-GA
T6_n3_m3	48.30 ± 4.50 (0.008s)	43.63 ± 5.17 (0.002s)	44.13 ± 4.75 (**0.002**s)	41.07 ± 3.82 (0.108s)	**40.03** ± **3.60** (0.133s)
T6_n5_m5	172.10 ± 14.19 (0.028s)	144.00 ± 3.00 (**0.025**s)	143.00 ± 2.45 (0.027s)	143.00 ± 2.45 (0.491s)	**142.50** ± **2.50** (1.066s)
T8_n3_m3	59.40 ± 6.53 (0.014s)	56.23 ± 4.39 (**0.003**s)	57.31 ± 5.12 (0.003s)	53.31 ± 4.01 (0.142s)	**51.46** ± **3.20** (0.183s)
T8_n5_m5	223.50 ± 12.34 (0.084s)	195.20 ± 10.02 (0.053s)	196.20 ± 9.43 (**0.053**s)	190.50 ± 2.69 (0.847s)	**190.00** ± **3.16** (2.472s)
T10_n1_m2	**4.50** ± **0.50** (0.004s)	4.60 ± 0.66 (0.001s)	4.70 ± 0.78 (**0.001**s)	**4.50** ± **0.50** (0.089s)	**4.50** ± **0.50** (0.075s)
T10_n1_m4	14.00 ± 2.10 (0.015s)	13.80 ± 1.99 (**0.001**s)	13.90 ± 2.02 (0.002s)	13.40 ± 1.56 (0.140s)	**13.20** ± **1.54** (0.152s)
T10_n1_m8	45.90 ± 4.30 (0.057s)	49.50 ± 3.56 (**0.007**s)	48.20 ± 4.98 (0.007s)	45.30 ± 4.20 (0.288s)	**44.80** ± **4.38** (0.400s)
T10_n3_m3	71.80 ± 7.90 (0.019s)	69.20 ± 5.10 (0.005s)	69.10 ± 5.72 (**0.005**s)	62.30 ± 4.80 (0.195s)	**59.00** ± **2.72** (0.266s)
T10_n5_m5	277.00 ± 19.61 (0.202s)	240.00 ± 5.92 (**0.089**s)	241.00 ± 5.83 (0.089s)	237.00 ± 5.57 (1.530s)	**236.00** ± **5.39** (4.976s)
T20_n1_m2	5.10 ± 0.54 (0.013s)	5.20 ± 0.60 (**0.001**s)	5.10 ± 0.54 (0.001s)	**4.90** ± **0.30** (0.162s)	**4.90** ± **0.30** (0.154s)
T20_n1_m4	18.10 ± 1.87 (0.046s)	18.30 ± 1.10 (0.003s)	19.10 ± 1.97 (**0.003**s)	16.80 ± 0.75 (0.331s)	**16.50** ± **0.50** (0.509s)
T20_n1_m8	**71.40** ± **6.41** (0.415s)	77.00 ± 7.97 (**0.025**s)	77.30 ± 5.92 (0.045s)	74.00 ± 5.67 (0.880s)	**71.40** ± **6.23** (1.416s)
T20_n3_m3	113.40 ± 8.89 (0.130s)	120.30 ± 8.26 (**0.010**s)	122.60 ± 10.76 (0.013s)	104.20 ± 5.06 (1.041s)	**96.10** ± **3.81** (1.628s)
T20_n5_m5	518.40 ± 21.12 (3.390s)	470.00 ± 11.40 (0.562s)	467.50 ± 9.01 (**0.552**s)	459.00 ± 8.31 (18.089s)	**452.00** ± **6.00** (32.016s)
T30_n3_m3	143.10 ± 12.51 (0.396s)	154.70 ± 9.95 (**0.021**s)	164.80 ± 11.70 (0.024s)	138.40 ± 7.14 (2.821s)	**131.90** ± **7.02** (3.518s)
T30_n5_m5	752.10 ± 25.58 (15.051s)	28.40 ± 0.66 (**0.017**s)	29.00 ± 0.45 (0.024s)	**27.10** ± **0.30** (58.713s)	**27.10** ± **0.30** (90.580s)
T50_n1_m2	5.50 ± 0.50 (0.148s)	5.20 ± 0.40 (**0.003**s)	5.40 ± 0.49 (0.003s)	**5.00** ± **0.00** (0.653s)	**5.00** ± **0.00** (0.656s)
T50_n1_m4	19.30 ± 1.10 (0.626s)	21.10 ± 1.92 (0.004s)	20.90 ± 1.70 (**0.004**s)	18.50 ± 0.67 (1.258s)	**18.40** ± **0.49** (1.716s)
T50_n1_m8	**123.60** ± **13.51** (5.470s)	135.10 ± 14.82 (**0.148**s)	136.50 ± 13.03 (0.191s)	131.30 ± 13.84 (7.273s)	127.10 ± 13.40 (6.853s)
T50_n3_m3	211.10 ± 16.83 (1.806s)	227.30 ± 14.45 (**0.086**s)	236.60 ± 15.92 (0.097s)	197.90 ± 6.70 (13.097s)	**191.33** ± **9.01** (18.018s)
T50_n5_m5	1168.00 ± 16.00 (109.201s)	1127.50 ± 12.50 (**6.604**s)	1122.50 ± 12.50 (6.623s)	1102.50 ± 7.50 (214.734s)	**1100.00** ± **0.00** (311.548s)
T100_n1_m2	**5.00** ± **0.00** (1.188s)	5.20 ± 0.40 (**0.004**s)	5.40 ± 0.49 (0.004s)	**5.00** ± **0.00** (1.628s)	**5.00** ± **0.00** (1.619s)
T100_n1_m4	20.60 ± 1.43 (4.948s)	20.00 ± 1.10 (0.007s)	21.30 ± 1.55 (**0.006**s)	19.30 ± 0.46 (3.078s)	**19.00** ± **0.00** (4.048s)
T100_n1_m8	**178.20** ± **11.41** (37.071s)	198.50 ± 11.72 (0.399s)	196.10 ± 10.15 (**0.380**s)	185.10 ± 8.94 (25.846s)	183.40 ± 9.32 (25.341s)
T200_n1_m2	5.33 ± 0.47 (9.418s)	**5.00** ± **0.00** (**0.008**s)	5.33 ± 0.47 (0.009s)	**5.00** ± **0.00** (4.230s)	**5.00** ± **0.00** (4.374s)

Table 2
Objective type: area. Each cell shows objective (top) as mean ± std and runtime is mean only (bottom).

Key observations (balanced-area objective, Table 3).

- Even stronger relative performance: ST-GA reduces cost by > 50% vs. merge-greedy on some configurations.
- Merge-greedy's overlap-maximization is poorly suited for aspect-ratio control.
- All heuristics scale to $n = 50$ strings; GA runtimes grow to ~230 s on large instances.

Config	M-Greedy	T-Greedy	ST-Greedy	T-GA	ST-GA
T6_n3_m3	7.70 ± 0.90 (0.007s)	7.33 ± 0.54 (**0.001**s)	7.47 ± 0.56 (**0.001**s)	7.00 ± 0.37 (0.165s)	**6.77 ± 0.42** (0.225s)
T6_n5_m5	15.30 ± 1.27 (0.030s)	14.90 ± 1.04 (**0.001**s)	14.80 ± 0.60 (0.001s)	13.50 ± 0.50 (0.476s)	**13.10 ± 0.30** (1.376s)
T6_n10_m10	33.00 ± 1.73 (0.681s)	31.80 ± 1.25 (**0.009**s)	31.10 ± 1.22 (0.009s)	29.60 ± 0.66 (5.311s)	**28.90 ± 0.30** (14.190s)
T8_n3_m3	8.60 ± 0.92 (0.013s)	8.23 ± 0.56 (**0.001**s)	8.40 ± 0.55 (0.001s)	7.77 ± 0.42 (0.304s)	**7.37 ± 0.48** (0.285s)
T8_n5_m5	17.40 ± 1.28 (0.088s)	16.30 ± 0.64 (**0.002**s)	16.60 ± 0.66 (0.002s)	15.10 ± 0.30 (1.561s)	**15.00 ± 0.00** (2.793s)
T8_n10_m10	42.40 ± 3.93 (2.267s)	34.30 ± 1.27 (0.016s)	34.10 ± 1.51 (**0.016**s)	31.70 ± 0.90 (24.422s)	**30.60 ± 0.49** (40.423s)
T10_n3_m3	9.20 ± 0.40 (0.020s)	8.97 ± 0.41 (0.001s)	9.13 ± 0.43 (**0.001**s)	8.17 ± 0.37 (0.237s)	**8.03 ± 0.18** (0.380s)
T10_n5_m5	19.50 ± 2.91 (0.202s)	17.90 ± 0.70 (**0.002**s)	18.20 ± 0.40 (0.003s)	16.70 ± 0.46 (1.786s)	**16.50 ± 0.50** (6.692s)
T10_n10_m10	51.50 ± 5.00 (5.939s)	39.60 ± 1.20 (**0.024**s)	39.50 ± 0.50 (0.027s)	37.10 ± 0.54 (58.876s)	**36.90 ± 0.30** (111.969s)
T20_n3_m3	11.80 ± 0.75 (0.127s)	11.33 ± 0.47 (0.002s)	11.77 ± 0.56 (**0.002**s)	10.27 ± 0.44 (1.280s)	**10.13 ± 0.34** (2.314s)
T20_n5_m5	26.40 ± 3.32 (3.331s)	24.00 ± 0.45 (**0.008**s)	24.00 ± 0.77 (0.010s)	**22.60 ± 0.49** (16.813s)	22.80 ± 0.40 (27.894s)
T20_n10_m10	105.50 ± 2.50 (129.009s)	53.60 ± 1.11 (**0.113**s)	53.40 ± 0.92 (0.135s)	50.30 ± 0.64 (739.766s)	**50.00 ± 0.00** (1088.163s)
T30_n3_m3	13.80 ± 1.72 (0.391s)	12.83 ± 0.52 (**0.003**s)	13.57 ± 0.56 (0.004s)	11.93 ± 0.36 (2.897s)	**11.90 ± 0.30** (5.085s)
T30_n5_m5	30.60 ± 2.76 (15.057s)	28.70 ± 1.00 (**0.019**s)	29.10 ± 0.70 (0.030s)	**27.10 ± 0.30** (75.373s)	27.40 ± 0.49 (108.513s)
T50_n3_m3	16.30 ± 1.19 (1.793s)	15.00 ± 0.52 (**0.006**s)	15.77 ± 0.80 (0.008s)	**14.13 ± 0.34** (12.208s)	14.20 ± 0.48 (18.474s)
T50_n5_m5	35.50 ± 0.50 (109.531s)	35.00 ± 0.00 (**0.055**s)	36.00 ± 0.00 (0.076s)	**33.50 ± 0.50** (236.194s)	35.00 ± 0.00 (358.765s)

Table 3
Objective type: `square`. Each cell shows objective (top) as mean ± std and runtime is mean only (bottom).

Config	CPLEX	M-Greedy	T-Greedy	ST-Greedy	T-GA	ST-GA
T6_n3_m3	**6.60 ± 0.49** (1.194s)	7.70 ± 0.90 (0.007s)	7.33 ± 0.54 (**0.001**s)	7.47 ± 0.56 (**0.001**s)	7.00 ± 0.37 (0.165s)	6.77 ± 0.42 (0.225s)
T8_n3_m3	**7.30 ± 0.46** (18.737s)	8.60 ± 0.92 (0.013s)	8.23 ± 0.56 (**0.001**s)	8.40 ± 0.55 (0.001s)	7.77 ± 0.42 (0.304s)	7.37 ± 0.48 (0.285s)
T10_n3_m3	**8.00 ± 0.00** (127.740s)	9.20 ± 0.40 (0.020s)	8.97 ± 0.41 (0.001s)	9.13 ± 0.43 (**0.001**s)	8.17 ± 0.37 (0.237s)	8.03 ± 0.18 (0.380s)
T20_n3_m3	11.00 ± 0.53 (201.079s)	11.80 ± 0.75 (0.127s)	11.33 ± 0.47 (0.002s)	11.77 ± 0.56 (**0.002**s)	10.27 ± 0.44 (1.280s)	**10.13 ± 0.34** (2.314s)
T30_n3_m3	14.00 ± 0.00 (203.087s)	13.80 ± 1.72 (0.391s)	12.83 ± 0.52 (**0.003**s)	13.57 ± 0.56 (0.004s)	11.93 ± 0.36 (2.897s)	**11.90 ± 0.30** (5.085s)

Table 4
Objective type: `square`. Each cell shows objective (top) as mean ± std and runtime is mean only (bottom).

Comparison with CPLEX (Table 4). On small instances where CPLEX finds provably optimal solutions, `ST-GA` achieves optimality gaps of 0.4–2.6%. On larger instances (T20–T30), `ST-GA` outperforms CPLEX (which hits the 300 s time limit), validating the GA's effectiveness.

Runtime trade-offs. Greedy methods complete in milliseconds; GA variants require seconds to minutes. This defines a Pareto frontier: greedy for interactive or exploratory use, GA for offline optimization where solution quality justifies additional computation time.

Algorithm	Scale	Mean #crossovers	Repair rate r_{repair}	Direct strings ρ_{direct}	Avg. strings/repair $\bar{t}_{\text{repair}}$
GA	small	3 152	9.2%	97.0%	3.0
GA	medium	12 594	6.7%	97.0%	22.6
GA	large	28 357	3.1%	97.5%	68.3
GA (Stochastic)	small	3 151	12.8%	95.9%	2.7
GA (Stochastic)	medium	12 602	8.9%	95.2%	22.0
GA (Stochastic)	large	28 354	6.3%	96.1%	60.8

Table 5
Internal statistics of the genetic algorithms. "Direct strings" is the fraction of string placements coming directly from crossover ($\rho_{\text{direct}} = 1 - \rho_{\text{greedy}}$); the remainder are filled by the greedy completion procedure. All experiments here use the area-based objective.

Visual comparison. Figure **??** illustrates the qualitative difference on a $N = 50$ instance: the GA solution achieves a ∼10% smaller bounding box through tight interlocking that greedy's myopic decisions cannot discover.

5.3. Genetic Algorithm Dynamics

5.3.1. *Crossover statistics*

To better understand how the genetic algorithms construct their solutions, we instrument GA and GA (Stochastic) with additional counters. For each run we record:

- the total number of crossover operations, C (`total_crossovers`);
- the number of crossovers that produce an incomplete placement and therefore require greedy completion, C_{repair} (`crossovers_needing_completion`);
- the total number of strings that are finally placed by the greedy completion step across all incomplete offspring, T_{fix} (`total_strings_completed`).

From these quantities we derive:

$$r_{\text{repair}} = \frac{C_{\text{repair}}}{C} \qquad \text{repair rate: fraction of crossovers that need greedy fix}$$

$$\rho_{\text{greedy}} = \frac{T_{\text{fix}}}{Cn} \qquad \text{share of string placements coming from greedy completion}$$

$$\rho_{\text{direct}} = 1 - \rho_{\text{greedy}} \qquad \text{“completion rate”: share of strings placed directly by crossover}$$

$$\bar{t}_{\text{repair}} = \frac{T_{\text{fix}}}{C_{\text{repair}}} \qquad \text{average \#missing strings per repaired offspring} \quad .$$

Several trends are apparent:

- The number of crossovers per run grows with instance size. On small instances each GA performs about 3.1×10^3 crossovers; this increases to roughly 1.26×10^4 on medium instances and 2.8×10^4 on large instances.
- The repair rate r_{repair} is modest: between 3% and 13% of crossovers produce offspring that are not complete placements. The GA (Stochastic) variant tends to trigger repairs slightly more often than GA.
- At the string level, only a very small fraction of the solution is delegated to the greedy completion phase. Across all scales and both GA variants, the mean ρ_{greedy} is about 3.6%, so approximately 96% of string placements come directly from crossover. In other words, the GA's recombination operators are doing almost all of the constructive work.
- When a repair is needed, it can be substantial on large instances: on small instances, an incomplete offspring is missing on average 3 strings out of 8; on medium instances, around 22 out of 33 strings; and on large instances, around 61–68 out of roughly 80 strings. This reflects the increasing difficulty of producing fully consistent placements purely by recombination when the search space grows.

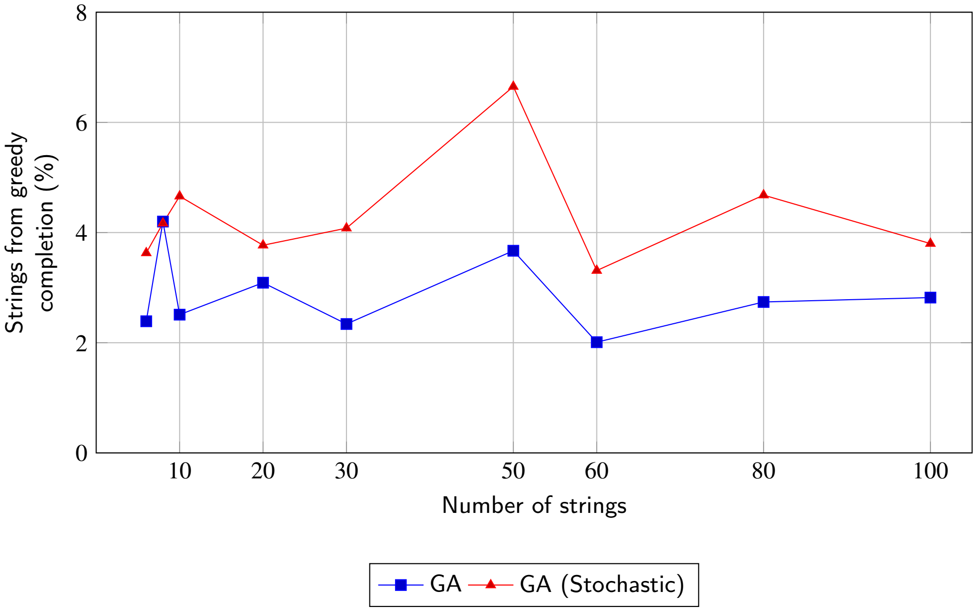


Figure 3: Share of string placements produced by the greedy completion step as a function of the number of strings. The complement to 100% can be interpreted as the "completion rate" of the crossover operators.

The role of greedy completion. Table 5 shows that 95–97.5% of string placements are inherited from parents via crossover; greedy completion acts as lightweight boundary repair for the remaining 3–5%. The evolutionary search over tree structures is the primary driver of solution quality.

Figure 3 focuses on the string-level interaction between crossover and greedy repair by plotting ρ_{greedy} (the fraction of strings placed by the greedy completion step) as a function of the number of strings. For both GA variants this fraction remains between roughly 2% and 7% across all sizes, confirming that the vast majority of strings in the final placements are produced by crossover rather than by the repair heuristic.

6. Conclusion

This paper has introduced the Two-Dimensional Shortest Superstring Problem (2D-SSP), establishing it as a rich combinatorial optimization problem unifying string sequencing with geometric packing.

Theoretical foundations. We established NP-hardness for both objectives and APX-hardness for area via L-reduction from 1D-SSP. The *Bounded-Offset Tree Representation* transforms 2D-SSP from geometric optimization over infinite coordinates into a finite combinatorial problem over spanning trees. The connectivity/compaction theorem (Theorem 4) proves optimal solutions can be made 4-connected without increasing cost.

Algorithmic innovation. Our Memetic Algorithm (T-GA) features locality-preserving crossover that preserves beneficial subtree structures. Crossover dominates solution construction ($>$ 95% of placements), with greedy completion providing lightweight repair. The GA achieves optimality gaps $\leq$ 2.6% on ILP-verifiable instances and outperforms greedy baselines by 6–12% on larger instances.

Extensions. The theoretical framework extends directly to d-dimensional SSP for any $d \geq 1$, with $2d$-adjacency replacing 4-adjacency; see Appendix A.6.

Limitations. The ILP does not scale beyond $n \leq 10$. The gap between existence of optimal trees and the GA's greedy-completion subspace remains theoretically open. Experiments focus on binary alphabets; behavior on higher-entropy instances is unexplored.

Future directions. Key open problems include: (1) approximation algorithms generalizing 1D-SSP's 2.5-approximation; (2) APX-hardness of 2D-SSP$_{\text{bal}}$ in the sequencing regime; (3) scaling exact methods via decomposition or column generation; and (4) extension to non-rectangular patterns.

A. Supplementary Technical Details

This appendix collects extended discussions moved from the main text for space reasons.

A.1. A planar crossing lemma for the grid

The proof of Theorem 4 uses the following standard separation property of the rectangular grid graph.

Lemma 8 (Grid crossing)**.** *Let $W, H \geq 1$ and consider the grid graph with vertex set*

$$V := \{0, \dots, W-1\} \times \{0, \dots, H-1\}$$

and edges between 4-neighbours (Manhattan distance 1). Let P be a vertex-simple path in this graph from the top side (some vertex with $y = H-1$) to the bottom side (some vertex with $y = 0$). Let Q be a vertex-simple path from the left side (some vertex with $x = 0$) to the right side (some vertex with $x = W-1$). Then P and Q share a vertex.

Proof. Embed the grid graph in the plane by placing each vertex (x, y) at its Euclidean position and drawing each edge as a straight-line segment. The path P together with the boundary of the rectangle forms a closed curve that separates the left side from the right side in the planar embedding (a direct consequence of the Jordan curve theorem; see, e.g., standard texts on planar graphs). Since Q connects a vertex on the left side to a vertex on the right side using grid edges that do not cross each other, Q must intersect P in the embedding. Because distinct grid edges intersect only at common endpoints, this intersection must occur at a common vertex of P and Q. □

A.2. APX-Hardness Proof Details

Recall that an L-reduction from problem A to problem B requires constants $\alpha, \beta > 0$ such that:

1. $\text{OPT}_B \leq \alpha \cdot \text{OPT}_A$, and
2. for any solution to I_B with cost c_B, one can construct a solution to I_A with cost c_A satisfying $|c_A - \text{OPT}_A| \leq \beta \cdot |c_B - \text{OPT}_B|$.

Condition (1): From Theorem 1, $\text{OPT}_{\text{2D}} = \text{OPT}_{\text{1D}}$, so $\alpha = 1$.

Condition (2): Given any 2D placement with area A, the row-concatenation argument in Theorem 1 produces a 1D superstring of length $\leq A$. Thus $c_{\text{1D}} \leq c_{\text{2D}}$, which implies $c_{\text{1D}} - \text{OPT}_{\text{1D}} \leq c_{\text{2D}} - \text{OPT}_{\text{2D}}$, so $\beta = 1$.

A.3. Complexity Details: Alphabet Size and Sequencing–Packing Spectrum

Approximation complexity of 2D-SSP$_{\text{bal}}$. The L-reduction does *not* extend to 2D-SSP$_{\text{bal}}$. For height-1 strings, multiple rows can reduce the balanced cost: if $\text{OPT}_{\text{1D}} = 100$ and strings can be arranged in 10 rows of width 10 each, the balanced cost becomes $\max\{10, 10\} = 10 < 100$.

Our NP-hardness proof for 2D-SSP$_{\text{bal}}$ (Theorem 2) operates in the *packing regime*: large alphabets force non-overlapping placements, reducing to rectangle packing. However, the *approximation complexity* in the *sequencing regime* (small $|\Sigma|$, frequent overlaps) remains open.

Alphabet size and hardness. The case $|\Sigma| = 1$ is trivial: all strings can be stacked at the origin, yielding bounding box equal to the largest string's dimensions. At the opposite extreme ($|\Sigma| \geq n$ with unique symbols), no overlaps are possible and 2D-SSP reduces to rectangle packing.

SSP vs. Packing: Opposing objectives. In packing problems, the goal is to fit rectangles into a container *without overlap*. In 2D-SSP, overlaps are *encouraged*—they reduce the bounding box. For $|\Sigma| = 1$, 2D-SSP is trivial (maximal overlap permitted) while packing remains NP-hard (no-overlap constraint persists).

Information entropy and the sequencing–packing spectrum. The effective difficulty depends on string entropy:

- *High entropy (random strings, large $|\Sigma|$):* Probability of symbol-consistent overlap decreases exponentially with overlap size. 2D-SSP degenerates toward a packing problem.
- *Low entropy (repetitive strings, small $|\Sigma|$):* Many pairs admit large overlaps. The problem becomes a sequencing problem with combinatorial explosion of valid configurations.

Binary alphabets represent the sequencing regime where 2D-SSP is most distinct from pure geometric packing.

A.4. Edge Density and String Entropy

The bound $O(n^2wh)$ on $|E(G^{\mathrm{pl}})|$ is a worst-case geometric bound. The set of valid offsets C_{ij} decomposes as $C_{ij} = C_{ij}^{\mathrm{adj}} \cup C_{ij}^{\mathrm{ovl}}$.

Adjacency edges $|C_{ij}^{\mathrm{adj}}|$: Always present, contributing $O(w_i + w_j + h_i + h_j)$ edges per pair (the perimeter of the contact region).

Overlap edges $|C_{ij}^{\mathrm{ovl}}|$: For symbols drawn uniformly from Σ, $\Pr[k\text{-cell overlap is consistent}] = |\Sigma|^{-k}$. For random strings over large alphabets, $|C_{ij}^{\mathrm{ovl}}| \approx 0$ and G^{pl} is sparse.

Periodic/low-entropy strings: Many overlaps become symbol-consistent. For $|\Sigma| = 1$, every offset in the contact region is valid, yielding $|C_{ij}| = O(w_iw_j + h_ih_j)$.

A.5. Optimized Offset Enumeration

The naïve approach enumerates all $O(wh)$ candidate offsets per pair. In the sequencing regime, we can improve performance by iterating offsets in order of *bounding-box increase*.

For two strings T_i $(w_i \times h_i)$ and T_j $(w_j \times h_j)$, placing T_j at offset $(\Delta x, \Delta y)$ increases the bounding box by a computable amount $\mathrm{inc}(\Delta x, \Delta y)$. We enumerate offsets in non-decreasing order of inc:

- $\mathrm{inc} = 0$: Full containment.
- $\mathrm{inc} = 1$: Partial overlap leaving 1 row/column exposed.
- …up to $\mathrm{inc} = w_j + h_j - 2$ (4-adjacent contact, no overlap).

For fixed inc, valid offsets form a predictable geometric “frame.” A greedy algorithm can **stop at the first level containing a symbol-consistent offset**. Worst-case complexity remains $O(wh)$, but average-case improves when overlaps are frequent.

A.6. Extension to Higher Dimensions

The theoretical framework extends directly to d-dimensional SSP for any $d \geq 1$:

(i) Geometry: Objects are hyper-rectangles with dimensions $n_1 \times \cdots \times n_d$. Cost becomes volume $\prod_{k=1}^{d} W_k$ or maximum side $\max_k W_k$. Offsets are vectors $\delta \in \mathbb{Z}^d$. The bounded offset property holds: valid offsets are bounded by the sum of dimensions along each axis.

(ii) Connectivity lemma: The sliding argument works in $\mathbb{Z}^d$: disconnected components A and B can be translated along a cardinal axis through empty space until they share a $(d-1)$-dimensional face, without increasing any bounding-box dimension.

(iii) Graph representation: 4-connectivity generalizes to $2d$-connectivity (sharing a $(d-1)$-face). The contact graph definition is identical; Corollary 7 holds verbatim with $2d$-adjacency replacing 4-adjacency.

Applications include 3D voxel assembly, volumetric data compression, and higher-dimensional tensor compression.

A.7. Existence vs. Search Reachability

Corollary 7 establishes *existence* of an optimal placement tree, but our GA searches only “greedily-completable” trees. A natural question: *can the optimal tree be “ungreedy”?* That is, might the optimal require a locally suboptimal edge to achieve minimum cost globally?

In principle, yes: the optimal tree might contain an edge (i, j, δ) dominated by (i, j, δ') with smaller local cost, yet δ enables a globally superior arrangement.

Two design choices mitigate this:

1. *Stochastic completion* randomly samples among equally-good candidates, exploring alternative attachment points.
2. *Crossover dominance*: Table 5 shows 95–97.5% of placements are inherited from parents via crossover, not constructed by greedy. The greedy-completion bias affects only a small fraction of solution structure.

Empirically, the GA matches ILP solutions on small instances, suggesting the greedy-completion subspace contains near-optimal solutions for typical inputs.

A.8. Population Diversity Analysis

Our greedy completion is *context-sensitive*: placement of each missing string depends on the current canvas state, which varies across offspring.

Table 5 confirms:

- Crossover dominates: 95–97.5% of placements come directly from crossover.
- Repair rate r_{repair} remains bounded at 3–13% even as instances scale.
- This indicates sustained population diversity rather than convergence to "super-individuals."

The greedy operator acts as lightweight boundary repair, filling 2–5% of solution structure. This is analogous to mutation: essential for feasibility and diversity, but not the primary driver of solution quality.

B. ILP Formulation Details

This appendix provides the complete mathematical formulation of the mixed-integer linear program used for exact verification of small instances. The main text (Section 4.1) provides an overview; here we present the full technical details.

B.1. Notation and Grid Setup

We reuse the notation from Section 3. We have a finite set of 2D strings $\mathcal{T} = \{T_1, \dots, T_n\}$, each represented by a finite set of local cells $C_i \subset \mathbb{Z}^2$ and a symbol function $T_i : C_i \to \Sigma$. For each i we denote the local bounding box of T_i by

$$x_i^{\min} = \min_{(u,v)\in C_i} u, \qquad x_i^{\max} = \max_{(u,v)\in C_i} u,$$
$$y_i^{\min} = \min_{(u,v)\in C_i} v, \qquad y_i^{\max} = \max_{(u,v)\in C_i} v,$$

and its width and height by

$$w_i = x_i^{\max} - x_i^{\min} + 1, \qquad h_i = y_i^{\max} - y_i^{\min} + 1.$$

We embed all strings into a common rectangular grid $[0, W_g] \times [0, H_g] \subset \mathbb{Z}^2$. To guarantee that the ILP is a *true exact solver*, the grid must be large enough to accommodate *any* possible optimal arrangement, including those with extreme aspect ratios.

Remark 14 (Grid bounds and exactness)**.** A naïve approach would set $W_g = \sum_i w_i$ and $H_g = \sum_i h_i$ (the maximum possible dimensions if all strings are placed in a line with no overlap). This guarantees global optimality but creates an enormous grid.

A tempting optimization is to use greedy bounds: run a heuristic to obtain dimensions $W_{\text{greedy}} \times H_{\text{greedy}}$, then set $W_g := W_{\text{greedy}}$ and $H_g := H_{\text{greedy}}$. However, this is *not* sound for all objectives:

- *For 2D-SSP*$_{\text{area}}$*:* An optimal solution may have a different aspect ratio than the greedy solution. For example, if greedy yields a 10×10 placement (area 100), the optimal might be 2×40 (area 80). If $W_g = H_g = 10$, the ILP would exclude the 2×40 solution since $40 > 10$.
- *For 2D-SSP*$_{\text{bal}}$*:* The greedy bound *is* sound. If greedy achieves $\max\{W_{\text{greedy}}, H_{\text{greedy}}\} = L$, any optimal solution has $\max\{W^*, H^*\} \leq L$, so both dimensions are bounded by L.

To ensure global exactness for both objectives, we use:

$$W_g := \min\Big(\sum_i w_i,\ \mathrm{cost}^{\mathrm{greedy}}_{\mathrm{area}}\Big), \qquad H_g := \min\Big(\sum_i h_i,\ \mathrm{cost}^{\mathrm{greedy}}_{\mathrm{area}}\Big).$$

This exploits the fact that if the greedy area is A_{greedy}, any optimal area satisfies $W^* \cdot H^* \leq A_{\mathrm{greedy}}$, so $W^* \leq A_{\mathrm{greedy}}$ (since $H^* \geq 1$). The grid remains bounded by the greedy area rather than the sum of all dimensions, while accommodating all aspect ratios.

For small verification instances, this grid size is manageable. For larger instances, the ILP becomes intractable regardless of grid bounds.

B.2. Symmetry Breaking and Candidate Origins

To eliminate redundant symmetric solutions arising from translation invariance, we apply *symmetry breaking*: we fix the first string T_1 at the origin by restricting its set of allowed origins to $\mathcal{O}_1 := \{(0,0)\}$. This constraint removes all translated copies of any given solution from the search space without affecting optimality.

For each string $i \geq 2$ we define a finite set of allowed *origins* $\mathcal{O}_i \subset \mathbb{Z}^2$ such that translating T_i by o keeps all its local cells inside the global grid:

$$\mathcal{O}_i := \Big\{ o = (x,y) \in \mathbb{Z}^2 \,\Big|\, (x+u, y+v) \in [0, W_g] \times [0, H_g] \text{ for all } (u,v) \in C_i \Big\}. \tag{1}$$

Recall that $\mathcal{O}_1 = \{(0,0)\}$ by the symmetry-breaking constraint. When T_i is placed at an origin $o = (x,y)$, each local cell $(u,v) \in C_i$ is mapped to global coordinates $(x+u, y+v)$.

B.3. Conflict Precomputation

Two candidate placements (i,o) and (j,o') are incompatible if they assign different symbols to the same global coordinate. Formally, we precompute a Boolean conflict indicator

$$\kappa_{ijoo'} = \begin{cases} 1, & \text{if there exist } (u,v) \in C_i,\ (u',v') \in C_j \\ & \text{with } (x+u, y+v) = (x'+u', y'+v') \\ & \text{and } T_i(u,v) \neq T_j(u',v'), \\ & \text{for } o = (x,y),\ o' = (x',y'), \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

for all $i < j$, $o \in \mathcal{O}_i$, and $o' \in \mathcal{O}_j$. This preprocessing reduces symbol consistency to simple pairwise constraints in the ILP.

B.4. Decision Variables

The model uses the following variables.

- For each string $i \in \{1, \dots, n\}$ and each origin $o \in \mathcal{O}_i$:

  $b_{io} \in \{0,1\}$ (1 if T_i is placed at origin o, 0 otherwise).

- Integer coordinates of the global bounding box:

  $X_{\min}, X_{\max}, Y_{\min}, Y_{\max} \in \mathbb{Z}$,

  which represent the minimum and maximum global row/column indices among all occupied cells.

- The width and height of the bounding box:

  $W, H \in \mathbb{Z}_{\geq 0}$.

- A maximum side variable (for the balanced-area objective):

  $$L \in \mathbb{Z}_{\geq 0},$$

  representing the maximum of width and height.

- An area variable (for the area objective):

  $$A \in \mathbb{Z}_{\geq 0},$$

  used to model or approximate the bounding-box area $W \cdot H$.

In the implementation we bound these variables by a constant

$$M \;:=\; \max\{W_g, H_g\} + \max_i \max\{w_i, h_i\},$$

so that $X_{\min}, X_{\max}, Y_{\min}, Y_{\max} \in [-M, M]$ and $W, H, L \in [0, M]$, $A \in [0, M^2]$.

Remark 15 (Big-M calibration)**.** The choice of M involves a trade-off. If M is too small, valid placements may be incorrectly excluded; if M is excessively large, the LP relaxation becomes weak (the big-M constraints provide little tightening when $b_{io} = 0$), leading to slow branch-and-bound convergence.

Our choice $M = \max\{W_g, H_g\} + \max_i \max\{w_i, h_i\}$ is valid given the grid bounds from Remark 14: since the grid accommodates all optimal solutions, no coordinate can exceed $\max\{W_g, H_g\}$ plus the maximum string dimension. The bound is instance-adaptive and typically much smaller than a naïve bound like $n \cdot \max_i \max\{w_i, h_i\}$, improving LP relaxation quality.

B.5. Common Constraints

Both objective variants share the following groups of constraints.

(i) Exactly one origin per string. Each 2D string must be placed at exactly one origin:

$$\sum_{o \in \mathcal{O}_i} b_{io} \;=\; 1 \qquad \forall i \in \{1, \dots, n\}. \tag{3}$$

(ii) No symbol conflicts. If two candidate placements (i, o) and (j, o') conflict, they cannot be chosen simultaneously:

$$b_{io} + b_{jo'} \;\leq\; 1 \quad \forall\, i < j,\ \forall\, o \in \mathcal{O}_i,\ \forall\, o' \in \mathcal{O}_j \text{ with } \kappa_{ijoo'} = 1. \tag{4}$$

(iii) Bounding box must contain all placed strings. Let $o = (x, y) \in \mathcal{O}_i$ be a candidate origin for T_i. If $b_{io} = 1$, then the global footprint of T_i is

$$[x + x_i^{\min},\ x + x_i^{\max}] \times [y + y_i^{\min},\ y + y_i^{\max}],$$

and this rectangle must lie inside $[X_{\min}, X_{\max}] \times [Y_{\min}, Y_{\max}]$. We encode these implications with big-M constraints:

$$X_{\min} \;\leq\; x + x_i^{\min} + M(1 - b_{io}), \tag{5}$$
$$X_{\max} \;\geq\; x + x_i^{\max} - M(1 - b_{io}), \tag{6}$$
$$Y_{\min} \;\leq\; y + y_i^{\min} + M(1 - b_{io}), \tag{7}$$
$$Y_{\max} \;\geq\; y + y_i^{\max} - M(1 - b_{io}), \tag{8}$$

for all i and all $o = (x, y) \in \mathcal{O}_i$. When $b_{io} = 1$ these reduce to the desired inequalities $X_{\min} \leq x + x_i^{\min}$, $X_{\max} \geq x + x_i^{\max}$, etc., and when $b_{io} = 0$ they are relaxed by the big-M terms.

(iv) Definition of width and height. The bounding box dimensions are defined by

$$W = X_{\max} - X_{\min} + 1, \qquad H = Y_{\max} - Y_{\min} + 1. \tag{9}$$

In addition, W and H must be at least as large as the widest and tallest individual 2D string:

$$W \;\geq\; \max_i w_i, \qquad H \;\geq\; \max_i h_i. \tag{10}$$

B.6. Balanced-Area Objective

The balanced-area variant minimises the maximum side length $\max\{W, H\}$. We link L to W and H via

$$L \geq W, \qquad L \geq H. \tag{11}$$

At optimality, $L = \max\{W, H\}$.

The *balanced-area* ILP is

$$\begin{aligned} \min \quad & L \\ \text{s.t.} \quad & (3)-(10),\ (11), \\ & b_{io} \in \{0,1\}\ \forall i, o, \quad X_{\min}, X_{\max}, Y_{\min}, Y_{\max}, W, H, L \in \mathbb{Z}. \end{aligned} \tag{12}$$

B.7. Area Objective

The area-based variant uses the product $W \cdot H$ as its cost. Conceptually we want

$$A = W \cdot H. \tag{13}$$

Remark 16 (Linearization of the bilinear term)**.** The product $W \cdot H$ makes the area objective inherently *non-convex*. A direct formulation would yield a Mixed-Integer *Nonlinear* Program (MINLP), which is significantly harder to solve than a Mixed-Integer *Linear* Program (MILP).

We linearize this bilinear term using the *McCormick envelope*, the tightest convex relaxation of the product over a box. This is a standard technique in global optimization (McCormick, 1976). The McCormick envelope provides the *convex hull* of points $(W, H, W \cdot H)$ when W and H are continuous, but for integer variables, it is only a relaxation: intermediate points satisfying the envelope constraints may not correspond to integer solutions with $A = W \cdot H$.

For our small verification instances, this relaxation is tight in practice: the branch-and-bound solver finds integer solutions where the McCormick constraints are binding. Alternative approaches for tighter formulations include:

- *Logarithmic discretization:* Introduce binary variables for each bit of W and H, yielding $O(\log W_{\max} \cdot \log H_{\max})$ auxiliary variables and an exact linearization.
- *SOS2 piecewise-linear approximation:* Approximate $W \cdot H$ using special ordered sets of type 2, trading exactness for tighter LP relaxations.
- *Surrogate minimization:* Since we minimize area, we could minimize $(W + H)$ as a surrogate (perimeter), which is linear. However, this changes the objective and may yield suboptimal area solutions.

We chose McCormick for simplicity, as it sufficed for our small test cases. For larger instances requiring tighter relaxations, logarithmic discretization would be preferred.

In contrast, the balanced objective $\max\{W, H\}$ is trivially linearized via auxiliary variable $L \geq W$, $L \geq H$, no relaxation is needed.

To remain within a MILP framework we linearize this product over a known box $W \in [W_{\min}, W_{\max}]$, $H \in [H_{\min}, H_{\max}]$. We take

$$W_{\min} := \max_i w_i, \quad H_{\min} := \max_i h_i, \quad W_{\max}, H_{\max} \leq M,$$

and impose the standard McCormick envelope

$$\begin{aligned} A &\geq W_{\min} H + H_{\min} W - W_{\min} H_{\min}, && (14) \\ A &\geq W_{\max} H + H_{\max} W - W_{\max} H_{\max}, && (15) \\ A &\leq W_{\min} H + H_{\max} W - W_{\min} H_{\max}, && (16) \\ A &\leq W_{\max} H + H_{\min} W - W_{\max} H_{\min}. && (17) \end{aligned}$$

These constraints define the convex hull of all triples (W, H, A) with $A = W \cdot H$ and (W, H) in the given box. On small instances one can further tighten this relaxation by introducing a discrete piecewise-linear approximation of $W \cdot H$, but the simple McCormick envelope above already proved sufficient for our experiments.

The *bounding area* ILP is

$$\begin{aligned} \min \quad & A \qquad (18)\\ \text{s.t.} \quad & (3) - (10),\ (14) - (17),\\ & b_{io} \in \{0,1\}\ \forall i, o, \quad X_{\min}, X_{\max}, Y_{\min}, Y_{\max}, W, H, A \in \mathbb{Z}. \end{aligned}$$

The objective value $A^\star$ coincides with the area of the minimal axis-aligned bounding rectangle $B(p)$ enclosing all occupied cells in the induced placement p.

This formulation is conceptually straightforward: each b_{io} encodes a specific choice of origin for string T_i; the conflict constraints (4) enforce symbol consistency; the big-M constraints (5)–(8) define a global bounding box that contains all chosen placements; and depending on the variant, either (11) or (14)–(17) expresses the chosen cost. The greedy-based grid bounds and symmetry breaking significantly reduce the search space, but the number of variables and conflict constraints still grows quickly with the number of strings.

References

Bennell, J.A., Oliveira, J.F., 2009. A tutorial in irregular shape packing problems. Journal of the Operational Research Society 60, S93–S105.

Blum, A., Jiang, T., Li, M., Tromp, J., Yannakakis, M., 1994. Linear approximation of shortest superstrings. Journal of the ACM 41, 630–647.

Cazaux, B., Rivals, E., 2018. Hierarchical overlap graph. Information Processing Letters 136, 78–84.

Chang, Y.C., Chang, Y.W., Wu, G.M., Wu, S.W., 2000. B*-trees: a new representation for non-slicing floorplans, in: Proceedings of the 37th Design Automation Conference, pp. 458–463.

Charalampopoulos, P., Pissis, S.P., Radoszewski, J., Waleń, T., Zuba, W., 2021. Computing covers of 2d strings, in: 32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021), Schloss Dagstuhl–Leibniz-Zentrum für Informatik. pp. 12:1–12:20.

De Jong, K.A., 1975. An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. thesis. University of Michigan.

Efros, A.A., Freeman, W.T., 2001. Image quilting for texture synthesis and transfer, in: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp. 341–346.

Gallant, J., Maier, D., Storer, J.A., 1980. On finding minimal length superstrings. Journal of Computer and System Sciences 20, 50–58.

Gilmore, P.C., Gomory, R.E., 1965. Multistage cutting stock problems of two and more dimensions. Operations Research 13, 94–120.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley.

Golomb, S.W., 1994. Polyominoes: Puzzles, Patterns, Problems, and Packings. 2nd ed., Princeton University Press.

Kaplan, H., Shafrir, N., 2005. The greedy algorithm for shortest superstrings. Information Processing Letters 93, 13–17.

Kwatra, V., Schödl, A., Essa, I., Turk, G., Bobick, A., 2003. Graphcut textures: image and video synthesis using graph cuts. ACM Transactions on Graphics (ToG) 22, 277–286.

Leung, J.Y.T., Tam, T.W., Wong, C.S., Young, G.H., Chin, F.Y.L., 1990. Packing squares into a square. Journal of Parallel and Distributed Computing 10, 271–275.

Lodi, A., Martello, S., Monaci, M., 2002. Two-dimensional packing problems: A survey. European Journal of Operational Research 141, 241–252.

McCormick, G.P., 1976. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. Mathematical Programming 10, 147–175.

Merz, P., Freisleben, B., 2000. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. IEEE Transactions on Evolutionary Computation 4, 337–352.

Moscato, P., 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826. Caltech Concurrent Computation Program.

Mucha, M., 2013. Lyndon words and short superstrings, in: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM. pp. 958–972.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996a. Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 1518–1524.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1996b. Vlsi module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 1518–1524.

Sholomon, D., David, O.E., Netanyahu, N.S., 2013. A genetic algorithm-based solver for very large jigsaw puzzles. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition , 1767–1774.

Winfree, E., Liu, F., Wenzler, L.A., Seeman, N.C., 1998. Algorithmic self-assembly of dna. Nature 394, 539–544.

Wolsey, L.A., 1998. Integer Programming. Wiley-Interscience.

Yehezkeally, Y., Schwartz, M., 2025. Constructions of covering sequences and 2d-sequences. Designs, Codes and Cryptography 93, 1–25.