Computing Nonsimple Polygons of Minimum Perimeter

Sándor P. Fekete[1(✉)], Andreas Haas[1], Michael Hemmer[1], Michael Hoffmann[2], Irina Kostitsyna[3], Dominik Krupke[1], Florian Maurer[1], Joseph S.B. Mitchell[4], Arne Schmidt[1], Christiane Schmidt[5], and Julian Troegel[1]

[1] TU Braunschweig, Braunschweig, Germany
s.fekete@tu-bs.de
[2] ETH Zurich, Zurich, Switzerland
[3] TU Eindhoven, Eindhoven, The Netherlands
[4] Stony Brook University, Stony Brook, NY, USA
[5] Linköping University, Linköping, Sweden

Abstract. We provide exact and approximation methods for solving a geometric relaxation of the Traveling Salesman Problem (TSP) that occurs in curve reconstruction: for a given set of vertices in the plane, the problem Minimum Perimeter Polygon (MPP) asks for a (not necessarily simply connected) polygon with shortest possible boundary length. Even though the closely related problem of finding a minimum cycle cover is polynomially solvable by matching techniques, we prove how the topological structure of a polygon leads to NP-hardness of the MPP. On the positive side, we show how to achieve a constant-factor approximation.

When trying to solve MPP instances to provable optimality by means of integer programming, an additional difficulty compared to the TSP is the fact that only a subset of subtour constraints is valid, depending not on combinatorics, but on geometry. We overcome this difficulty by establishing and exploiting additional geometric properties. This allows us to reliably solve a wide range of benchmark instances with up to 600 vertices within reasonable time on a standard machine. We also show that using a natural geometry-based sparsification yields results that are on average within 0.5 % of the optimum.

Keywords: Traveling Salesman Problem (TSP) · Minimum Perimeter Polygon (MPP) · Curve reconstruction · NP-hardness · Exact optimization · Integer programming · Computational geometry meets combinatorial optimization

1 Introduction

For a given set V of points in the plane, the Minimum Perimeter Polygon (MPP) asks for a polygon P with vertex set V that has minimum possible boundary length. An optimal solution may not be simply connected, so we are faced with a geometric relaxation of the Traveling Salesman Problem (TSP).

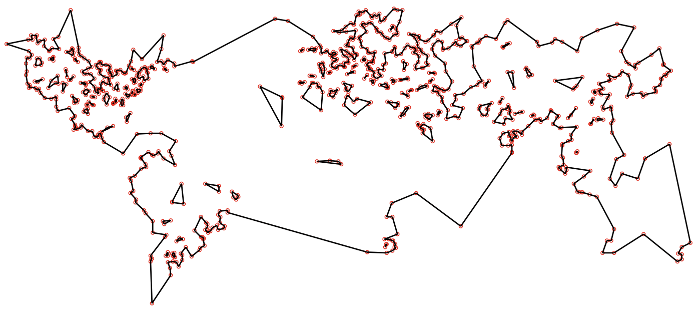


Fig. 1. A Minimum Perimeter Polygon for an instance with 960 vertices.

The TSP is one of the classic problems of Combinatorial Optimization. NP-hard even in special cases of geometric instances (such as grid graphs), it has served as one of the prototypical testgrounds for developing outstanding algorithmic approaches. These include constant-factor approximation methods (such as Christofides' 3/2 approximation [6] in the presence of triangle inequality, or Arora's [4] and Mitchell's [20] polynomial-time approximation schemes for geometric instances), as well as exact methods (such as Grötschel's optimal solution to a 120-city instance [14] or the award-winning work by Applegate et al. [2] for solving a 13509-city instance within 10 years of CPU time.) The well-established benchmark library TSPLIB [23] of TSP instances has become so widely accepted that it is used as a benchmark for a large variety of other optimization problems. See the books [15,18] for an overview of various aspects of the TSP and the books [3,7] for more details on exact optimization.

From a geometric point of view, the TSP asks for a shortest polygonal chain through a given set of vertices in the plane; as a consequence of triangle inequality, the result is always a simple polygon of minimum perimeter. Because of the fundamental role of polygons in geometry, this has made the study of TSP solutions interesting for a wide range of geometric applications. One such context is geometric shape reconstruction, where the objective is to re-compute the original curve from a given set of sample points; see Giesen [13], Althaus and Mehlhorn [1] or Dey et al. [9] for specific examples. However, this only makes sense when the original shape is known to be simply connected, i.e., bounded by a single closed curve. More generally, a shape may be multiply connected, with interior holes. Thus, computing a simple polygon may not yield the desired answer. Instead, the solution may be a Minimum Perimeter Polygon (MPP): for a set V of points in the plane, find a not necessarily simple polygon P with vertex set V, such that the boundary of P has smallest possible length[1]. See Fig. 1 for an optimal solution of an instance with 960 points; this also shows the possibly intricate structure of an MPP.

[1] Note that we exclude degenerate holes that consist of only one or two vertices.

While the problem MPP[2] asks for a cycle cover of the given set of vertices (as opposed to a single cycle required by the TSP), it is important to note that even the more general geometry of a polygon with holes imposes some topological constraints on the structure of boundary cycles; as a consequence, an optimal 2-factor (a minimum-weight cycle cover of the vertices, which can be computed in polynomial time) may not yield a feasible solution. Fekete et al. [11] gave a generic integer program for the MPP (and other related problems) that yields optimal solutions for instances up to 50 vertices. However, the main challenges were left unresolved. What is the complexity of computing an MPP? Is it possible to develop constant-factor approximation algorithms? And how can we compute provably optimal solutions for instances of relevant size?

Our Results
In this paper, we resolve the main open problems related to the MPP.

– We prove that the MPP is NP-hard. This shows that despite of the relationship to the polynomially solvable problem of finding a minimum 2-factor, dealing with the topological structure of the involved cycles is computationally difficult.
– We give a 3-approximation algorithm.
– We provide a general IP formulation with $O(n^2)$ variables to ensure a valid solution for the MPP.
– We describe families of cutting planes that significantly reduce the number of iterations needed to eliminate outer components and holes in holes, leading to a practically useful formulation.
– We present experimental results for the MPP, solving instances with up to 1000 points in the plane to provable optimality within 30 min of CPU time.
– We also consider a fast heuristic that is based on geometric structure, restricting the edge set to the Delaunay triangulation. Experiments on structured random point sets show that solutions are on average only about 0.5 % worse than the optimum, with vastly superior runtimes.

2 Complexity

Theorem 1. *The MPP problem is NP-hard.*

The proof is based on a reduction from the Minimum Vertex Cover problem for planar graphs. Details are omitted for lack of space; see the full version of the paper [12] for the detailed proof.

3 Approximation

In this section we show that the MPP can be approximated within a factor of 3. Note that we only sketch the general approach, skipping over some details for lack of space; a full proof is given in the full version of the paper [12].

[2] For simplicity, we will also refer to the problem of computing an MPP as "the MPP".

Theorem 2. *There exists a polynomial time* 3*-approximation for the MPP.*

Proof. We compute the convex hull, $CH(V)$, of the input set; this takes time $O(n \log h)$, where h is the number of vertices of the convex hull. Note that the perimeter, $|CH(V)|$, of the convex hull is a lower bound on the length of an optimal solution ($OPT \geq |CH(V)|$), since the outer boundary of any feasible solution polygon must enclose all points of V, and the convex hull is the minimum-perimeter enclosure of V.

Let $U \subseteq V$ be the input points interior to $CH(V)$. If $U = \varnothing$, then the optimal solution is given by the convex hull. If $|U| \leq 2$, we claim that an optimal solution is a simple (nonconvex) polygon, with no holes, on the set V, given by the TSP tour on V; since $|U| = 2$ is a constant, it is easy to compute the optimal solution in polynomial time, by trying all possible ways of inserting the points of U into the cycle of the points of V that lie on the boundary of the convex hull, $CH(V)$.

Thus, assume now that $|U| \geq 3$. We compute a minimum-weight 2-factor, denoted by $\gamma(U)$, on U, which is done in polynomial-time by standard methods [8]. Now, $\gamma(U)$ consists of a set of disjoint simple polygonal curves having vertex set U; the curves can be nested, with possibly many levels of nesting. We let F denote the directed *nesting forest* whose nodes are the cycles (connected components) of $\gamma(U)$ and whose directed edges indicate nesting (containment) of one cycle within another. Because an optimal solution consists of a 2-factor (an outer cycle, together with a set of cycles, one per hole of the optimal polygon), we know that $OPT \geq |\gamma(U)|$. (In an optimal solution, the nesting forest corresponding to the set of cycles covering all of V (not just the points U interior to $CH(V)$) is simply a single tree that is a star: a root node corresponding to the outer cycle, and a set of children adjacent to the root node, corresponding to the boundaries of the holes of the optimal polygon.) If the nesting forest F for our optimal 2-factor is a set of isolated nodes (i.e., there is no nesting among the cycles of the optimal 2-factor on U), then our algorithm outputs a polygon with holes whose outer boundary is the boundary of the convex hull, $CH(V)$, and whose holes are the (disjoint) polygons given by the cycles of $\gamma(U)$. In this case, the total weight of our solution is equal to $|CH(V)| + |\gamma(U)| \leq 2 \cdot OPT$.

Assume now that F has at least one nontrivial tree. We describe a two-phase process that transforms the set of cycles corresponding to F into a set of pairwise-disjoint cycles, each defining a simple polygon interior to $CH(V)$, with no nesting – the resulting simple polygons are disjoint, each having at least 3 vertices from $U \subset V$.

Phase 1 of the process transforms the cycles $\gamma(U)$ to a set of polygonal cycles that define *weakly simple* polygons whose interiors are pairwise disjoint. (A polygonal cycle β defines a *weakly simple* polygon P_β if P_β is a closed, simply connected set in the plane with a boundary, ∂P_β consisting of a finite union of line segments, whose traversal (e.g., while keeping the region P_β to one's left) is the (counterclockwise) cycle β (which can have line segments that are traversed twice, once in each direction).) The total length of the cycles at the end of phase 1 is at most 2 times the length of the original cycles, $\gamma(U)$. Then, phase 2 of the process transforms these weakly simple cycles into (strongly) simple cycles that define disjoint simple polygons interior to $CH(V)$. Phase 2 only does shortening operations on the weakly simple cycles; thus, the length of the resulting simple cycles at the end of phase 2 is at most 2 times the total length of $\gamma(U)$. Details of phase 1 and phase 2 processes are given in the full version of the paper. At the end of phase 2, we have a set of disjoint simple polygons within $CH(V)$, which serve as the holes of the output polygon, whose total perimeter length is at most $|CH(V)| + 2|\gamma(U)| \leq 3 \cdot OPT$. □

4 IP Formulation

4.1 Cutting-Plane Approach

In the following we develop suitable Integer Programs (IPs) for solving the MPP to provable optimality. The basic idea is to use a binary variable $x_e \in \{0, 1\}$ for any possible edge $e \in E$, with $x_e = 1$ corresponding to e being part of a solution P if and only if $x_e = 1$. This allows it to describe the objective function by $\min \sum_{e \in E} x_e c_e$, where c_e is the length of e. In addition, we impose a suitable set of linear constraints on these binary variables, such that they characterize precisely the set of polygons with vertex set V. The challenge is to pick a set of constraints that achieve this in a (relatively) efficient manner.

As it turns out (and is discussed in more detail in Sect. 5), there is a significant set of constraints that correspond to eliminating cycles within proper subsets $S \subset V$. Moreover, there is an exponential number of relevant subsets S, making it prohibitive to impose all of these constraints at once. The fundamental idea of a cutting-plane approach is that much fewer constraints are necessary for characterizing an optimal solution. To this end, only a relatively small subfamily of constraints is initially considered, leading to a relaxation. As long as solving the current relaxation yields a solution that is infeasible for the original problem, violated constraints are added in a piecemeal fashion, i.e., in *iterations*.

In the following, these constraints (which are initially omitted, violated by an optimal solution of the relaxation, then added to eliminate such infeasible solutions) are called *cutting planes* or simply *cuts*, as they remove solutions of a relaxation that are infeasible for the MPP.

4.2 Basic IP

We start with a basic IP that is enhanced with specific cuts, described in Sects. 5.2–5.4. We denote by E the set of all edges between two points of V, $\mathcal{C}$ a set of *invalid cycles* and $\delta(v)$ the set of all edges in E that are incident to $v \in V$. Then we optimize over the following objective function:

$$\min \sum_{e \in E} x_e c_e. \tag{1}$$

This is subject to the following constraints:

$$\forall v \in V : \sum_{e \in \delta(v)} x_e = 2, \tag{2}$$

$$\forall C \in \mathcal{C} : \sum_{e \in C} x_e \leq |C| - 1, \tag{3}$$

$$x_e \in \{0, 1\}. \tag{4}$$

For the TSP, $\mathcal{C}$ is simply the set of *all* subtours, making identification and separation straightforward. This is much harder for the MPP, where a subtour may end up being feasible by forming the boundary of a hole, but may also be required to connect with other cycles. Therefore, identifying valid inequalities requires more geometric analysis, such as the following. If we denote by CH the set of all convex hull points, then a cycle C is invalid if C contains:

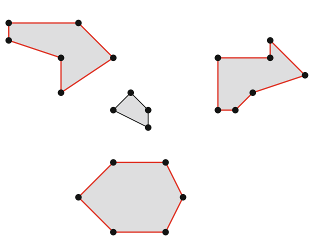
1. at least one and at most $|CH| - 1$ convex hull points. (See Fig. 2(a))
2. all convex hull points but does not enclose all other points. (See Fig. 2(b))
3. no convex hull point but encloses other points. (See Fig. 2(c))

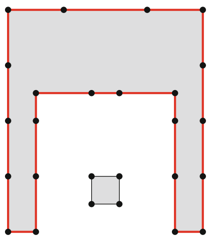
By $\mathcal{C}_i$ we denote the set of all invalid cycles with property i. Because there can be exponentially many invalid cycles, we add constraint (3) in separation steps.

For an invalid cycle with property 1, we use the equivalent cut constraint

$$\forall C \in \mathcal{C}_1 : \quad \sum_{e \in \delta(C)} x_e \geq 2. \tag{5}$$

We use constraint (3) if $|C| \leq \frac{2n+1}{3}$ and constraint (5) otherwise, where $\delta(C)$ denotes the "cut" edges connecting a vertex $v \in C$ with a vertex $v' \notin C$. As argued by Pferschy and Stanek [22], this technique of *dynamic subtour constraints* (DSC) is useful, as it reduces the number of non-zero coefficients in the constraint matrix.


(a) Invalid cycle of type 1


(b) Invalid cycle of type 2

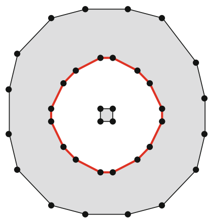

(c) Invalid cycle of type 3

Fig. 2. Examples of invalid cycles (red). Black cycles may be valid. (Color figure online)

4.3 Initial Edge Set

In order to quickly achieve an initial solution, we sparsify the $\Theta(n^2)$ input edges to the $O(n)$ edges of the Delaunay Triangulation, which naturally captures geometric nearest-neighbor properties. If a solution exists, this yields an upper bound. This technique has already been applied for the TSP by Jünger et al. [16]. In theory, this may not yield a feasible solution: a specifically designed example by Dillencourt shows that the Delaunay triangulation may be non-Hamiltonian [10]; this same example has no feasible solution for the MPP when restricted to Delaunay edges. We did not observe this behavior in practice.

CPLEX uses this initial solution as an upper bound, allowing it to quickly discard large solutions in a branch-and-bound manner. As described in Sect. 6, the resulting bounds are quite good for the MPP.

5 Separation Techniques

5.1 Pitfalls

When separating infeasible cycles, the Basic IP may get stuck in an exponential number of iterations, due to the following issues. (See Figs. 3–5 for illustrating examples.)

Problem 1: Multiple outer components containing convex hull points occur that (despite the powerful subtour constraints) do not get connected, because it is cheaper to, e.g., integrate subsets of the interior points. Such an instance can be seen in Fig. 3, where we have two equal components with holes. Since the two components are separated by a distance greater than the distance between their outer components and their interior points, the outer components start to include point subsets of the holes. This results in a potentially exponential number of iterations.

Problem 2: Outer components that do not contain convex hull points do not get integrated, because we are only allowed to apply a cycle cut on the outer component containing the convex hull points. An outer component that does not contain a convex hull point cannot be prohibited, as it may become a hole in later iterations. See Fig. 4 for an example in which an exponential number of iterations is needed until the outer components get connected.

Problem 3: If holes contain further holes, we are only allowed to apply a cycle cut on the outer hole. This outer hole can often cheaply be modified to fulfill the cycle cut but not resolve the holes in the hole. An example instance can be seen in Fig. 5, in which an exponential number of iterations is needed.

The second problem is the most important, as this problem frequently becomes critical on instances of size 100 and above. Holes in holes rarely occur on small instances but are problematic on instances of size >200. The first problem occurs only in a few instances.

In the following we describe three cuts that each solve one of the problems: The glue cut for the first problem in Sect. 5.2, the tail cut for the second problem in Sect. 5.3, and the HiH-Cut for the third problem in Sect. 5.4.

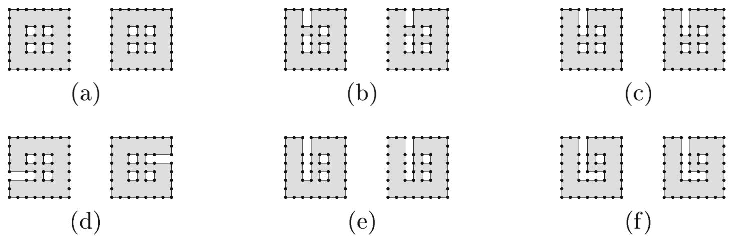


Fig. 3. (a)–(f) show consecutive iterations when trying to solve an instance using only constraint (5).

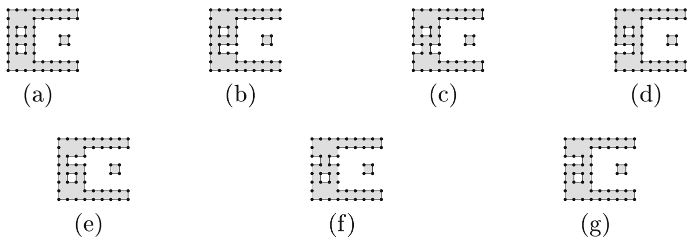


Fig. 4. (a)–(g) show consecutive iterations when trying to solve an instance using only constraint (3).

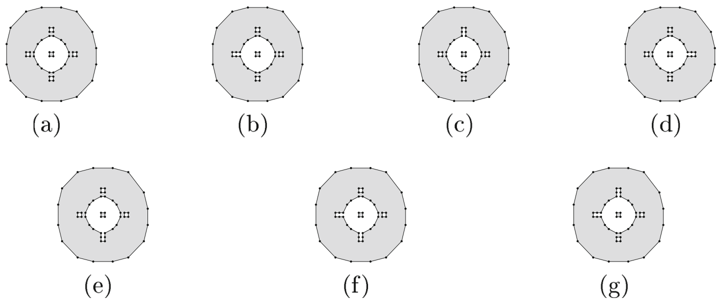


Fig. 5. (a)–(g) show consecutive iterations when trying to solve an instance using only constraint (3).

5.2 Glue Cuts

To separate invalid cycles of property 1 we use *glue cuts* (GC), based on a curve R_D from one unused convex hull edge to another (see Fig. 6). With $\mathcal{X}(R_D)$ denoting the set of edges crossing R_D, we can add the following constraint:

$$\sum_{e \in \mathcal{X}(R_D)} x_e \geq 2.$$

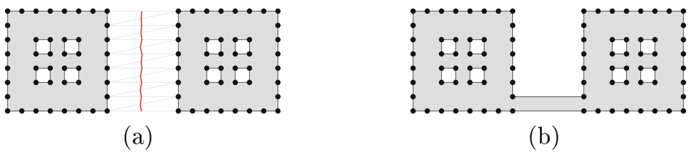


Fig. 6. Solving instance from Fig. 3 with a glue cut (red). (a) The red curve needs to be crossed at least twice; it is found using the Delaunay Triangulation (grey). (b) The first iteration after using the glue cut. (Color figure online)

Such curves can be found by considering a constrained Delaunay triangulation [5] of the current solution, performing a breadth-first-search starting from all unused convex hull edges of the triangulation. Two edges are adjacent if they share a triangle. Used edges are excluded, so our curve will not cross any used edge. As soon as two different search trees meet, we obtain a valid curve by using the middle points of the edges (see the red curve in Fig. 6).

For an example, see Fig. 6; as illustrated in Fig. 3, this instance is problematic in the Basic IP. This can we now be solved in one iteration.

5.3 Tail Cuts

An outer cycle C that does not contain any convex hull points cannot simply be excluded, as it may become a legal hole later. Such a cycle either has to be merged with others, or become a hole. For a hole, each curve from the hole to a point outside of the convex hull must be crossed at least once.

With this knowledge we can provide the following constraint, making use of a special curve, which we call a *tail* (see the red path in Fig. 7).

Let R_T be a valid tail and $\mathcal{X}(R_T)$ the edges crossing it. We can express the constraint in the following form:

$$\underbrace{\sum_{e\in\mathcal{X}(R_T)\setminus\delta(C)} x_e}_{\text{C gets surrounded}} + \underbrace{\sum_{e\in\delta(C)} x_e}_{\text{C merged}} \geq 1.$$

The tail is obtained in a similar fashion as the curves of the *Glue Cuts* by building a constrained Delaunay triangulation and doing a breadth-first search starting at the edges of the cycle. The starting points are not considered as part of the curve and thus the curve does not cross any edges of the current solution.

For an example, see Fig. 7; as illustrated in Fig. 4, this instance is problematic in the Basic IP. This can we now be solved in one iteration. Note that even though it is possible to cross the tail without making the cycle a hole, this is more expensive than simply merging it with other cycles.

5.4 Hole-in-Hole Cuts

The difficulty of eliminating holes in holes (Problem 3) is that they may end up as perfectly legal simple holes, if the outer cycle gets merged with the outer

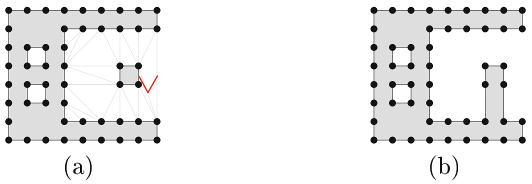


Fig. 7. Solving the instance from Fig. 4 with a tail cut (red line). (a) The red curve needs to be crossed at least twice or two edges must leave the component. The red curve is found via the Delaunay Triangulation (grey). (b) The first iteration after using the tail cut. (Color figure online)

boundary. In that case, every curve from the hole to the convex hull *cannot* cross the used edges exactly two times (edges of the hole are ignored). One of the crossed edges has to be of the exterior cycle, while the other one cannot: otherwise would again leave the polygon. It also cannot be of an interior cycle, as it would have leave to leave that cycle again to reach the hole.

Therefore the inner cycle of a hole in hole either has to be merged, or all curves from it to the convex hull do not have exactly two used edge crossings. As it is impractical to argue over all curves, we only pick one curve P that currently crosses exactly two used edges (see the red curve in Fig. 8 with crossed edges in green).

Because we cannot express the inequality that P is not allowed to be crossed exactly two times as an linear programming constraint, we use the following weaker observation. If the cycle of the hole in hole becomes a simple hole, the crossing of P has to change. Let e_1 and e_2 be the two used edges that currently cross P and $\mathcal{X}(P)$ the set of all edges crossing P (including unused but no edges of H). We can express a change on P by

$$\underbrace{\sum_{e\in\mathcal{X}(P)\setminus\{e_1,e_2\}} x_e}_{\text{new crossing}} + \underbrace{(-x_{e_1} - x_{e_2})}_{e_1 \text{ or } e_2 \text{ vanishes}} \geq -1.$$

Together we obtain the following LP constraint for either H being merged or the crossing of P changing.

$$\underbrace{\sum_{e\in\delta(V_H, V\setminus V_H)} x_e}_{H \text{ merged}} + \underbrace{\sum_{e\in\mathcal{X}(P)\setminus\{e_1,e_2\}} x_e + (-x_{e_1} - x_{e_2})}_{\text{Crossing of } P \text{ changes}} \geq -1.$$

Again we use a breadth-first search on the constrained Delaunay triangulation starting from the edges of the hole in hole. Unlike the other two cuts we need to cross used edges. Thus, we get a shortest path search such that the optimal path primarily has a minimal number of used edges crossed and secondarily has a minimal number of all edges crossed.

For an example, see Fig. 8; as illustrated in Fig. 3, this instance is problematic in the Basic IP. This can now be solved in one iteration. The corresponding path

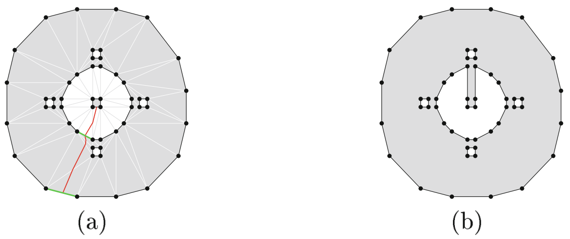


Fig. 8. Solving instance from Fig. 5 with hole in hole cut (red line). (a) The red line needs to be crossed at least two times or two edges must leave the component or one of the two existing edges (green) must be removed. The red line is built via Delaunay Triangulation. (b) The first iteration after using the hole in hole cut. (Color figure online)

is displayed in red and the two crossed edges are highlighted in green. Changing the crossing of the path is more expensive than simply connecting the hole in hole to the outer hole and thus the hole in hole gets merged.

6 Experiments

6.1 Implementation

Our implementation uses CPLEX to solve the relevant IPs. Important is also the geometric side of computation, for which we used the CGAL Arrangements package [24]. CGAL represents a planar subdivision using a doubly connected edge list (DCEL), which is ideal for detecting invalid boundary cycles.

6.2 Test Instances

While the TSPLIB is well recognized and offers a good mix of instances with different structure (ranging from grid-like instances over relatively uniform random distribution to highly clustered instances), it is rather sparse. Observing that the larger TSPLIB instances are all geographic in nature, we designed a generic approach that yields arbitrarily large and numerous clustered instances. This is based on illumination maps: A satellite image of a geographic region at night time displays uneven light distribution. The corresponding brightness values can be used as a random density function that can be used for sampling (see Fig. 12). To reduce noise, we cut off brightness values below a certain threshold, i.e., we set the probability of choosing the respective pixels to zero.

6.3 Results

All experiments were run on an *Intel Core i7-4770* CPU clocked at 3.40 GHz with 16 GB of RAM. We set a 30 min time limit to solve the instances. In Table 1, all results are displayed for every instance with more than 100 points that we

Table 1. The runtime in milliseconds of all variants on the instances of the TSPLib with more than 100 points, solved within 30 min. The number in the name of an instance indicates the number of points.

	BasicIP	+JS+DC+TC+ HIHC	+JS+TC+ HIHC	+JS+DC+ HIHC	+JS+DC+ TC	+DC+TC+ HIHC
eil101	-	575	445	-	527	1090
lin105	-	390	359	-	412	931
pr107	550	401	272	346	513	923
pr124	495	348	264	322	355	940
bier127	439	288	270	267	276	476
ch130	-	1758	1802	-	1594	2853
pr136	1505	964	1029	992	950	3001
gr137	-	1262	1361	-	1252	1724
pr144	6276	1028	2926	985	1030	2012
ch150	-	4938	5167	-	5867	7997
kroA150	-	3427	5615	-	3327	7474
kroB150	-	2993	2396	-	2943	5265
pr152	13285	2161	1619	10978	2151	19479
u159	13285	1424	1262	5339	1410	2513
rat195	106030	16188	19780	77216	16117	27580
d198	-	19329	155550	-	19398	41118
kroA200	-	26360	13093	-	26389	11844
kroB200	-	5492	6239	-	5525	15238
gr202	-	4975	7512	-	4304	9670
ts225	18902	7746	9750	7595	7603	60167
tsp225	91423	11600	9741	28756	11531	44297
pr226	-	8498	2800	-	7204	18848
gr229	-	5462	26478	-	10153	25674
gil262	-	23000	22146	-	-	72772
pr264	24690	6537	-	6719	6549	23641
a280	22023	3601	3857	3980	3619	12983
pr299	-	16251	355323	-	16173	85789
lin318	-	23863	1511219	-	24035	75312
linhp318	-	23107	1313680	-	23064	79352
rd400	-	111128	92995	-		302363
fl417	-	198013	-	-	215210	825808
gr431	-	56716	173609	-	78133	265416
pr439	-	46685	36592	-	48231	273873
pcb442	-	1356796	-	-	-	-
d493	-	359072	-	-	-	837229
att532	-	217679	256394	-	218665	817096
ali535	-	93771	427800	-	91828	323104
u574	-	371523	199114	-	-	1010276
rat575	-	417494	191198	-	580320	934988
p654	-	864066	-	-	-	-
d657	-	455378	253374	-	646148	1352747
gr666	-	366157	-	-	670818	-

solved within the time limit. The largest instance solved within 30 min is gr666 with 666 points, which took about 6 min. The largest instance solved out of the TSPLib so far is dsj1000 with 1000 points, solved in about 37 min. In addition, we generated 30 instances for each size, which were run with a time limit of 30 min.

We observe that even without using glue cuts and jumpstart, we are able to solve more than 50 % of the instances up to about 550 input points. Without the tail cuts, we hit a wall at 100 points, without the HiH-cut instances, at about 370 input points; see Fig. 9, which also shows the average runtime of all 30 instances for all variants. Instances exceeding the 30 min time limit are marked with a 30-minutes timestamp. The figure shows that using jumpstart shortens the runtime significantly; using the glue cut is almost as fast as the variant without the glue cut.

Figure 10 shows that medium-sized instances (up to about 450 points) can be solved in under 5 min. We also show that restricting the edge set to the Delaunay triangulation edges yields solutions that are about 0.5 % worse on average than the optimal solution. Generally the solution of the jumpstart gets very close to the optimal solution until about 530 points. After that, for some larger instances,

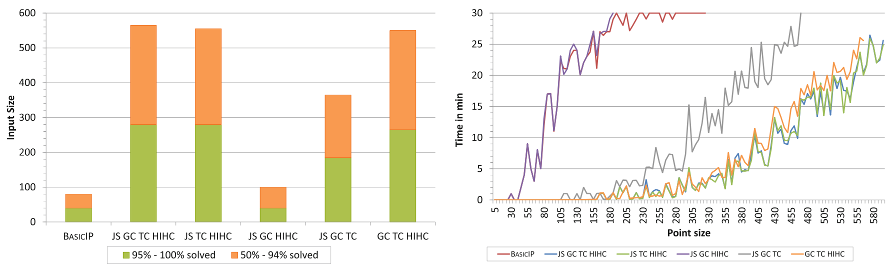


Fig. 9. (Left) Success rate for the different variants of using of the cuts, with 30 instances for each input size (y-axis). (Right) The average runtime of the different variants for all 30 instances. A non-solved instance is interpreted as 30 min runtime.

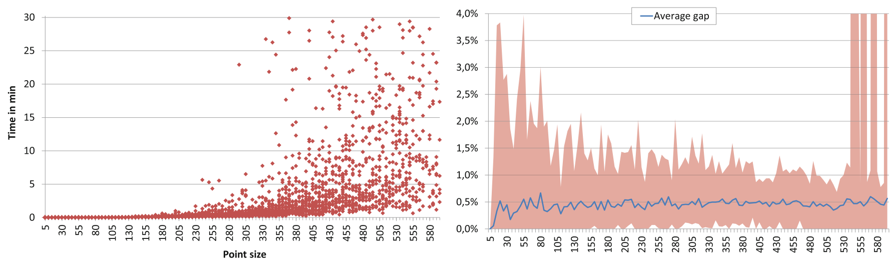


Fig. 10. (Left) The distribution of the runtime within 30 min for the case of using the jumpstart, glue cuts, tail cuts and HiH-cuts. (Right) The relative gap of the value on the edges of the Delaunay triangulation to the optimal value. The red area marks the range between the minimal and maximal gap.

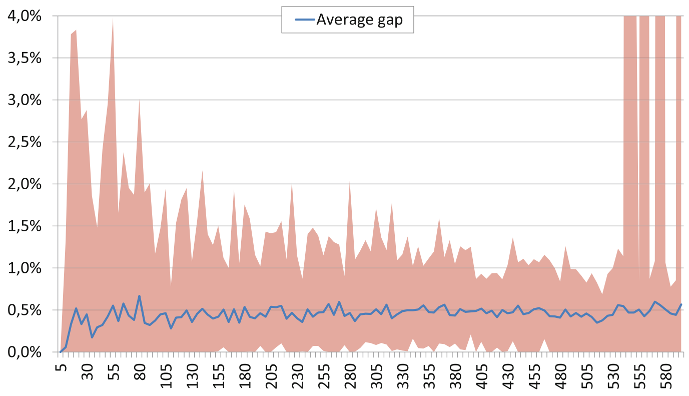
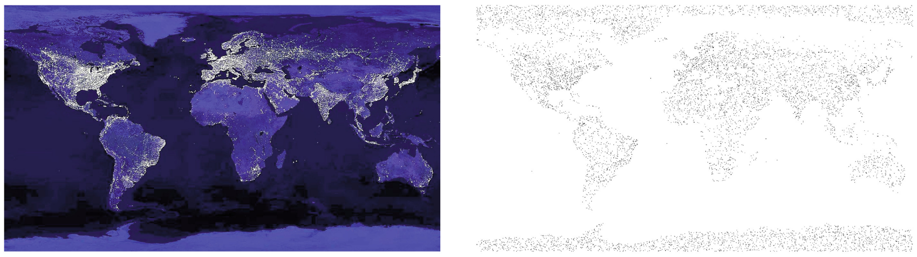


Fig. 11. The relative gap of the value on the edges of the Delaunay triangulation to the optimal value. The red area marks the range between the minimal and maximal gap. (Color figure online)



(a) Earth by night (b) A sampled instance

Fig. 12. Using a brightness map as a density function for generating clustered point sets.

we get solutions on the edge set of the Delaunay triangulation that are up to 50 % worse than the optimal solution.

7 Conclusions

As discussed in the introduction, considering general instead of simple polygons corresponds to searching for a shortest cycle cover with a specific topological constraint: one outside cycle surrounds a set of disjoint and unnested inner cycles. Clearly, this is only one example of considering specific topological constraints. Our techniques and results should be applicable, after suitable adjustments, to other constraints on the topology of cycles. We gave a 3-approximation for the MPP; we expect that the MPP has a polynomial-time approximation scheme, base on PTAS techniques [4, 20] for geometric TSP, and we will elaborate on this in a future version of the full paper.

There are also various practical aspects that can be explored further. It will be interesting to evaluate the practical performance of the theoretical approximation algorithm, not only from a practical perspective, but also to gain some insight on whether the approximation factor of 3 can be tightened. Pushing the limits of solvability can also be attempted, e.g., by using more advanced techniques from the TSP context. We can also consider sparsification techniques other than the Delaunay edges; e.g., the union between the best known tour and the k-nearest-neighbor edge set ($k \in \{2, 5, 10, 20\}$) has been applied for TSP by Land [17], or (see Padberg and Rinaldi [21]) by taking the union of k tours acquired by Lin's and Kernighan's heuristic algorithm [19].

Acknowledgements. We thank Stephan Friedrichs and Melanie Papenberg for helpful conversations. Parts of this work were carried out at the 30th Bellairs Winter Workshop on Computational Geometry (Barbados) in 2015. We thank the workshop participants and organizers, particularly Erik Demaine. Joseph Mitchell is partially supported by NSF (CCF-1526406). Irina Kostitsyna is supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 639.023.208.

References

1. Althaus, E., Mehlhorn, K.: Traveling salesman-based curve reconstruction in polynomial time. SIAM J. Comput. **31**(1), 27–66 (2001)
2. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: On the solution of traveling salesman problems. Documenta Mathematica – Journal der DeutschenMathematiker-Vereinigung, ICM, pp. 645–656 (1998)
3. Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton Series in Applied Mathematics. Princeton University Press, Princeton (2007)
4. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM **45**(5), 753–782 (1998)
5. Chew, L.P.: Constrained Delaunay triangulations. Algorithmica **4**(1–4), 97–108 (1989)
6. Christofides, N.: Worst-case analysis of a new heuristic for the Travelling Salesman Problem, Technical report 388, Graduate School of Industrial Administration, CMU (1976)
7. Cook, W.J.: In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation. Princeton University Press, Princeton (2012)
8. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley, New York (1998)
9. Dey, T.K., Mehlhorn, K., Ramos, E.A.: Curve reconstruction: connecting dots with good reason. Comput. Geom. **15**(4), 229–244 (2000)
10. Dillencourt, M.B.: A non-Hamiltonian, nondegenerate Delaunay triangulation. Inf. Process. Lett. **25**(3), 149–151 (1987)
11. Fekete, S.P., Friedrichs, S., Hemmer, M., Papenberg, M., Schmidt, A., Troegel, J.: Area- and boundary-optimal polygonalization of planar point sets. In: EuroCG 2015, pp. 133–136 (2015)
12. Fekete, S.P., Haas, A., Hemmer, M., Hoffmann, M., Kostitsyna, I., Krupke, D., Maurer, F., Mitchell, J.S.B., Schmidt, A., Schmidt, C., Troegel, J.: Computing nonsimple polygons of minimum perimeter. CoRR, abs/1603.07077 (2016)

13. Giesen, J.: Curve reconstruction, the traveling salesman problem and Menger's theorem on length. In: Proceedings of 15th Annual Symposium on Computational Geometry (SoCG), pp. 207–216 (1999)
14. Grötschel, M.: On the symmetric travelling salesman problem: solution of a 120-city problem. Math. Program. Study **12**, 61–77 (1980)
15. Gutin, G., Punnen, A.P.: The Traveling Salesman Problem and Its Variations. Springer, New York (2007)
16. Jünger, M., Reinelt, G., Rinaldi, G.: The traveling salesman problem. In: Handbooks in Operations Research and Management Science, vol. 7, pp. 225–330 (1995)
17. Land, A.: The solution of some 100-city Travelling Salesman Problems, Technical report, London School of Economics (1979)
18. Lawler, E.L., Lenstra, J.K., Rinnooy-Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley, Chichester (1985)
19. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**(2), 498–516 (1973)
20. Mitchell, J.S.B.: 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)
21. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Rev. **33**(1), 60–100 (1991)
22. Pferschy, U., Stanek, R.: Generating subtour constraints for the TSP from pure integer solutions. Department of Statistics and Operations Research, University of Graz, Technical report (2014)
23. Reinelt, G.: TSPlib - a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
24. Wein, R., Berberich, E., Fogel, E., Halperin, D., Hemmer, M., Salzman, O., Zukerman, B.: 2D arrangements. In: CGAL User and Reference Manual, 4.3rd edn. CGAL Editorial Board (2014)