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

Sándor P. Fekete[1], Rouven Kniep[1], Dominik Krupke[1], and Michael Perk[1]

1 **Department of Computer Science, TU Braunschweig**
`s.fekete@tu-bs.de, {kniep, krupke, perk}@ibr.cs.tu-bs.de`

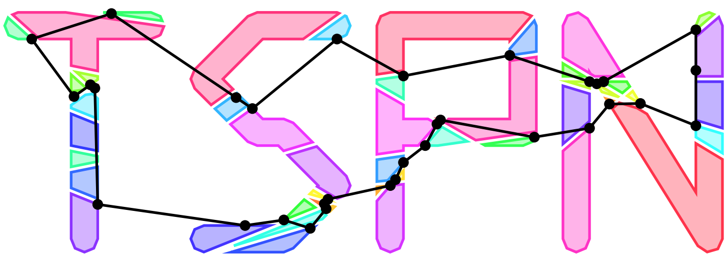
Abstract

The Traveling Salesman Problem with Neighborhoods (TSPN) generalizes the classical Traveling Salesman Problem by requiring a tour to visit 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 provable optimality. In this paper, we propose a branch-and-bound algorithm that solves polygonal TSPN instances to optimality. Through computational experiments on 500 benchmark instances with 50 polygons each, our method achieves a 85.6 % optimality rate within 60 s. We also explore the impact of key design choices, providing insights into effective solution strategies for TSPN.

Lines 175

1 Introduction

A natural generalization of the classical Traveling Salesman Problem (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.



50 polygons, solved in 0.75s

Figure 1 Example TSPN instance with a feasible solution.

While the geometric TSP allows both polynomial-time approximation schemes [21, 4] and solution to provable optimality for point sets of considerable size (such as the 85 900-city instance solved by [1, 7]), the TSPN is considerably more challenging, both in theory (with APX-hardness [9, 10]), and practice (with the state of the art being provably optimal solutions for benchmark instances up to 16 regions [16]).

Contribution. We present a branch-and-bound algorithm that solves TSPN instances to provable optimality. Across 500 benchmark instances with 50 polygons each, 85.6 %

41st European Workshop on Computational Geometry, Liblice, Czech republic, April 9–11, 2025.
This is an extended abstract of a presentation given at EuroCG'25. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

are solved to optimality within 60 s, significantly advancing the state-of-the-art. We also evaluate the impact of various algorithmic design choices.

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 [2, 3, 13, 14]. Branch-and-bound algorithms have been studied for CE-TSP [11, 8]; however, the TSPN allows arbitrary (including non-convex) neighborhoods, making it more general. Many heuristics and approximation algorithms for the TSPN rely on assumptions (e.g., fatness [22], disjoint neighborhoods [9, 10], or comparable region diameters [12]) to manage its APX-hardness. Specialized approaches address specific settings such as aerial vehicle routing [19], and hybrid methods combine meta-heuristics with TSP solvers [24]. Non-convex Mixed Integer Nonlinear Programming (MINLP) formulations have been proposed for TSPN, including symmetric and asymmetric variants [16], 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 Minimum Spanning Tree with Neighborhoods [6].

2 Branch-and-Bound Algorithm

Our algorithm (Algorithm 1) builds on the branch-and-bound framework by Coutinho et al. [8] for the CE-TSP, improving and extending it to handle the TSPN. We begin by constructing a root node using a universal ordering on a subset of the polygons (see Section 2.2). From there, we branch by selecting a polygon that is not yet visited in the current solution and creating a new branch for each possible insertion position (Section 2.3).

For the sequence of polygons in each node, we solve a Second-Order Cone Program (SOCP) to obtain the optimal solution for that ordering (Section 2.1). If this solution intersects all polygons, it is a feasible solution; otherwise, the SOCP value provides a valid lower bound to prune the node if a superior solution is already known. Because evaluating a single node can yield multiple child nodes, we apply a search strategy to decide which node to process next (Section 2.4).

Throughout the search, the algorithm maintains the best known (feasible) solution and the node with the lowest lower bound, which together define the current optimality gap. We may terminate once this gap is smaller than a desired threshold.

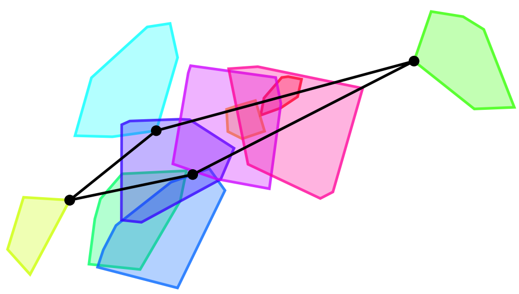


Figure 2 A sequence with only 4 polygons is feasible for this example instance with 10 polygons.

Algorithm 1 Branch-and-Bound Algorithm

Require: Set of polygons $\mathcal{I}$
  Preprocess and simplify $\mathcal{I}$.
  $\mathcal{Q} \leftarrow [\,]$ ▷ Queue of leaf nodes to explore
  $\mathcal{Q}$.push(getRootNode($\mathcal{I}$)) ▷ See Section 2.2
  $ub \leftarrow \infty$
  while $\mathcal{Q} \neq \emptyset \wedge \min\{v'.lb \mid v' \in Q\} < ub$ **do**
    $v \leftarrow$ getNextNode($\mathcal{Q}$) ▷ Search strategy, see Section 2.4
    Remove v from $\mathcal{Q}$
    if $v.lb \geq ub$ **then**
      continue
    end if
    if Solution in v covers all polygons in $\mathcal{I}$ **then**
      $ub \leftarrow v.lb$
    else
      for $child \in$ Branch(v) **do** ▷ Branching, see Section 2.3
        $\mathcal{Q}$.push($child$)
      end for
    end if
  end while
  return Solution corresponding to ub, or $\perp$ if no solution was found.

2.1 Touring a Sequence of Polygons

▶ **Theorem 2.1.** *Let $P_0, \ldots, P_{n-1} \subset \mathbb{R}^2$ 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. 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

$$\hat{x}_i \geq x_i - x_{i+1}, \quad \hat{x}_i \geq x_{i+1} - x_i, \quad \hat{y}_i \geq y_i - y_{i+1}, \quad \hat{y}_i \geq y_{i+1} - y_i,$$

and impose second-order cone constraints $d_i^2 \geq \hat{x}_i^2 + \hat{y}_i^2$. Minimizing $\sum_{i=0}^{n-1} d_i$ can be done in polynomial time, and yields the shortest tour as d_i will be tight in the optimal solution. ◀

To handle non-convex polygons P_i', including those with holes, we propose an alternative constraint set $S_i'(x, y)$ that employs binary variables and can be expressed as a Mixed Integer Second-Order Cone Program (MISOCP). Suppose each P_i' can be decomposed into a finite set of convex polygons $\mathcal{R}_i$, see Figure 3. We introduce a binary variable $r_c \in \mathbb{B}$ for each $c \in \mathcal{R}_i$ and require exactly one of these polygons to be active by enforcing $\sum_{c \in \mathcal{R}_i} r_c = 1$. For each convex polygon c, let $S_i^c(x, y)$ denote its associated constraints. We then ensure $(x, y) \in P_i'$ by imposing the implications $r_c \implies S_i^c(x, y)$ for all $c \in \mathcal{R}_i$. These implications can be enforced via indicator constraints (available in many solvers) or through Big-M linearizations, where M can be limited by the bounding box of P_i'.

For polygons without holes, a minimum convex decomposition can be computed in polynomial time [17, 5]. For polygons with holes, a decomposition is always feasible by triangu-

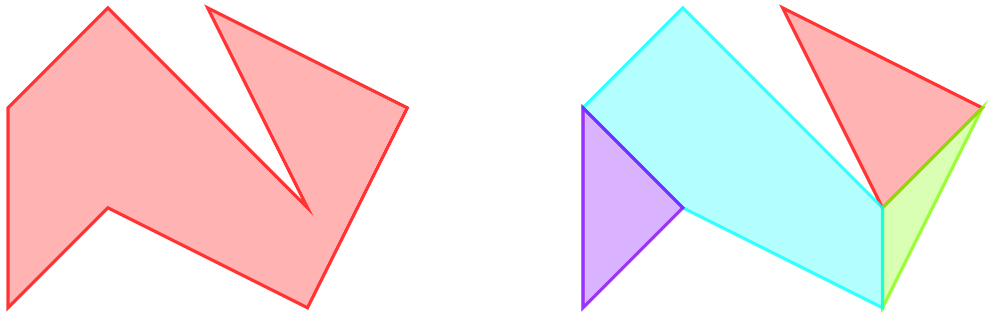


Figure 3 A non-convex polygon (left) and its decomposition in convex areas (right).

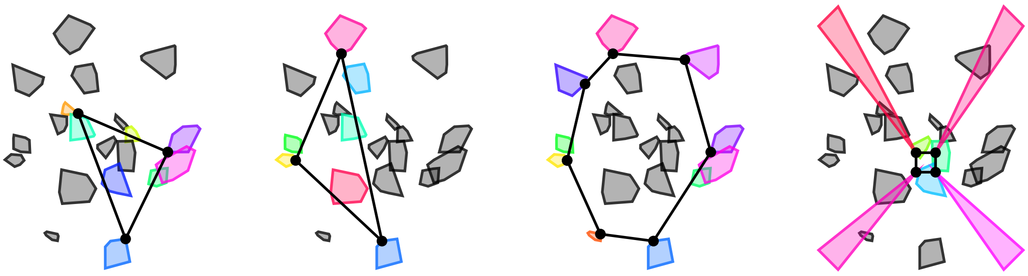


Figure 4 Root node strategies (left to right): Random, longest edge + farthest polygon, convex hull, and an example demonstrating poor convex hull performance.

lation, but finding a minimal decomposition is NP-hard [20]. Solving the resulting MISOCP is also NP-hard, and the use of indicator variables or Big-M constraints can lead to weak relaxations, making these methods computationally expensive in practice. Thus, we will investigate later to lazily decompose the polygons in our branch-and-bound algorithm, instead of using the MISOCP formulation directly.

2.2 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 three methods for selecting these polygons: **Random:** Select any three polygons at random. **Longest Edge+Farthest Polygon (LEFP):** Pick two polygons with the largest pairwise distance, then add the polygon farthest from these two. **Convex Hull (CHR):** Exploit the observation that any set of disjoint polygons on the instance's convex hull must appear in the same order in some optimal solution. Although this strategy may include more than three polygons in the root sequence, certain instances can yield weaker relaxations at the root node. Figure 4 illustrates all three strategies, including a case where CHR performs poorly.

2.3 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. Figure 5 illustrates this approach. We compare two strategies for choosing the polygon to branch on:

Random: Select a missing polygon uniformly at random. **Farthest Polygon:** Select the polygon that lies farthest from the current tour.

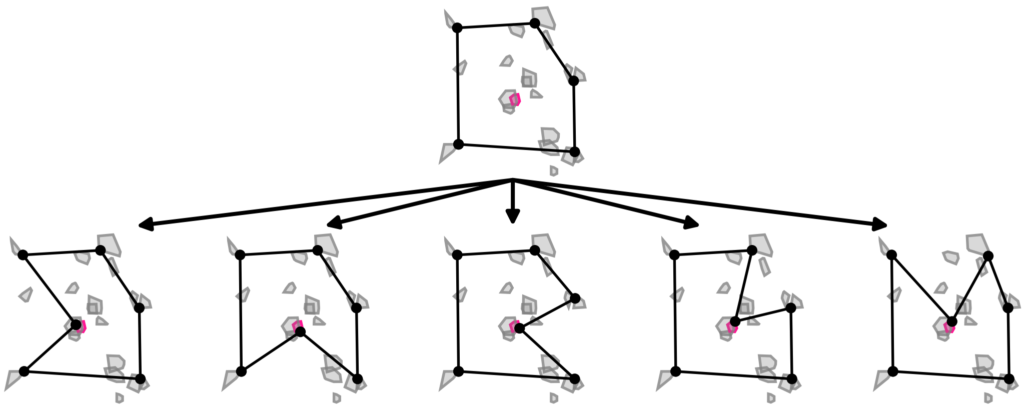


Figure 5 Path from the root to a leaf in a BnB tree using the farthest polygon branching strategy.

Because each polygon is initially replaced by its convex hull for efficiency, a polygon may appear in the partial sequence but still be effectively unvisited. If the MISOCP formulation is used, we can simply replace the convex hull with the polygon itself. Otherwise, we must branch on this polygon by decomposing it into convex parts and creating one branch per part (see Figure 6). This ensures that every leaf node includes the polygon, with at least one leaf containing its optimal hitting point.

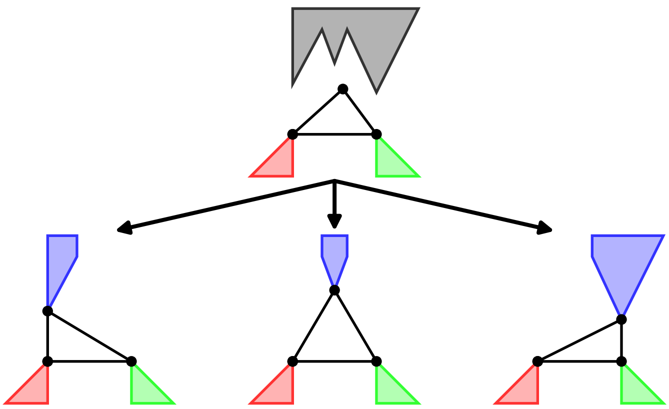


Figure 6 Branching on a non-convex polygon by decomposing it into convex pieces.

2.4 Search Strategies

We implement four different strategies to select the next node for exploration during the branch-and-bound search: **Random:** Pick a node uniformly at random from the queue. **BFS:** Choose the node with the best, i.e., smallest, lower bound. **DFS:** Continue exploring the best child of the current node, aiming to quickly improve the upper bound. **DFS+BFS:** Initially proceed like DFS, but whenever a node is pruned or a new feasible solution is discovered, sort the queue by lower bounds. This approach combines the fast upper-bound updates of DFS with the tighter lower-bound focus of BFS.

3 Experiments

In the following, we investigate how various algorithmic choices affect performance. We tested 500 instances, each containing 50 random polygons: 45 % convex, 45 % concave (up to 10 units), and 10 % larger concave polygons with holes (up to 20 units), see Figure 7 for examples. The instance `n50_ps70_001` is used for the convergence plots throughout this section. We regard a solution as optimal if its optimality gap is below 0.01 % within 60 s.

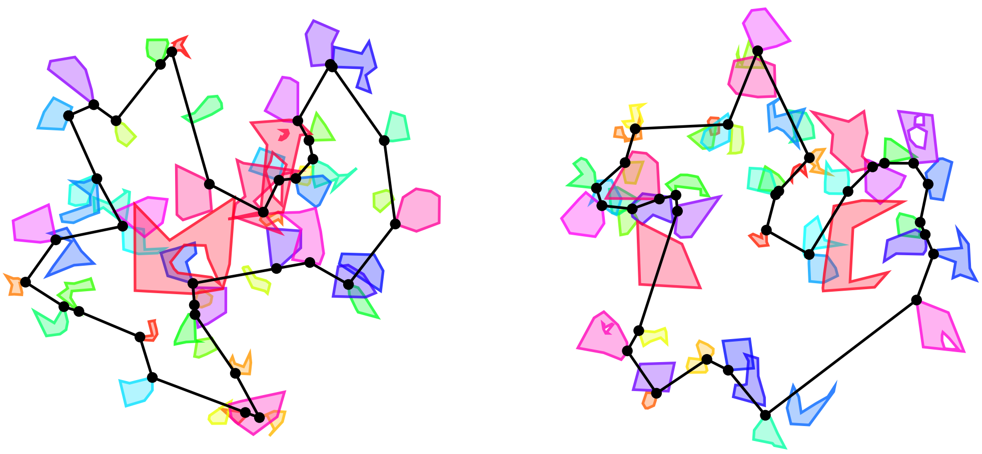


Figure 7 Optimal solutions for `n50_ps70_001` (left) and `n50_ps70_002` (right).

Our algorithm is implemented in C++ and uses Gurobi 12.0 [18] to solve the mathematical programs. Geometry operations rely on Boost.Geometry 1.83 [15] and CGAL 6.0.1 [23] with exact predicates and constructions. We compiled using `g++` 13.3 and ran all tests on an AMD Ryzen 7900 workstation with 96 GiB of DDR5-5200 RAM under Ubuntu 24.04.

Preprocessing Initial preprocessing to simplify the instances showed a modest improvement on some instances and improved average runtimes slightly. However, no additional instances were solved in time; see Figure 8.

Warm Start Heuristically computing an initial solution only benefited the random search strategy (Figure 9). For other strategies, the high upfront cost (usually between 10 s to 60 s) of the naive algorithm used outweighed gains.

Root Node The choice of root node sequence critically affects performance (Figure 10). Using a convex hull root node (CHR) was generally fastest, while a LEFP approach performed better on instances with large polygons. A random strategy is not recommended.

Search Strategy DFS found solutions fastest, whereas BFS proved optimality fastest. A combined DFS+BFS strategy is a good compromise; see Figure 11. It excels when slightly relaxed optimality tolerances are acceptable.

Polygon Selection When selecting the next polygon to branch on, choosing the *farthest polygon* consistently outperformed random selection (Figure 12).

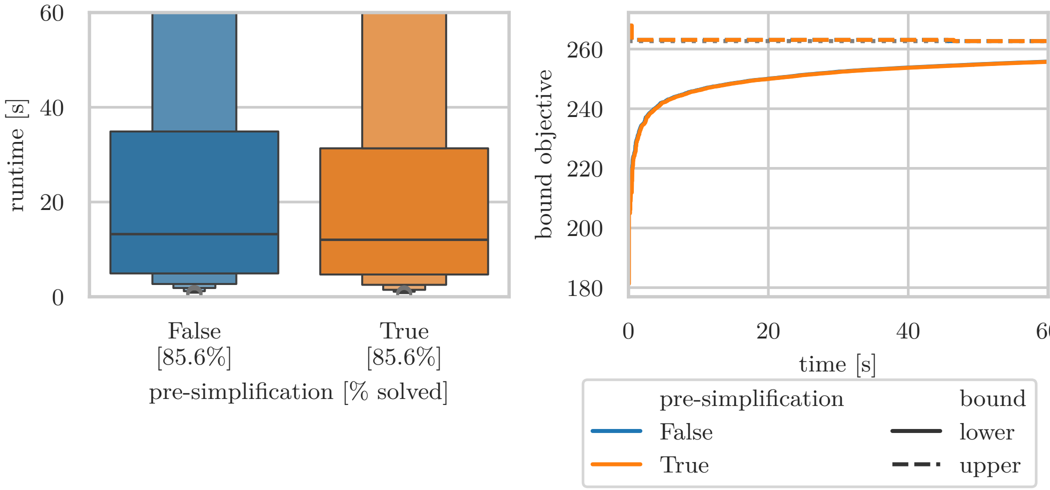


Figure 8 Runtime (left) and bound convergence (right) for different preprocessing settings. Sometimes pre-simplification improved initial bounds and sped up convergence.

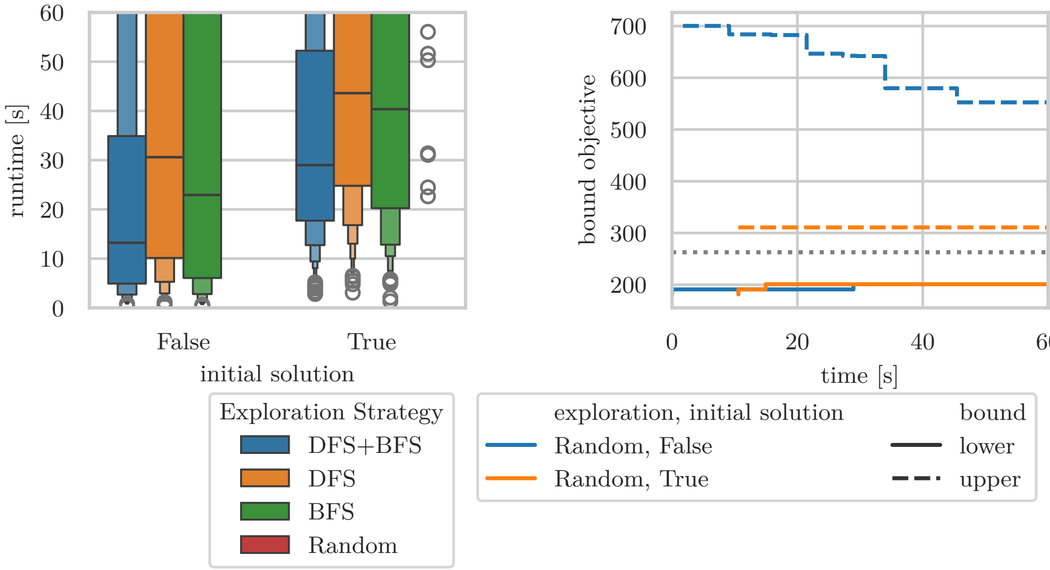


Figure 9 Runtime analysis (left) and random node exploration bound convergence (right) for different initial solutions. Other exploration strategies showed no improvement from warm starts.

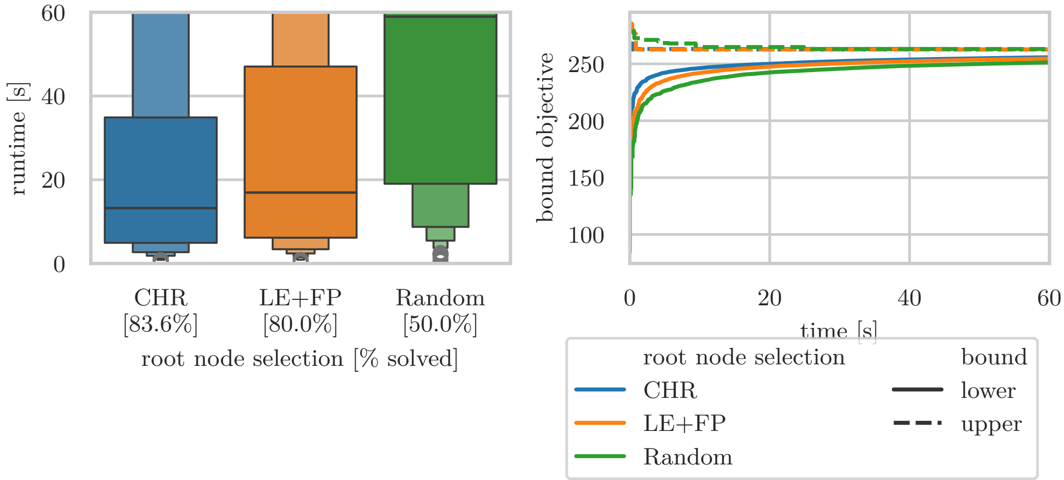


Figure 10 Runtime (left) and bound convergence (right) for different root node choices. CHR performed best overall; however, for larger polygons, LEFP was more robust.

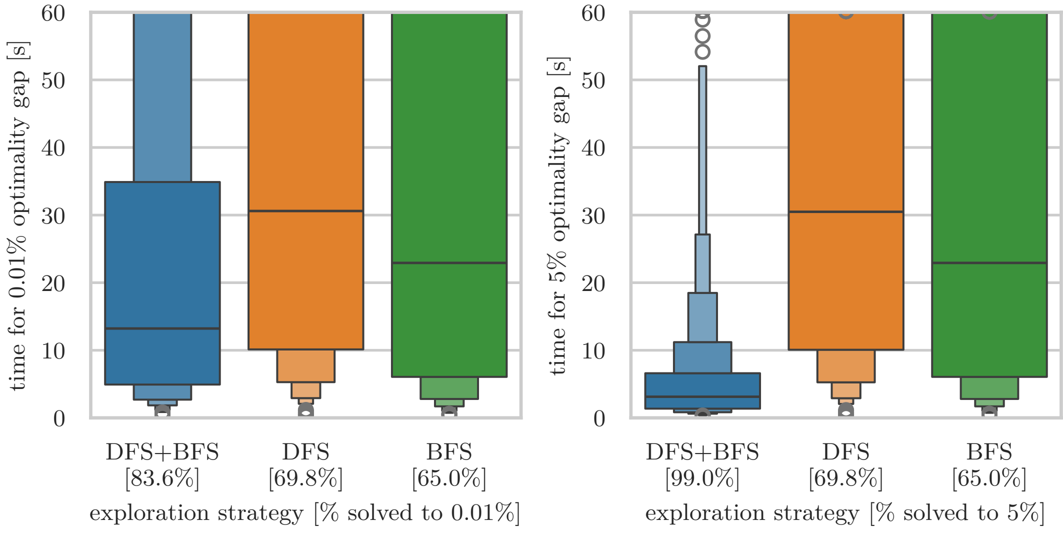


Figure 11 Runtime for 0.01 % (left) and 5 % (right) optimality gap using different search strategies. With a 5 % gap, DFS+BFS converges quickly.

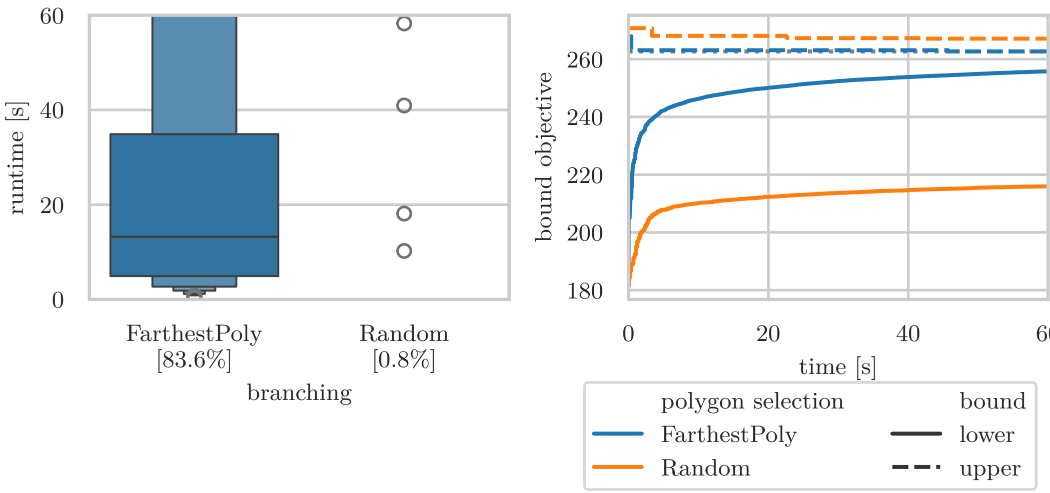


Figure 12 Runtime (left) and bound convergence (right) for different polygon selection methods. Furthest polygon significantly improved performance over random.

Decomposition Modeling Decomposition branching yields faster and more reliable performance than indicator modeling in Gurobi (Figure 13). Although indicator modeling is simpler to implement, it leads to weaker relaxations.

Optimality Tolerance Relaxing the optimality gap significantly reduces runtime for gaps of 5 % to 10 %; see Figure 14. Differences between 0.01 % and 0.1 % are negligible, but a gap of 5 % or higher often saves substantial time.

Threats to Validity All solutions were validated, and a set of unit tests ensured correctness of core components. Results were checked for consistency between upper and lower bounds. Moreover, the selected 500 instances may not be fully representative of real-world scenarios, though the set provides diversity by fixing instance size and varying polygon shapes.

4 Conclusion and Future Work

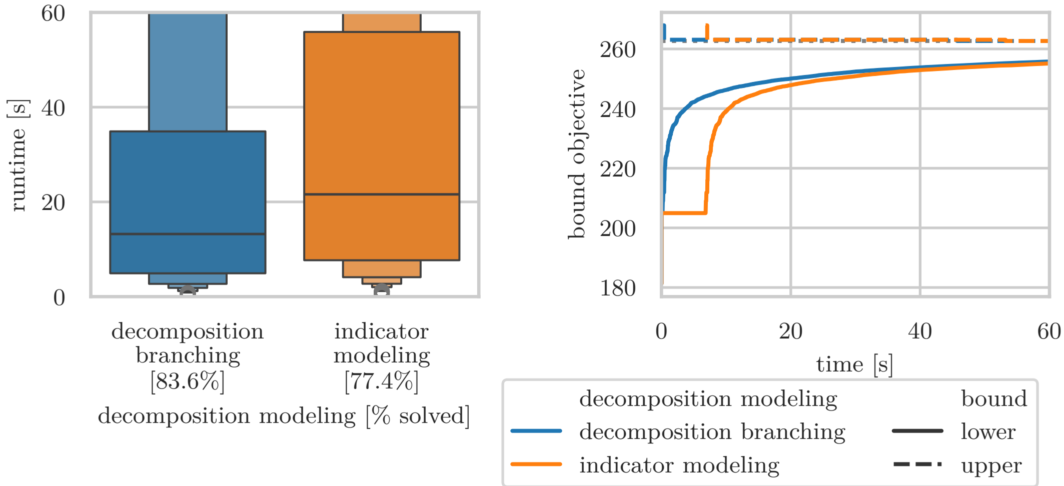
In this paper, we presented a branch-and-bound algorithm for the TSPN and evaluated the impact of various design decisions on its performance. While most of the results aligned with our expectations, it was surprising to find that manually branching on the decomposition of non-convex polygons outperformed handling them with Gurobi. Looking ahead, we have several ideas for further improvements, including enhanced early-pruning strategies and more advanced parallelization techniques.

Acknowledgments

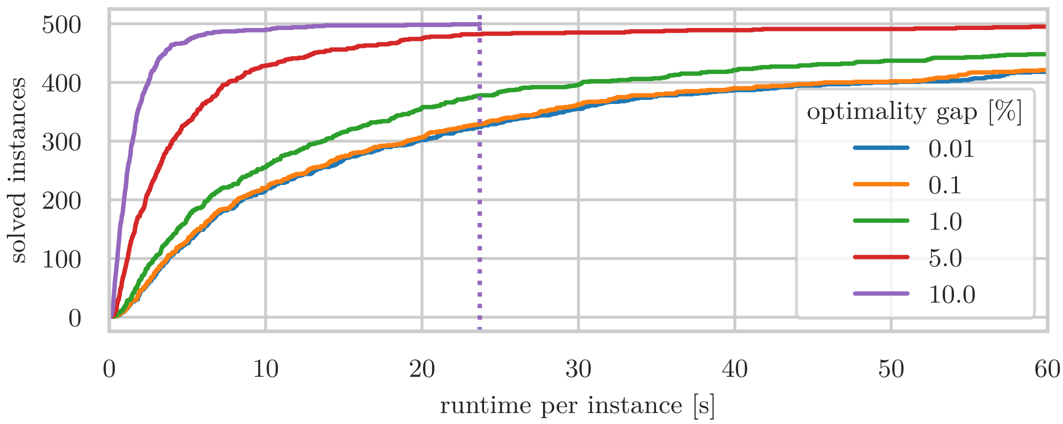
The implementation builds on code from an enhanced branch-and-bound algorithm for the CE-TSP developed by Dominik Krupke, Barak Ugav, and Michael Perk.

References

1 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`.



157 **Figure 13** Runtime (left) and bound convergence (right) for different modeling approaches.
158 Decomposition branching converged faster, whereas indicator modeling slowed early convergence.



162 **Figure 14** Number of instances solved within a given gap versus time. Wider gaps of 5 % to
163 10 % substantially reduce computation time.

2 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`.

3 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`.

4 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`.

5 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`.

6 Andrew Clark. A submodular optimization approach to the metric traveling salesman problem with neighborhoods. In *58th IEEE Conference on Decision and Control, CDC*, pages 3383–3390. IEEE, 2019. `doi:10.1109/CDC40024.2019.9030031`.

7 William J Cook, David L Applegate, Robert E Bixby, and Vasek Chvatal. *The traveling salesman problem: a computational study*. Princeton University Press, 2011.

8 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`.

9 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. In Rolf H. Möhring and Rajeev Raman, editors, *10th Annual European Symposium ESA*, volume 2461, pages 187–199. Springer, 2002. `doi:10.1007/3-540-45749-6\_20`.

10 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`.

11 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`.

12 Adrian Dumitrescu and Joseph SB Mitchell. Approximation algorithms for tsp with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003.

13 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`.

14 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`.

15 Barend Gehrels, Bruno Lalande, Mateusz Loskot, Adam Wulkiewicz, Menelaos Karavelas, and Fisikopoulos. Vissarion. Boost geometry 1.82, 2023. URL: `https://www.boost.org/`.

16 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`.

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

18 Gurobi Optimization LLC. Gurobi Optimizer 10.0. `https://www.gurobi.com/`, 2023.

19 Dae-Sung Jang, Hyeok-Joo Chae, and Han-Lim Choi. Optimal control-based UAV path planning with dynamically-constrained TSP with neighborhoods. *CoRR*, 2016. `doi:10.48550/arXiv.1612.06008`.

20 Andrzej Lingas. The power of non-rectilinear holes. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium*, volume 140, pages 369–383. Springer, 1982. `doi:10.1007/BFB0012784`.

21 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`.

22 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 11–18. Society for Industrial and Applied Mathematics, 2007.

23 The CGAL Project. CGAL 5.6. `https://www.cgal.org`, 2023.

24 Bo Yuan and Tiantian Zhang. Towards solving TSPN with arbitrary neighborhoods: A hybrid solution. In Markus Wagner, Xiaodong Li, and Tim Hendtlass, editors, *Artificial Life and Computational Intelligence - Third Australasian Conference, ACALCI*, volume 10142, pages 204–215, 2017. `doi:10.1007/978-3-319-51691-2\_18`.

A Pre-Simplification

We apply several simplification rules to reduce the complexity of an instance. that do not exclude any optimal solution.

Superset Elimination If a polygon $p \in \mathcal{I}$ fully contains another polygon $q \in \mathcal{I}$, then p can be safely removed from the instance as any solution covering q will also cover p, see Figure 15.

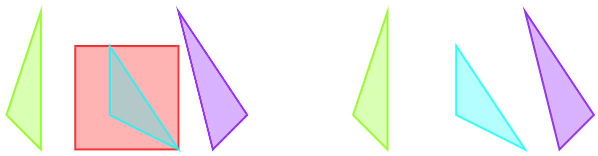


Figure 15 Example instance (left) with superset elimination applied (right). A polygon can be removed if it fully contains any other instance polygon.

Hole Removal For a polygon $p \in \mathcal{I}$ that contains a hole h, h can be removed if there is another polygon $q \in \mathcal{I}$ that does not intersect h. If not all holes can be removed, the polygon can be replaced by any other polygon that contains at least one other polygon and the complements of all remaining holes, see Figure 16.

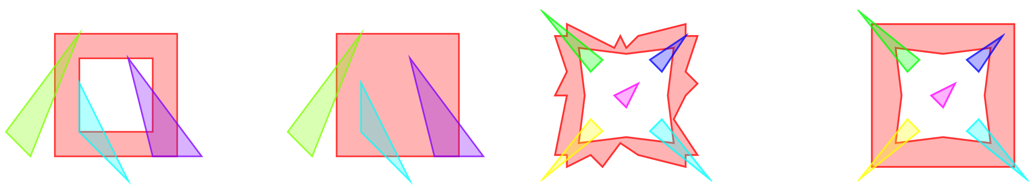
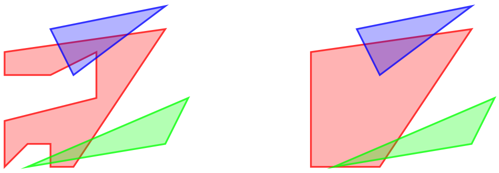


Figure 16 Example instance (left) with hole removal applied (center left). A hole can be removed if there is another polygon that does not intersect it. The hole in the instance (center right) cannot be removed but the polygon can be simplified (right).

Convex Hull Filling If a polygon $p \in \mathcal{I}$ cuts the instance into seperate sets $\mathcal{I}_1, \mathcal{I}_2 \subseteq \mathcal{I}$ such that any TSPN tour connnecting $\mathcal{I}_1$ and $\mathcal{I}_2$ must intersect p, then p can be simplified to its convex hull, see Figure 17.



181 **Figure 17** Example instance (left) with convex hull filling applied (right).