A Branch-And-Bound Algorithm for the Traveling Salesman Problem with Difficult Neighborhoods

Sándor P. Fekete
Department of Computer Science, TU Braunschweig, Germany
L3S Research Center, Hannover, Germany

Rouven Kniep
Department of Computer Science, TU Braunschweig, Germany

Dominik Krupke
Department of Computer Science, TU Braunschweig, Germany

Michael Perk
Department of Computer Science, TU Braunschweig, Germany

Abstract

The Traveling Salesman Problem with Neighborhoods (TSPN) generalizes the classical Traveling Salesman Problem (TSP) by requiring a tour to visit a set of polygonal regions rather than fixed points, a natural goal that arises in various applications. While the geometric TSP allows arbitrarily close approximation and provably optimal solutions for benchmark instances of significant size, the TSPN is considerably more challenging, both in theory (due to APX-hardness) and practice, for which only benchmark instances up to 16 regions have been solved to optimality.

Here we present a branch-and-bound algorithm that combines a spectrum of geometry-based filters (for reducing the number of considered sequences) with Second-Order Cone Programs (SOCP) (for computing optimal tours for a given permutation of neighborhoods). This allows us to solve larger polygonal TSPN instances than before to within an optimality tolerance of 0.1%; moreover, while previous work (both in theory and practice) relied on relatively benign neighborhoods, we can handle non-convex, non-simple neighborhoods of different sizes. In experiments on 490 benchmark instances with up to 50 polygons each, our method achieves a 99.6 % optimality rate within 300 s, with the remaining two instances solved within 595 s. For 68 larger instances of size $n = 60$, our method still allows solving 86.8 % of instances to optimality within 900 s, leaving only 3 of the instances with optimality gaps above 3 %, with the maximum being 5.53 %.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Discrete optimization; General and reference → Experimentation

Keywords and phrases Geometric optimization, geometric covering, TSP with neighborhoods, exact algorithms, algorithm engineering

Digital Object Identifier 10.4230/LIPIcs.SoCG.2026.46

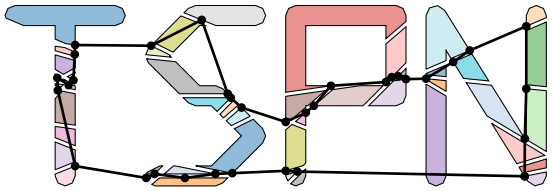
Supplementary Material *Software*: `https://doi.org/10.5281/zenodo.19202809`
Software: `https://github.com/tubs-alg/TSPN-SoCG-2026`
archived at `swh:1:dir:798ae47a3445f53c64575fac631a25a1921a137b`

Funding Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) as part of project *Computational Geometry: Solving Hard Optimization Problems* (CG:SHOP) – 444569951.

Acknowledgements We thank Barak Ugav for his contributions to an earlier version of the branch-and-bound implementation for the CE-TSP, on which parts of the current code are based.

1 Introduction

The Traveling Salesman Problem (TSP) asks for the shortest route that visits a given set of points. Central to many applications, it has also become *the* prototype of a provably difficult, NP-hard optimization problem. A natural geometric generalization of the TSP is the Traveling Salesman Problem with Neighborhoods (TSPN), which asks for a shortest roundtrip that visits each of a given family $P_1, \ldots, P_n$ of regions in the plane; see Figure 1 for an example. This reflects scenarios in which the locations to be visited are not point-sized, but subject to some flexibiliy. Moreover, the TSPN combines the search for optimal touring with aspects of set cover: among others, discrete variants of both the Lawn Mowing Problem (LMP) and the Art Gallery Problem (AGP) can be seen as special cases of the TSPN.



50 polygons, solved in 0.45s

Figure 1 Example TSPN instance with a feasible solution.

Even though both the TSP and the TSPN are NP-hard (making them comparable in theoretical complexity), it can be argued that the TSPN is considerably more challenging. This is clearly reflected in theory: While the geometric TSP allows polynomial-time approximation schemes (Arora [6], Mitchell [37]), APX-hardness of the TSPN was established by de Berg et al. [14]. Moreover, a wide range of sophisticated approximation algorithms rely on assumptions on the shape of the neighborhoods, such as special shapes (e.g., line segments or disks), congruency, convexity, or fatness; see *Related Work* for further details.

A similar gap can be observed on the practical side. The discrete nature of the TSP allows employing a large toolbox of advanced methods from combinatorial optimization and integer programming, enabling provably optimal solutions for point sets of considerable size (such as the 85 900-city instance solved by [3, 12]). On the other hand, the TSPN requires considerable additional effort: even for a given order in which the neighborhoods must be visited, computing a shortest tour is non-trivial. As a consequence, the state of the art until now are provably optimal solutions for benchmark instances up to just 16 regions [29]; see Figure 2 (Left) for an example instance. While hardware and solvers have improved significantly since its publication 13 years ago, solving the problem as a MIQCP with Gurobi on modern hardware did not significantly change the results (see Section 4.6 for details).

Another key difference between discrete TSP in a graph and continuous TSPN in geometry arises from accuracy and error bounds: For a given graph with integer edge weights, verifying that a given bound is met is straightforward; however, in a geometric setting with algebraic complications arising from Euclidean distances and floating-point arithmetic, that is far from clear. The well-known TSPLIB instances [40] use distances rounded to the nearest integers, while even LP solvers such as CPLEX [33] and Gurobi [31] operate with an *optimality tolerance*: The final result is within a given small tolerance of the provable optimum.

Contributions. We present a branch-and-bound algorithm that combines the search for an optimal order for a given set of neighborhoods with Second-Order Cone Programs (SOCP) for computing the best tour for any given sequence of neighborhoods. This significantly

expands the range of instances that can be solved to within an optimality tolerance of 0.1%; note that even the default tolerance of Gurobi is already 0.1‰. Across 490 diverse benchmark instances with up to 50 polygons each, 99.6 % are solved to optimality within a time limit of 300 s, with the remaining two instances solved within 595 s. For 68 larger instances of size $n = 60$, our method still allows solving 86.8 % of instances within 900 s, leaving only 3 of the instances with optimality gaps above 3 %, with the maximum being 5.53 %. Note that $16! \approx 2 \cdot 10^{13}$, while $50! \approx 3 \cdot 10^{64}$ and $60! \approx 8 \cdot 10^{81}$. What is more, we achieve this for non-convex neighborhoods of different sizes, including neighborhoods with holes and even disconnected regions. See Figure 2 for a comparison with the previous state of the art. We also evaluate the impact of various algorithmic design choices, along with geometry-based filters, providing perspectives for even larger instances.

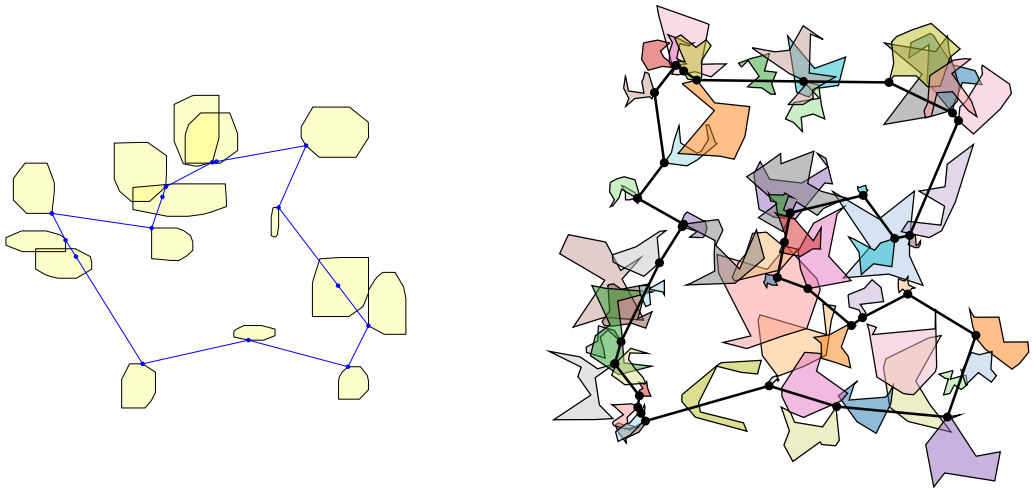


Figure 2 (Left) Previous state of the art [29]: optimal solution for $n = 15$ fat convex regions. (Right) This paper: optimal solution for $n = 60$ non-convex regions of various aspect ratios.

Related Work. A special case of TSPN is the Close-Enough TSP (CE-TSP), where each neighborhood is a circle, which is also related to the Lawn Mowing Problem [4, 5, 26, 27]. Branch-and-bound algorithms have been studied for CE-TSP [19, 13] (possibly with obstacles [15]); however, the TSPN allows arbitrary (including non-convex) neighborhoods, making it more general. A common approach to such problems is to discretize neighborhoods into representative points and solve a Generalized TSP (GTSP) [35], e.g. by a heuristic [32, 41] or an exact method [9, 16]. Advancements in machine learning have also led to unsupervised learning methods for CE-TSP [25, 24]. Many heuristics and approximation algorithms for the TSPN rely on assumptions (e.g., fatness [38, 23], disjoint neighborhoods [14], or comparable region diameters [21]) to manage its APX-hardness. Recently, Antoniadis et al. [2] developed an FPTAS for minimum-perimeter convex polygon intersecting a set of convex objects. Specialized approaches address specific settings such as aerial vehicle routing [34], and hybrid methods combine meta-heuristics with TSP solvers [44]. Non-convex Mixed Integer Nonlinear Programming (MINLP) formulations have been proposed for TSPN, including symmetric and asymmetric variants [29], with algorithmic improvements to reduce solution times. However, computational tests were limited to smaller or convex neighborhoods. Other work derives approximations and bounds for the metric TSPN using the MST with Neighborhoods [11].

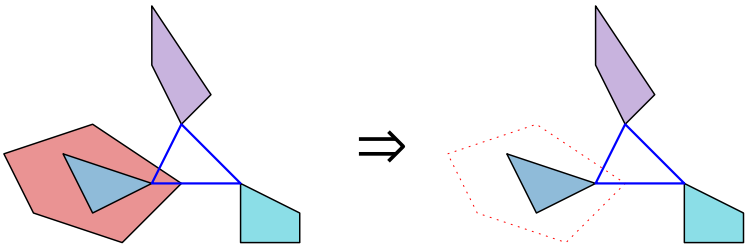


Figure 3 Example of *superset elimination*: a polygon fully contained in another.

Preliminaries. We are given a set of n polygons $\mathcal{P} = \{P_1, \ldots, P_n\}$ in the plane. Polygons may be non-convex and may contain holes. A *tour* is a closed curve in the plane that intersects each polygon $P_i \in \mathcal{P}$ at least once. Without loss of generality, we may assume that an optimal tour is a polygonal cycle visiting one *hitting point* $h(P_i) \in P_i$ for each polygon $P_i \in \mathcal{P}$, connected by straight-line segments. We represent a tour T by the ordered sequence of its hitting points, $(h(P_{\pi(1)}), \ldots, h(P_{\pi(n)}))$, where π is a permutation of $[n]$ and $h(P_{\pi(i)}) \in P_{\pi(i)}$ for all i. The length of T is the sum of the Euclidean distances between consecutive hitting points, including the distance between the last and the first hitting point. Overall, the goal is to find a feasible tour of minimum total length. By triangle inequality, optimal tours cannot self-intersect, so this is a simple polygon of minimum perimeter that intersects all $P_i \in \mathcal{P}$.

2 Preprocessing

Preprocessing is a crucial step in most practical optimization algorithms, as it can significantly reduce the search space and improve algorithmic efficiency. See the literature on MIP presolve [1] and SAT preprocessing [7] for more details on the extensive use of preprocessing in combinatorial optimization. We also investigated and implemented several preprocessing techniques for TSPN instances that either eliminate polygons entirely when any tour that visits the remaining polygons will necessarily cover them, simplify polygons by cutting off unnecessary parts or replacing them with simpler shapes, or add annotations that guide the branch-and-bound algorithm. We first describe basic geometric simplification rules in Section 2.1 and then discuss order annotations in Section 2.2.

2.1 Basic Simplification

We iteratively apply the following three simplification rules until no further reduction is possible. In practice, these rules eliminate nearly all holes and many large polygons.

▶ **Theorem 1** (Superset elimination). *Let $\mathcal{P}$ be a set of polygons in a TSPN instance. If a polygon $P \in \mathcal{P}$ is fully contained in another polygon $P' \in \mathcal{P}$ (see Figure 3), then P' can be safely removed without affecting the optimal tour.*

Proof. Any tour that visits P necessarily intersects P', since P' completely contains P. ◀

▶ **Theorem 2** (Hole removal). *Let $\mathcal{P}$ be a set of polygons for a TSPN instance. Suppose a polygon $P \in \mathcal{P}$ contains a hole h, and there exists a polygon $P' \in \mathcal{P}$ lying entirely outside h (see Figure 4). Then the hole h can be removed from P without affecting the optimal tour.*

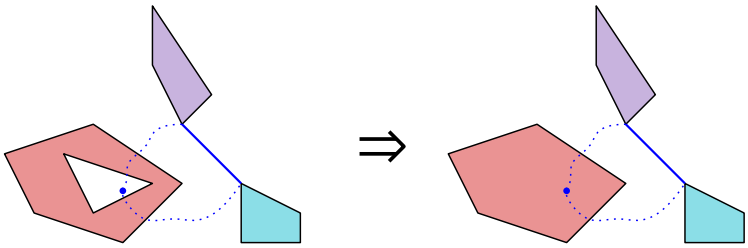


Figure 4 Example of *hole removal*: the hole can be discarded when another polygon lies outside of it. If we were to visit it, we would already intersect the polygon containing it.

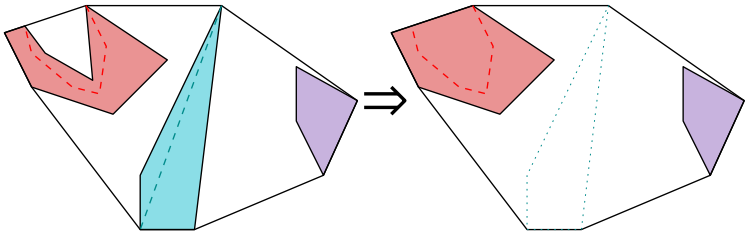


Figure 5 Example of *pocket removal*: of the red polygon, we can remove the pocket, and the blue polygon, we can remove entirely.

Proof. Any tour visiting P' must pass through P, since P' lies outside the hole. If the tour enters the hole h, it already intersects P on the way to or from P'. Thus coverage of h guarantees coverage of P, and the hole does not impose any additional constraint. ◀

▶ **Theorem 3** (Pocket removal). *Let B be a convex closed curve containing an optimal TSPN tour for an instance $\mathcal{P}$, e.g., the convex hull of $\mathcal{P}$. Suppose that a polygon $P \in \mathcal{P}$ contains a path with both endpoints on B, fully lying inside P. This path, together with the arc of B between its endpoints, forms a closed curve that encloses a region (the* pocket*). If at least one polygon of $\mathcal{P}$ lies fully outside the enclosed area, then the enclosed region may be added to P, i.e., the pocket can be removed, without affecting the optimal tour. Furthermore, if also the enclosed region fully contains a polygon from $\mathcal{P}$, then P itself can be safely removed.*

Proof. A tour entering the enclosed region must cross the enclosing path. As the tour must also visit polygons outside the enclosed region, it must cross this path again when leaving. Each crossing intersects P, so P is inevitably covered. If the enclosed region also contains polygons from $\mathcal{P}$, the tour must enter and leave the region, crossing the path at least twice. Thus, removing the pocket preserves feasibility, and P becomes redundant. ◀

Note that the second case of Theorem 3 is automatically handled by Theorem 1 once the pocket has been removed from the polygon, because the enclosed polygon will be fully contained in the modified polygon. If B is not the convex hull, we may first intersect all polygons with B. This ensures that pockets adjacent to B are also removed consistently and that the simplification rules behave exactly as in the convex-hull case.

2.2 Order Annotations

Our algorithm suffers from a large branching factor, as it considers every possible ordering of input polygons. Fortunately, geometric properties of optimal tours let us prune many insertion choices, in particular for polygons that intersect the boundary of a tour-enclosing

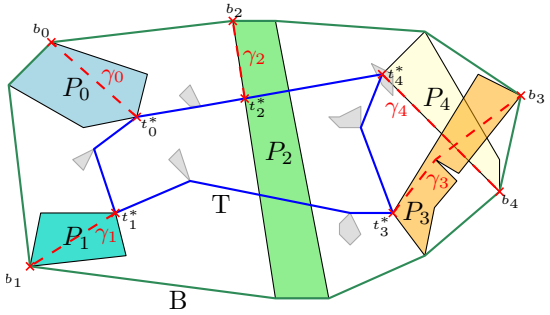


Figure 6 Illustration for b and t^* on B and T. Note that P_3 and P_4 overlap, so we can have $\mathcal{P}'_B = \{P_0, P_1, P_2, P_3\}$ or $\mathcal{P}'_B = \{P_0, P_1, P_2, P_4\}$ as maximal sets. While preprocessing would remove P_2 (via pocket removal and superset elimination, cf. Figure 5), we retain it here to emphasize that Theorem 5 holds independently of preprocessing.

hull B, such as the convex hull of all polygons $\mathcal{P}$. A key property is that an optimal tour must respect the order of the polygons along this hull. Although this is intuitive, complications arise when polygons overlap, because the order is then only partially defined. Let $\sigma_B : [|\mathcal{I}_B|] \to \mathcal{I}_B$ be a cyclic order on a subset $\mathcal{I}_B \subseteq [n]$ of indices. The relation $\circlearrowleft_{\sigma_B}(i,j,k)$ is satisfied if either index is not in $\mathcal{I}_B$ or polygons P_i, P_j, P_k are not pairwise disjoint, or if in the cyclic order of σ_B, index j lies between indices i and k following the fixed orientation of σ_B.

▶ **Definition 4.** *Let $\sigma_B : [|\mathcal{I}_B|] \to \mathcal{I}_B$ be a cyclic order on a subset $\mathcal{I}_B \subseteq [n]$ of indices, then $\circlearrowleft_{\sigma_B}(i,j,k)$ is defined as*

$$\circlearrowleft_{\sigma_B}(i,j,k) = \begin{cases} 1 & \text{if } \{i,j,k\} \nsubseteq \mathcal{I}_B \vee P_i \cap P_j \neq \emptyset \vee P_j \cap P_k \neq \emptyset \vee P_i \cap P_k \neq \emptyset \\ 1 & \text{if } (\sigma_B^{-1}(j) - \sigma_B^{-1}(i)) \bmod |\mathcal{I}_B| \leq (\sigma_B^{-1}(k) - \sigma_B^{-1}(i)) \bmod |\mathcal{I}_B| \\ 0 & \text{otherwise} \end{cases}.$$

▶ **Theorem 5.** *Let B be a simple closed convex curve that contains an optimal TSPN tour for a given instance. Let $\mathcal{I}_B \subseteq [n]$ and $\mathcal{P}_B = \{P_j : j \in \mathcal{I}_B\}$ be the set of polygons intersecting B, and for each polygon fix the reference points $\{b_j \in P_j \cap B \mid j \in \mathcal{I}_B\}$ which define a cyclic order σ_B on $\mathcal{I}_B$ that corresponds to the order of the reference points along B. Then there exists an optimal tour T with visiting order π such that, for every $a < b < c \in [n]$ the order of their hitting points coincides with the order along B, i.e. $\forall a < b < c \in [n] : \circlearrowleft_{\sigma_B}(\pi(a), \pi(b), \pi(c))$.*

Proof. Let T be an optimal tour contained in a simple closed convex curve B. Both T and B are counterclockwise oriented. For each $P_j \in \mathcal{P}_B$ we fix a reference point $b_j \in P_j \cap B$ which is used to define the order σ_B on $\mathcal{I}_B$. As T is feasible it must visit all polygons; for all $j \in \mathcal{I}_B$ we choose the hitting point $t_j^* \in P_j \cap T$ minimizing the geodesic distance in P_j to b_j. Let γ_j be a shortest path in P_j from b_j to t_j^* and denote γ_j^{-1} its reversal. By convexity of B, γ_j meets B only at b_j; by minimality, γ_j meets T only at t_j^*; and since $\gamma_j \subseteq P_j$, it can intersect $\gamma_{j'}$ only if $P_j \cap P_{j'} \neq \emptyset$, see Figure 6.

Let $\phi : [m] \to \mathcal{I}_B$ be an injection with $m \leq n$ that defines an ordered subsequence (induced by σ_B) corresponding to pairwise disjoint polygons, i.e., $\forall i \neq j \in [m] : P_{\phi(i)} \cap P_{\phi(j)} = \emptyset$ and $\forall i < j < k \in [m] : \circlearrowleft_{\sigma_B}(\phi(i), \phi(j), \phi(k))$. We write $[\cdot,\cdot]_B$ for the arc of B, and $[\cdot,\cdot]_T$ for the subpath of T between two points following B's and T's orientation respectively. For each $i \in [m]$ we define the closed curve Γ_i between $b_{\phi(i)}$ and $b_{\phi(i+1)}$ (indices should be seen modulo m) as $\Gamma_i = [b_{\phi(i)}, b_{\phi(i+1)}]_B \cup \gamma_{\phi(i+1)} \cup [t^*_{\phi(i+1)}, t^*_{\phi(i)}]_T^{-1} \cup \gamma_{\phi(i)}^{-1}$, which bounds a simply connected region R_i. Thus, each region boundary is explicitly composed of: (i) the

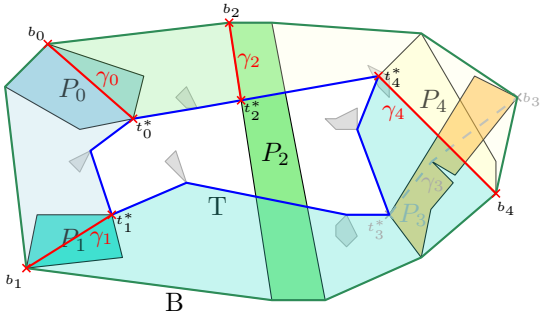


Figure 7 The paths of $\mathcal{P}'_B = \{P_0, P_1, P_2, P_4\}$ tile the area between B and T, forming a cycle of regions and boundaries. We can also see that it is important to select a pairwise disjoint set of polygons, as the crossing paths of P_3 and P_4 would break the argument otherwise. While one can find an alternative b_3 to avoid this, we need to remember that we actually do not know T in advance but provide an argument that works for any T.

arc of B, (ii) the two shortest paths $\gamma_{\phi(i)}, \gamma_{\phi(i+1)}$, and (iii) the corresponding subpath of T. These regions are interior-disjoint and tile exactly the annular portion between B and T, see Figure 7. Each shortest path $\gamma_{\phi(i)}$ with $i \in [m]$ has two incident regions, one on each side which means that the regions form a cycle around T.

Tracing T along the subpaths $[t^*_{\phi(i)}, t^*_{\phi(i+1)}]_T$ implies that T is a concatenation of hitting points that respect $\circlearrowleft_{\sigma_B}(\cdot,\cdot,\cdot)$ for any valid injection ϕ which implies that $\circlearrowleft_{\sigma_B}(\cdot,\cdot,\cdot)$ holds for all triples of pairwise disjoint polygons in $\mathcal{P}_B$. Degenerate contacts between T and B (or degenerate γ_j) can be removed by an infinitesimal perturbation preserving optimality and incidences. Hence, for any $\circlearrowleft_{\sigma_B}$, there exists an optimal tour T for which every triple $P_{\pi(a)}, P_{\pi(b)}, P_{\pi(c)} \in \mathcal{P}_B$ with $a < b < c$ must satisfy $\circlearrowleft_{\sigma_B}(\pi(a), \pi(b), \pi(c))$. ◀

During preprocessing, we annotate each polygon $P_i \in \mathcal{P}_B$ intersecting the convex hull B with its position in the cyclic order σ_B, together with the indices of all polygons $P_j \in \mathcal{P}_B$ that overlap with P_i. This information enables us to select a maximal subset of pairwise disjoint polygons $\mathcal{P}'_B \subseteq \mathcal{P}_B$, which induces a total order that any optimal tour can respect.

3 Branch-and-Bound Algorithm

Our branch-and-bound algorithm (Algorithm 1) explores the space of polygon orderings by incrementally building sequences: starting from a small initial subsequence that is guaranteed to be extendable to an optimal solution (Section 3.2), it repeatedly selects a polygon not yet in the sequence and creates a child node for each possible insertion position, forming a search tree (Section 3.3). For each candidate sequence, a Second-Order Cone Program (SOCP) computes the optimal tour for that ordering (Section 3.1). If this tour visits all polygons, it is a feasible solution (see Figure 8); otherwise, the SOCP provides a lower bound on any tour extending this partial sequence – if it exceeds the best known solution, no completion can improve upon it and the entire subtree is pruned. This builds on the framework of Coutinho et al. [13] for the CE-TSP, which we improve and extend to handle the TSPN.

Throughout the search, the algorithm keeps track of the incumbent (best known feasible) solution and the node with the lowest lower bound, which together define the current optimality gap. As the search tree can grow large, the order in which nodes are evaluated can have significant impact on performance: finding good feasible solutions early strengthens

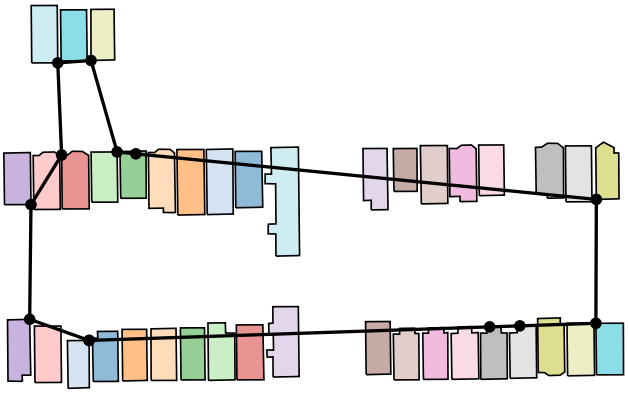


Figure 8 Illustration of implicit coverage in the TSPN. Many polygons will be covered naturally when connecting other polygons, without needing to explicitly model them in the optimization.

pruning, while evaluating nodes with low lower bounds tightens the optimality gap. A search strategy balances these objectives (Section 3.4). Search is terminated if this gap is smaller than a desired threshold, or if all nodes are explored.

3.1 Touring a Sequence of Polygons

By Dror et al. [20], the Touring Polygons Problem (TPP) that asks for the shortest tour visiting a given sequence of polygons can be solved in polynomial time for convex polygons when the visitation order and a start and end point are fixed. The runtime of the algorithm was later improved by Tan and Jiang [42] to $O(k^2m)$ for k convex regions with a total number of m vertices. If the start and end points are not fixed, we can still solve it in polynomial time using a Second-Order Cone Program (SOCP), similar to Coutinho et al. [13].

▶ **Theorem 6.** *Let $P_0, \dots, P_{n-1} \subset \mathbb{R}^2$ be convex polygons. Then the shortest tour visiting these polygons in order can be computed in polynomial time.*

Proof. A convex polygon P_i can be represented by linear constraints $S_i(x, y)$. We formulate the problem as a Second-Order Cone Program (SOCP), which can be solved in polynomial time via interior-point methods [8]. For each i, introduce a point $(x_i, y_i) \in \mathbb{R}^2$ constrained by $S_i(x_i, y_i)$, ensuring $(x_i, y_i) \in P_i$. For readability, all indices are assumed modulo n.

To encode the total tour length, let $d_i \geq 0$ be the distance between (x_i, y_i) and (x_{i+1}, y_{i+1}). We introduce auxiliary variables $\hat{x}_i, \hat{y}_i$ subject to $\hat{x}_i \geq x_i - x_{i+1}$, $\hat{x}_i \geq x_{i+1} - x_i$, $\hat{y}_i \geq y_i - y_{i+1}$, $\hat{y}_i \geq y_{i+1} - y_i$, and impose second-order cone constraints $d_i^2 \geq \hat{x}_i^2 + \hat{y}_i^2$.

All constraints are linear or second-order cone constraints, so the formulation is an SOCP of polynomial size. Therefore, the shortest tour can be computed in polynomial time. ◀

As we ultimately aim to solve TSPN instances with non-convex polygons (and potentially holes if preprocessing does not eliminate them), we must extend the SOCP approach. Dror et al. [20] showed that the problem becomes NP-hard for non-convex polygons, so no polynomial-time algorithm can be expected in general. We propose two strategies, both based on convex decompositions, which partition a non-convex polygon into convex components, see Figure 9.

The first approach is to replace the SOCP by a Mixed-Integer SOCP (MISOCP). For a non-convex polygon P_i', let $\mathcal{R}_i$ be its convex decomposition. Introduce a binary variable $r_c \in \mathbb{B}$ for each convex piece $c \in \mathcal{R}_i$ and enforce $\sum_{c \in \mathcal{R}_i} r_c = 1$. Let $S_i^c(x, y)$ be the linear constraints

Algorithm 1 Branch-and-Bound Algorithm.

Require: Set of polygons $\mathcal{P}$
Preprocess and simplify $\mathcal{P}$.
$\mathcal{Q} \leftarrow [\,]$ ▷ Queue of leaf nodes to explore
$v \leftarrow \textsc{getRootNode}(\mathcal{P})$ ▷ See Section 3.2
Compute optimal tour and lower bound on v via SOCP ▷ See Section 3.1
$\mathcal{Q}$.push(v)
$ub \leftarrow \infty$
while $\mathcal{Q} \neq \emptyset \wedge \min\{v'.lb \mid v' \in \mathcal{Q}\} < ub$ **do**
    $v \leftarrow \textsc{getNextNode}(\mathcal{Q})$ ▷ Search strategy, see Section 3.4
    Remove v from $\mathcal{Q}$
    if $v.lb \geq ub$ **then**
        continue
    end if
    if Solution in v covers all polygons in $\mathcal{P}$ **then**
        $ub \leftarrow v.lb$
    else
        for *child* $\in \textsc{Branch}(v)$ **do** ▷ Branching, see Section 3.3
            Compute optimal tour and lower bound on *child* via SOCP
            $\mathcal{Q}$.push(*child*)
        end for
    end if
end while
return Incumbent solution corresponding to ub, or $\perp$ if no solution was found.

defining ∂c. To ensure $(x, y) \in P'_i$, impose the implications $r_c = 1 \Rightarrow S^c_i(x, y) \quad \forall c \in \mathcal{R}_i$. These implications can be modeled using indicator constraints (when supported) or with Big-M formulations, where M can be bounded using the polygon's bounding box. Both indicator and Big-M constraints tend to produce weak relaxations for the MISOCP solver, which provides limited guidance to its internal branch-and-bound. To improve the relaxation, one may additionally include the convex-hull constraints $S_i(x, y)$ of P'_i, which are redundant but can strengthen the continuous relaxation.

The second approach retains the efficient SOCP formulation and incorporates convex decomposition only through branching. Instead of encoding the convex components by binary variables, the branch-and-bound tree explicitly chooses which convex part of a non-convex polygon the tour must hit. This extends the branching decision from the visitation order

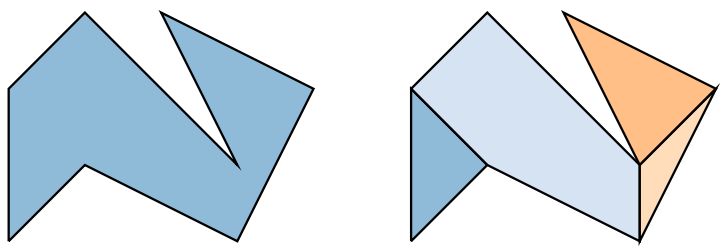


Figure 9 A non-convex polygon (left) and its decomposition in convex areas (right).

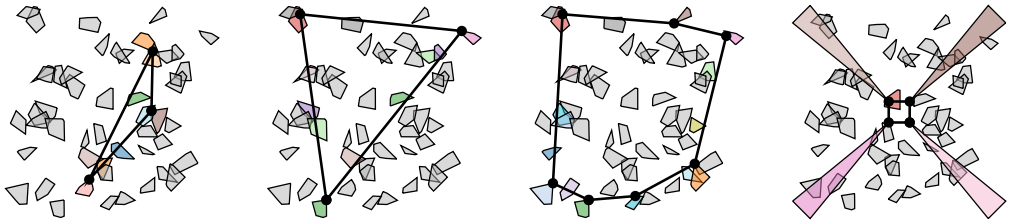


Figure 10 Root node strategies (left to right): Random, LEFP, CHR, and an example demonstrating poor CHR performance.

to the choice of the convex component, while avoiding indicator and Big-M constraints entirely. As a result, this approach avoids the weak relaxations that appear in the MISOCP formulation, although it may increase the size of the search tree.

For polygons without holes, a minimum convex decomposition can be computed in polynomial time [30, 10]. For polygons with holes, a decomposition is always possible, for example by triangulation, but finding a minimal decomposition is NP-hard [36]. In practice, our preprocessing eliminates all holes from the benchmark instances, so the NP-hard minimal decomposition is not needed. The theoretical caveat remains for degenerate instances where hole removal cannot apply, e.g., when every other polygon lies inside the hole.

Finally, any non-convex polygon can either weaken the relaxation in the MISOCP approach or introduce substantial branching overhead. However, in optimal tours we often observe that visitation points of spanning polygons, meaning those not implicitly covered by connections between other polygons, tend to lie on extreme points, that is, on the convex hull of the polygon. This motivates a lazy strategy in which every polygon is initially replaced by its convex hull, making all polygons appear convex in the relaxation. Only when a computed visitation point lies outside the true polygon do we enforce its full non-convex geometry.

3.2 Root Node

The root node's initial polygon sequence must be extendable to an optimal solution. Any sequence of up to three polygons trivially satisfies this condition. We evaluate four strategies: **Random** selects three polygons at random. **Longest Edge (LE)** selects the two polygons with largest pairwise distance. **Longest Edge+Farthest Polygon (LEFP)** extends LE by adding the polygon farthest from both. **Convex Hull (CHR)** selects polygons based on their convex hull positions.

Using the observations from Section 2.2, we can construct larger initial sequences: for any set of pairwise disjoint polygons intersecting the convex hull, σ_B defines a total order that an optimal solution can respect. Finding a cardinality-maximal set of such polygons is an instance of the NP-hard *maximum independent set* problem, but optimality is not required here; a greedy heuristic yields a sufficiently large set in practice. This preserves completeness while often producing significantly stronger initial relaxations. Although it is possible to construct artificial instances where this strategy produces arbitrarily weak relaxations, for example by placing polygons on the convex hull that protrude far inward and thereby induce an initial tour of negligible length, such instances appear to be rare in practice. Figure 10 illustrates three of the strategies, including an example where CHR performs poorly.

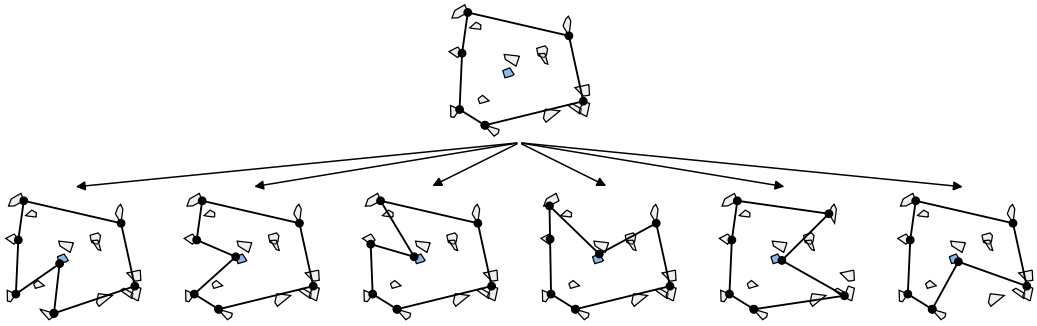


Figure 11 Branching on the insertion position of the selected farthest polygon.

3.3 Branching

When the (partial) tour in a node does not cover all polygons, we branch by selecting a missing polygon and creating a new branch for each possible insertion position. In our implementation, we select the polygon that lies farthest from the current tour, see Figure 11.

We consider three modeling strategies for handling non-convex polygons. **Eager indicator-based modeling (EIM)** models non-convex polygons explicitly using indicator variables in a MISOCP formulation. Because convex hulls often contain the extremal points that serve as optimal hitting points, we propose a lazy strategy that initially replaces each polygon by its convex hull and only models it as non-convex when necessary. Both of the following strategies employ this lazy approach. **Indicator-based modeling (IM)** uses the same indicator-variable formulation as EIM but only introduces it when the convex hull replacement is insufficient. **Decomposition branching (DB)** avoids indicator variables entirely by decomposing the polygon into convex parts and creating one branch per part, see Figure 12.

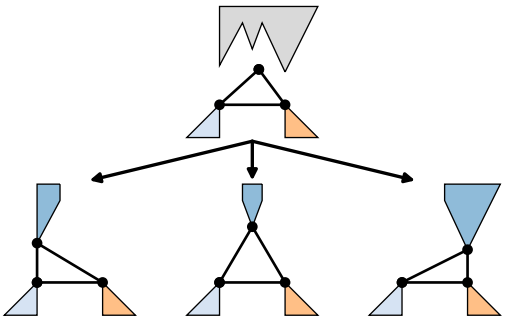


Figure 12 Branching on a non-convex polygon by decomposing it into convex pieces.

3.4 Search Strategies

We implement four strategies to select the next node for exploration during the search.

Best First Search (BFS) selects the node with the smallest lower bound, improving the global bound quickly. However, incomplete sequences tend to have shorter estimated lengths, so BFS often prioritizes them over full coverage. **Depth First Search (DFS)** continues

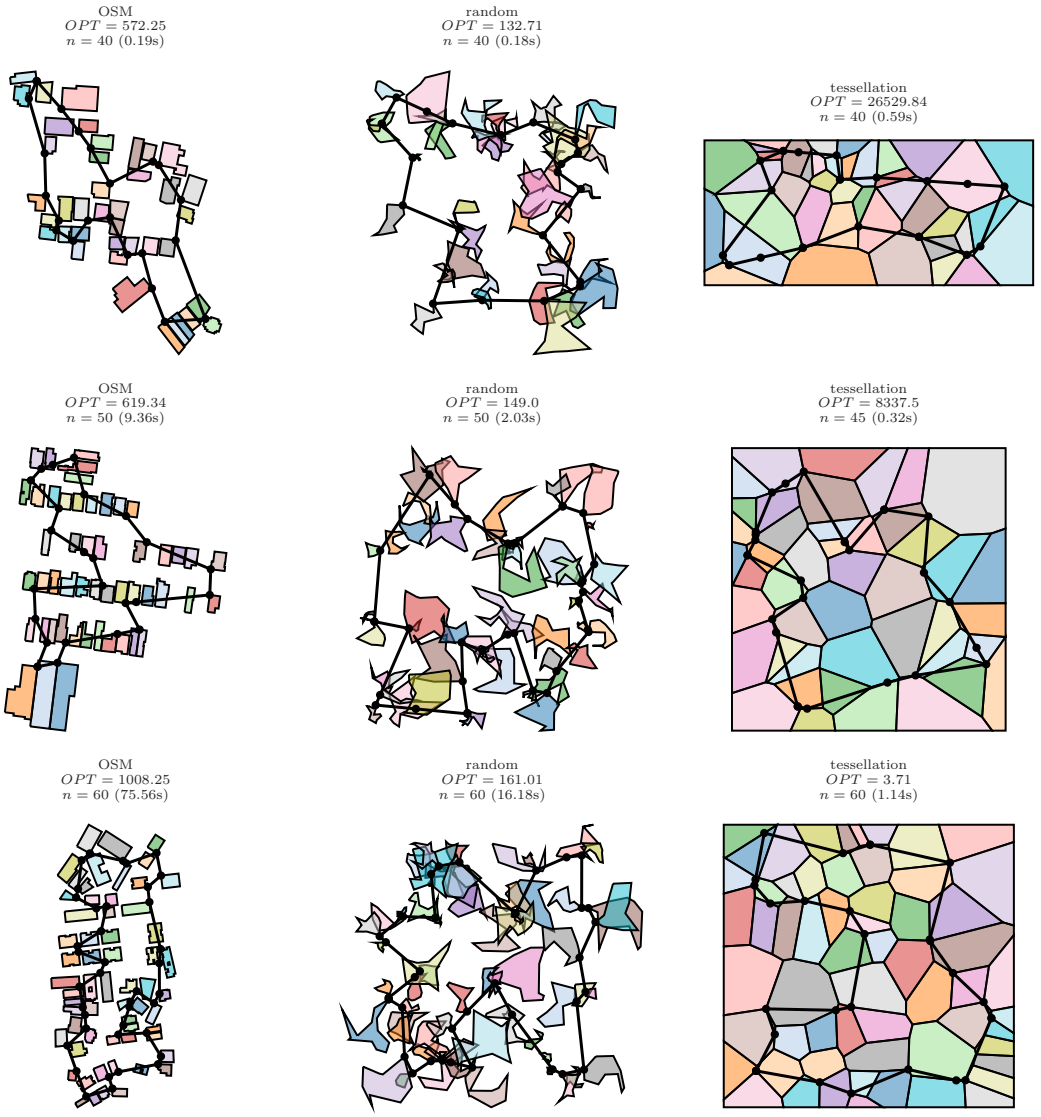


Figure 13 Example solutions within 0.1 % of optimality for instances of size $n \in \{40, 50, 60\}$.

exploring the most promising child, quickly reaching feasible solutions. Good incumbents strengthen pruning, though DFS largely ignores lower bounds. **DFS+BFS** behaves like DFS, but reorders the queue by lower bounds whenever a node is pruned or a new solution is found, combining rapid incumbent improvements with lower-bound tightening.

4 Empirical evaluation

Our algorithm is implemented in C++ and uses Gurobi 12.0 [31] to solve the SOCP and MISOCP formulations. Geometry operations rely on Boost.Geometry 1.83 [28] and CGAL 6.0.1 [43] with exact predicates and constructions. We compiled using g++ 13.3 and ran all experiments on an AMD Ryzen 7900 workstation with 96 GiB of DDR5-5200 RAM under Ubuntu 24.04. We use the term *optimality gap* to refer to the relative difference between the best known upper bound and the global lower bound, i.e., $^{ub}/_{lb} - 1$. The optimality tolerance is the maximum allowed optimality gap after which the algorithm can terminate.

4.1 Research questions

Our experimental evaluation aims to answer the following questions.

Q1 How does the choice of root node sequence affect runtime across instance types?

Q2 Which search strategy (DFS, BFS, or a combined DFS+BFS) offers the best trade-off between time-to-solution and bound improvement?

Q3 For non-convex polygons, does decomposition branching outperform indicator-based modeling in practice with respect to runtime and relaxation quality?

Q4 How does relaxing the optimality tolerance affect the performance and how does our approach scale with increasing instance sizes?

4.2 Experiment design

To answer our questions, we collected and generated a large set of 558 instances, consisting of instances from the following instance classes. In all cases, the coordinates of points in the instances are either integers or double precision floating-point numbers.

random We generated 20 random instances with $n \in \{5, 10, 15, 20, 30, 40, 50, 60\}$ random polygons with roughly 45 % concave points and between 10 and 15 edges. Polygons may overlap and may have holes. This resulted in 160 random instances.

tessellation Instances derived from well-known publicly available benchmark instance sets. To obtain polygonal regions, we applied a Voronoi tessellation to point sets from various sources. These include point sets previously used in the CG:SHOP challenges [18, 17], TSPLIB instances [40], and point sets from the Salzburg Database of Polygonal Inputs [22]. In total we generated 78 instances with $n \leq 60$ polygons.

OSM Instances derived from *Open Street Map* (OSM) data [39]. Our dataset covers building footprints from 20 major cities. We used a quadtree-like subdivision to achieve desired sizes of $n \in \{5, 10, 15, 20, 30, 40, 50, 60\}$ and then randomly sampled two instances from each city in our collection. This resulted in a total of 320 instances.

All instances were preprocessed with the techniques from Section 2, resulting in 360 (out of 558) altered instances with no holes. This took 0.01 s on average, with a maximum of 0.06 s. The resulting instances show reduced complexity. In particular, the number of convex pieces (i.e., the number of convex components in an optimal convex decomposition) decreased by an average of 2.9 per instance. Example instances and their optimal solutions can be found in Figure 13. Unless stated otherwise, all experiments use the full set of 558 instances.

4.3 RQ1: Root node selection

The choice of the root node sequence can significantly influence the search performance. A larger root sequence yields a tighter initial lower bound and fixes the relative order of more polygons. However, it may include polygons that would otherwise be implicitly covered by the tour (see Figure 8), unnecessarily enlarging the explicit sequence and increasing both the branching factor and the SOCP size at every node. We compared four strategies for selecting the root node sequence with a time limit of 60 s and 1 % optimality tolerance. CHR selects polygons based on the convex hull order, LE selects two polygons that are farthest apart, LEFP extends LE by adding a third polygon that maximizes the distance to the first two, and Random selects a random sequence, see Section 3.2 for details. Figure 14 shows the number of instances and the optimality gaps after 60 s. Overall, using a CHR and LEFP strategy appear to be best across our benchmarks with solutions within 1 % tolerance found

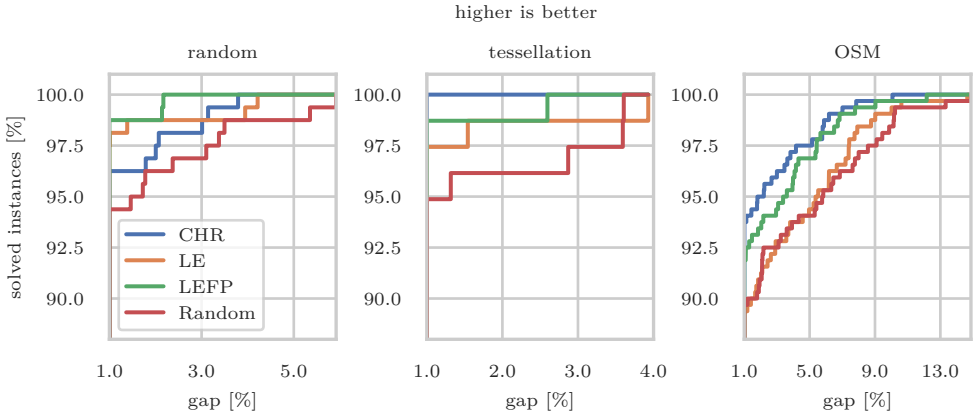


Figure 14 Gaps for different root node strategies after 60 s with 1 % optimality tolerance, separated by instance type. CHR and LEFP outperform the others, with CHR strongest on OSM and tessellation instances and LEFP more robust on random instances.

in 95.16 % and 94.80 % of the instances, respectively. LEFP was more robust for instances that contain many intersections (such as our random instances), while CHR performed best on average for the OSM and tessellation instances which have many polygons on their convex hull. LE performed reasonably well but suffers from bad initial lower bounds. Random root selection is not recommended due to high variance and inferior median performance.

4.4 RQ2: Search strategy

The choice of search strategy affects how quickly solutions are found and how fast the global lower bound improves. We compared the three strategies from Section 3.4 – DFS, BFS, and combined DFS+BFS – with a time limit of 60 s for optimality tolerances of 1 % and 5 %. The outcomes are shown in Figure 15. We observe that for 5 % optimality tolerance, DFS and DFS+BFS have similar performance with about 98 % of the instances solved to optimality, while DFS+BFS is the winner for smaller tolerances with about 95 % of the instances solved to 1 % optimality. DFS achieves the fastest time-to-first-solution in most instances, while BFS improves the global lower bound more quickly. The combined DFS+BFS provides a good trade-off, particularly when slightly relaxed optimality tolerances are acceptable (it often terminates quickly for moderate gaps). For the remainder of our experiments, we use the DFS+BFS strategy as it provides the best overall performance.

4.5 RQ3: Decomposition branching

We evaluate the performance of the three modeling strategies for non-convex polygons introduced in Section 3.3. IM represents non-convex regions using indicator variables, which can result in long solve times at individual nodes. In contrast, DB avoids indicator variables but increases the size of the search tree, potentially leading to more branching. In both DB and IM, each polygon is initially replaced by its convex hull and only treated as non-convex when necessary. EIM does not employ this lazy strategy and models all non-convex polygons explicitly from the outset. We compared all approaches with a time limit of 60 s and an optimality tolerance of 1 %, excluding the purely convex instances (480 instances remaining). The results are shown in Figure 16. DB and IM exhibit very similar performance. While DB is slightly faster on some instances, it solves fewer instances to within 1 % optimality within

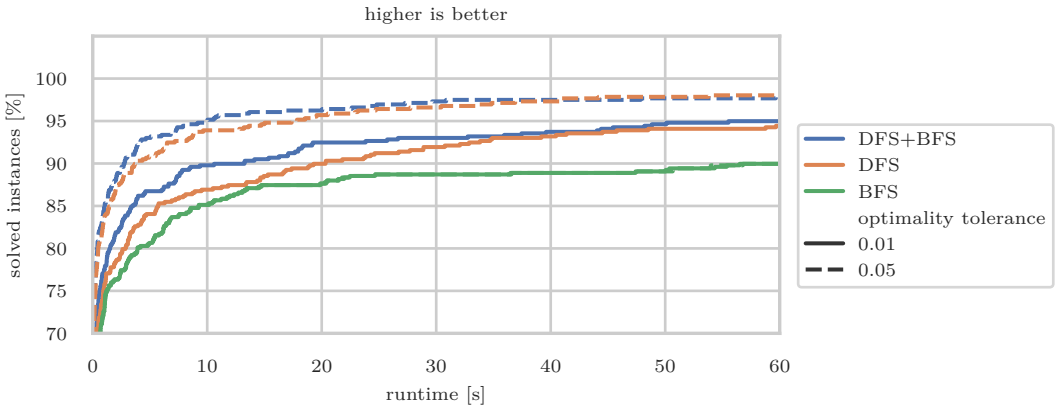


Figure 15 Instances solved within 60 s for different search strategies with 1 % and 5 % optimality tolerance. DFS+BFS offers the best trade-off between fast solutions (DFS) and rapid lower-bound improvement (BFS).

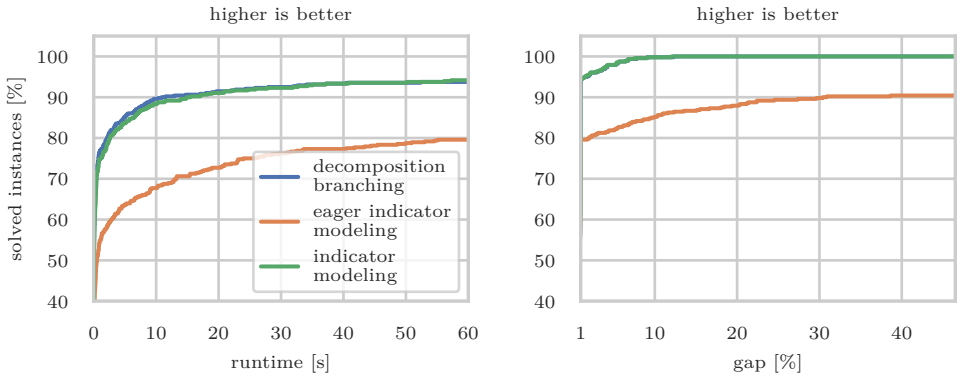


Figure 16 (Left) Runtime for solving instances with decomposition branching as well as eager and non-eager indicator-based modeling within a time limit of 60 s and 1 % optimality tolerance. (Right) Optimality gaps for the remaining instances. Decomposition branching has a slightly better performance than indicator-based modeling, and eager indicator-based modeling performs worse.

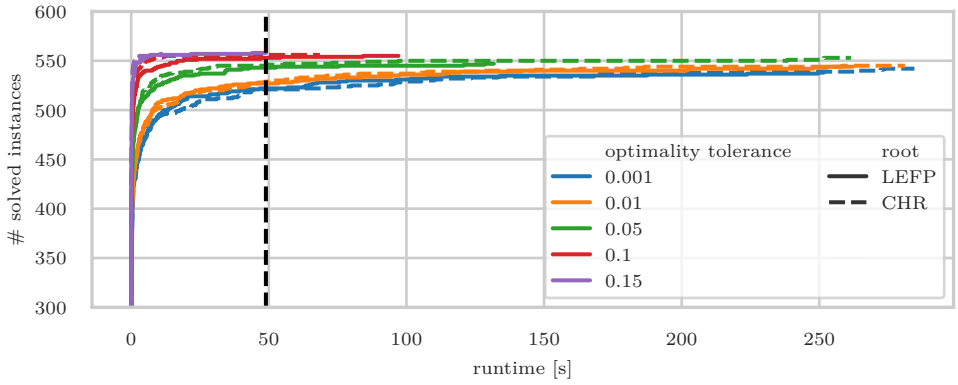


Figure 17 Number of instances solved within a given optimality tolerance within a given time. Wider gaps of 5 % to 15 % substantially reduce computation time, while for smaller gaps of 0.1 %, not all instances could be solved within the time limit.

the time limit (93.75 % for DB vs. 94.17 % for IM). In contrast, EIM performs substantially worse, solving only 79.58 % of instances within the time limit. Overall, these results support the use of the lazy initialization strategy for non-convex polygons. Because DB and IM perform similarly but DB avoids the need for indicator variables, we adopt decomposition branching as our default approach for the remainder of the experiments.

4.6 RQ4: Scalability with optimality tolerance

Relaxing the optimality tolerance is often acceptable in practical applications and can significantly reduce computation time in exact optimization. Gentilini et al. [29] reported optimal solutions for up to 16 convex regions which impose less complexity than our general polygonal regions. We evaluated the performance of both the CHR and LEFP root node strategies with DFS+BFS search for optimality tolerances of 0.1 %, 1 %, 5 %, 10 %, and 15 % with a time limit of 300 s. This instance set comprises 558 instances with sizes ranging from $n = 5$ to $n = 60$ polygons of varying complexity. The number of solved instances after a given time is shown in Figure 17. We observe negligible differences between 0.1 % and 1 %, whereas tolerating 5 %, 10 % or 15 % often yields large savings in solve time and increases the number of instances solved. For 0.1 % tolerance, runs using the CHR root node strategy yielded the best results with 97.13 % of all instances solved to optimality in less than 300 s. The instances not solved to optimality had either $n = 50$ or $n = 60$ polygons. More precisely, for $n = 50$, only 2 out of 65 instances were not solved to optimality within the time limit, with a maximum optimality gap of 0.83 %. Both instances can however be solved to the desired tolerance within 595 s and 368 s, respectively. The remaining 14 unsolved instances of size 60 all had gaps within 7 %. A subsequent run with a time limit of 900 s solved 5 additional instances to 0.1 % optimality, leaving 9 instances unsolved with a maximum gap of 5.53 %. This amounts to a total of 98.39 % of all instances solved to optimality within 900 s. Relaxing the requirements for an optimal solution to 15 % allows the LEFP strategy to solve all instances within less than 50 s.

Comparison to previous work. Prior to our geometric branch-and-bound approach, TSPN was commonly modeled as a MINLP and solved using general-purpose MINLP solvers. In these formulations, subtour elimination constraints are typically enforced through a cutting-plane framework where a relaxation of the problem is solved, subtours are detected for example via max-flow separation, and violated constraints are added as they are identified.

A representative example of this line of work is the method of Gentilini et al. [29]. Their approach integrates COUENNE, Ipopt, and CPLEX, and incorporates TSPN-specific strengthening techniques within COUENNE. Nevertheless, the method remains fundamentally based on the classical MINLP formulation and therefore inherits its high-dimensional spatial branching behavior. Empirical results reported in their study indicate practical tractability primarily for relatively small instances, typically with $n \leq 30$ within a few percent optimality gap, but on older hardware and with less efficient solvers than those available today.[1]

Our approach differs both conceptually and algorithmically. Instead of branching in the high-dimensional MINLP formulation, we perform branch-and-bound directly in the underlying planar geometry. Node relaxations are solved as SOCPs, allowing the algorithm to exploit the polynomial-time solvability of convex subproblems. This design shifts the computational effort from generic nonlinear spatial branching toward geometry-aware decisions.

4.7 Threats to Validity

All solutions were validated, and a set of unit tests ensured correctness of core components. Results were checked for consistency between objective values and lower bounds across different runs. Feasibility of the resulting tours was independently verified after the optimization. However, as with any empirical evaluation, there are threats to validity. Implementation bugs may affect results, though we took care to minimize this risk through testing and validation. The choice of Gurobi as the underlying SOCP solver may influence performance; different solvers could yield varying results. Due to the nature of these solvers, all computations were performed using floating-point arithmetic, which may introduce numerical inaccuracies. Finally, the selected instance sets, while diverse, may not cover all possible scenarios encountered in practice.

5 Conclusion and Future Work

We presented significant practical progress for the TSPN with arbitrary neighborhoods of differing sizes, not only picking the best out of $n!$ permutations, but also computing the best tour for a given order of regions; even for k convex regions with m vertices and given start and end point, the fastest known algorithm for this task requires a time of $O(k^2m)$ [42].

At this point, we are able to solve all instances in our benchmark set of size up to $n = 50$ to within 0.1 % of optimality within reasonable time, and 86.8 % of the larger instances of size $n = 60$, with small optimality gap for the remaining ones. With our current techniques, this size appears to be the threshold between manageable and intractable instances without resorting to runtimes in the order of hours.

This raises the question of how to extend this range by employing more advanced geometric ideas. We are optimistic that further progress is possible, e.g., by using enhanced pruning strategies and additional cuts during the branch-and-bound, making use of further theoretical insights into the structure of locally optimal trajectories.

[1] We note that a similar MIQCP-based approach was implemented by the authors in preliminary experiments which did not yield competitive results and can be found along with the source code.

References

1 Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2):473–506, 2020. doi:10.1287/IJOC.2018.0857.

2 Antonios Antoniadis, Mark de Berg, Sándor Kisfaludi-Bak, and Antonis Skarlatos. Computing smallest convex intersecting polygons. *Journal of Computational Geometry*, 16(1):167–202, 2025. doi:10.20382/jocg.v16i1a6.

3 David L. Applegate, Robert E. Bixby, Vasek Chvátal, William J. Cook, Daniel G. Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Oper. Res. Lett.*, 37(1):11–15, 2009. doi:10.1016/J.ORL.2008.09.006.

4 Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. The lawnmower problem. In *Canadian Conference on Computational Geometry (CCCG)*, pages 461–466, 1993. URL: https://cglab.ca/~cccg/proceedings/1993/Paper79.pdf.

5 Esther M. Arkin, Sándor P. Fekete, and Joseph S. B. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17:25–50, 2000. doi:10.1016/S0925-7721(00)00015-8.

6 Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

7 Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in SAT solving. In *Handbook of Satisfiability*, pages 391–435. IOS Press, 2021. doi:10.3233/FAIA200992.

8 S.P. Boyd and L. Vandenberghe. *Convex Optimization, Pt. 1*. Berichte über verteilte Messsysteme. Cambridge University Press, 2004. URL: https://books.google.de/books?id=mYm0bLd3fcoC.

9 Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Manlio Gaudioso. A novel discretization scheme for the Close Enough Traveling Salesman Problem. *Comput. Oper. Res.*, 78:163–171, 2017. doi:10.1016/J.COR.2016.09.003.

10 Bernard Chazelle and David P. Dobkin. Optimal convex decompositions. In Godfried T. Toussaint, editor, *Computational Geometry*, volume 2 of *Machine Intelligence and Pattern Recognition*, pages 63–133. North-Holland, 1985. doi:10.1016/B978-0-444-87806-9.50009-8.

11 Andrew Clark. A submodular optimization approach to the metric Traveling Salesman Problem with neighborhoods. In *IEEE Conference on Decision and Control, CDC*, pages 3383–3390, 2019. doi:10.1109/CDC40024.2019.9030031.

12 William J Cook, David L Applegate, Robert E Bixby, and Vasek Chvatal. *The Traveling Salesman Problem: A computational study*. Princeton University Press, 2011.

13 Walton Pereira Coutinho, Roberto Quirino do Nascimento, Artur Alves Pessoa, and Anand Subramanian. A branch-and-bound algorithm for the Close-Enough Traveling Salesman Problem. *INFORMS J. Comput.*, 28(4):752–765, 2016. doi:10.1287/IJOC.2016.0711.

14 Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levcopoulos, Mark H. Overmars, and A. Frank van der Stappen. TSP with neighborhoods of varying size. *J. Algorithms*, 57(1):22–36, 2005. doi:10.1016/J.JALGOR.2005.01.010.

15 Jindřiška Deckerová, Kristýna Kučerová, and Jan Faigl. On improvement heuristic to solutions of the close enough traveling salesman problem in environments with obstacles. In *2023 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2023. doi:10.1109/ECMR59166.2023.10256328.

16 Jindřiška Deckerová, Petr Váňa, and Jan Faigl. Combinatorial lower bounds for the generalized traveling salesman problem with neighborhoods. *Expert Systems with Applications*, 258:125185, 2024. doi:10.1016/j.eswa.2024.125185.

17 Erik D Demaine, Sándor P Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Computing convex partitions for point sets in the plane: The CG:SHOP Challenge 2020. *arXiv preprint arXiv:2004.04207*, 2020. arXiv:2004.04207.

18 Erik D Demaine, Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. Area-optimal simple polygonalizations: The CG Challenge 2019. *ACM Journal on Experimental Algorithms*, 27(2):1–12, 2022.

19 Andrea Di Placido, Claudia Archetti, Carmine Cerrone, and Bruce Golden. The generalized Close Enough Traveling Salesman Problem. *European Journal of Operational Research*, 310(3):974–991, 2023. doi:10.1016/j.ejor.2023.04.010.

20 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *Symposium on Theory of Computing (STOC)*, pages 473–482, 2003. doi:10.1145/780542.780612.

21 Adrian Dumitrescu and Joseph SB Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003. doi:10.1016/S0196-6774(03)00047-6.

22 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. Salzburg database of polygonal data: Polygons and their generators. *Data in Brief*, 31:105984, 2020. doi:10.1016/j.dib.2020.105984.

23 Khaled M. Elbassioni, Aleksei V. Fishkin, and René Sitters. On approximating the TSP with intersecting neighborhoods. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 213–222, 2006. doi:10.1007/11940128_23.

24 Jan Faigl. Gsoa: Growing self-organizing array – unsupervised learning for the close-enough traveling salesman problem and other routing problems. *Neurocomputing*, 312:120–134, 2018. doi:10.1016/j.neucom.2018.05.079.

25 Jan Faigl and Libor Přeučil. Self-organizing map for the multi-goal path planning with polygonal goals. In Timo Honkela, Włodzisław Duch, Mark Girolami, and Samuel Kaski, editors, *International Conference on Artificial Neural Networks and Machine Learning (ICANN)*, pages 85–92, 2011.

26 Sándor P. Fekete, Dominik Krupke, Michael Perk, Christian Rieck, and Christian Scheffer. A closer cut: Computing near-optimal tours for the Lawn Mowing Problem. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 1–14, 2023. doi:10.1137/1.9781611977561.

27 Sándor P. Fekete, Dominik Krupke, Michael Perk, Christian Rieck, and Christian Scheffer. The Lawn Mowing Problem: From algebra to algorithms. In *European Symposium on Algorithms (ESA)*, pages 45:1–45:18, 2023. doi:10.4230/LIPIcs.ESA.2023.45.

28 Barend Gehrels, Bruno Lalande, Mateusz Loskot, Adam Wulkiewicz, Menelaos Karavelas, and Fisikopoulos. Vissarion. Boost geometry 1.83, 2024. URL: https://www.boost.org/.

29 Iacopo Gentilini, François Margot, and Kenji Shimada. The Travelling Salesman Problem with neighbourhoods: MINLP solution. *Optimization Methods Software*, 28(2):364–378, 2013. doi:10.1080/10556788.2011.648932.

30 Daniel H Greene. The decomposition of polygons into convex parts. *Computational Geometry*, 1:235–259, 1983.

31 Gurobi Optimization LLC. Gurobi Optimizer 12.0. https://www.gurobi.com/, 2025.

32 Keld Helsgaun. Solving the equality generalized Traveling Salesman Problem using the Lin–Kernighan–Helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287, September 2015. doi:10.1007/s12532-015-0080-8.

33 IBM. IBM ILOG CPLEX Optimization Studio. https://www.ibm.com/products/ilog-cplex-optimization-studio, 2025.

34 Dae-Sung Jang, Hyeok-Joo Chae, and Han-Lim Choi. Optimal control-based uav path planning with dynamically-constrained tsp with neighborhoods. In *International Conference on Control, Automation and Systems (ICCAS)*, pages 373–378, 2017.

35 Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. Some applications of the generalized Travelling Salesman Problem. *Journal of the Operational Research Society*, 47(12):1461–1467, 1996. doi:10.1057/jors.1996.190.

36 Andrzej Lingas. The power of non-rectilinear holes. In *International Colloquium on Automata, Languages and Programming (ISAAC)*, pages 369–383, 1982. doi:10.1007/BFB0012784.

37 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. doi:10.1137/S0097539796309764.

38 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Symposium on Discrete Algorithms (SODA)*, pages 11–18, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283385.

39 OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org, 2017.

40 G. Reinelt. TSPLIB–a Traveling Salesman Problem library. *ORSA Journal of Computing*, 3(4):376–384, 1991. doi:10.1287/IJOC.3.4.376.

41 Stephen L. Smith and Frank Imeson. Glns: An effective large neighborhood search heuristic for the generalized Traveling Salesman Problem. *Computers & Operations Research*, 87:1–19, 2017. doi:10.1016/j.cor.2017.05.010.

42 Xuehou Tan and Bo Jiang. Efficient algorithms for touring a sequence of convex polygons and related problems. In *Theory and Applications of Models of Computation (TAMC)*, volume 10185, pages 614–627, 2017. doi:10.1007/978-3-319-55911-7_44.

43 The CGAL Project. CGAL 6.0. https://www.cgal.org, 2025.

44 Bo Yuan and Tiantian Zhang. Towards solving TSPN with arbitrary neighborhoods: A hybrid solution. In *Artificial Life and Computational Intelligence (ACALCI)*, pages 204–215, 2017. doi:10.1007/978-3-319-51691-2_18.