Constructing Concise Convex Covers via Clique Covers

Mikkel Abrahamsen
University of Copenhagen, Denmark

William Bille Meyling
University of Copenhagen, Denmark

André Nusser
Université Côte d'Azur, CNRS, Inria, I3S, France

Abstract

This work is the full version describing the winning implementation of the CG:SHOP 2023 Challenge. The topic of the Challenge was the convex cover problem: given a polygon P (with holes), find a minimum-cardinality set of convex polygons whose union equals P. We use a three-step approach: (1) Create a suitable triangulation of P. (2) Compute a visibility graph of the triangles. (3) Solve a vertex clique cover problem on the visibility graph, from which we then derive the convex cover. This way we capture the geometric difficulty in the first step and the combinatorial difficulty in the third step.

Keywords and phrases Convex cover, Polygons with holes, Algorithm engineering, Vertex clique cover

Digital Object Identifier 10.57717/cgt.v5i4.76

Supplementary Material `https://github.com/willthbill/ExtensionCC`

Funding *Mikkel Abrahamsen*: Supported by Starting Grant 1054-00032B from the Independent Research Fund Denmark under the Sapere Aude research career programme. Part of BARC, supported by the VILLUM Foundation grant 16582.
William Bille Meyling: Supported by Starting Grant 1054-00032B from the Independent Research Fund Denmark under the Sapere Aude research career programme.
André Nusser: Part of this work was conducted at BARC, supported by the VILLUM Foundation grant 16582.

Acknowledgements We want to thank the CG:SHOP 2023 organizers and the other participants (especially Guilherme Dias da Fonseca) for creating such a fun challenge and for helpful comments on our write-up. We also want to thank Martin Aumüller and Rasmus Pagh for access and help with using their server. Finally, we want to thank Darren Strash for quick and last-minute support using the ReduVCC implementation.

1 Introduction

Covering a polygon with the minimum number of convex pieces is a fundamental problem in computational geometry and the problem chosen for the CG:SHOP 2023 Challenge [7]. In this problem we are given a polygon P (potentially with holes) and we have to find a smallest possible set of convex polygons whose union equals P. This problem is NP-hard [5] and was later shown to be even $\exists\mathbb{R}$-complete [1]. Note that in the Challenge, all coordinates of the solutions had to be rational, and then the decision problem is not even known to be decidable. Thus, it is not expected that there exists any fast algorithm that always finds the optimal solution. Likewise, we are not aware of any previously described algorithm which is fast in practice. In this 5th CG:SHOP Challenge, there were a total of 22 teams who signed up, out of which 18 submitted solutions. Our team — named *DIKU (AMW)* — obtained

the highest total score, despite finding fewer smallest solutions than the runner-up [6], as we achieved significantly smaller solutions on many of the largest Challenge instances. See [7] for a survey about the Challenge.

Our algorithm consists of three steps. In step (1), the aim is to capture the geometry of the problem. We do this by triangulating the input polygon P. Note that the corners of these triangles do not have to be corners of P, i.e., they can be Steiner points. In step (2), we move from the geometric structure to a combinatorial structure. We do this by computing a visibility graph G of the triangulation, with each triangle corresponding to a vertex and an edge is inserted for two vertices only if the convex hull of the corresponding triangles lies within P (i.e., the convex hull would be a valid piece of the convex cover). Finally, in step (3), we solve a combinatorial problem: we find a vertex clique cover (VCC) of G with small cardinality. Recall that a *vertex clique cover* is a set of cliques in G, such that each vertex in G appears in one of the cliques. When possible, we use the convex hull H of the triangles of a clique C as a piece of our convex cover. However, H may intersect holes of P, which makes H an invalid piece. This happens rarely for the Challenge instances, but in that case we split C into smaller cliques.

The steps (1)–(3) are described in more detail in Sections 2.1–2.3 below. In Section 3, we describe some of the algorithmic ideas used to make polygon containment tests fast in practice, which is used in both steps (2) and (3). Finally, in Section 4, we give examples of the resulting covers and report on experiments.

We believe that the main insights and highlights of our approach are:

1. Assembling the pieces and the cover at the same time (instead of first deciding on the pieces and then assembling the cover) allows for great flexibility and adaptivity.
2. Reduction to a fundamental graph problem allows for usage of a powerful set of already existing tools.
3. As we can arbitrarily choose a partition in the first step of the algorithm, our approach is very adaptive with respect to input structure and instance size (e.g., simpler partitions can be chosen for larger instances).

Since the preliminary version of this paper was published [3], a similar approach has been applied to approximating robot configuration spaces with few convex sets [13]. Such a collection of convex sets covering nearly all of the free space can dramatically accelerate many computations in robotics, such as finding shortest paths. See also [2], a survey that suggests variants of the convex cover problem that may be relevant in robotics.

2 Algorithm

In this section we describe our algorithmic approach to solve the convex cover problem. There is a subsection for each of the three main steps of our approach: compute a partition of the polygon, compute a visibility graph of the parts, and then create a convex cover from the visibility graph.

2.1 Step 1: triangulation

First, we triangulate the input polygon. While our approach in principle works with any kind of partition, for simplicity we only use triangulations. Recall that the goal is to obtain triangles from which we can later assemble good pieces for a convex cover, and that the corners of these triangles are not restricted to lie on the corners of P. In fact, to obtain good solutions one often needs Steiner points.

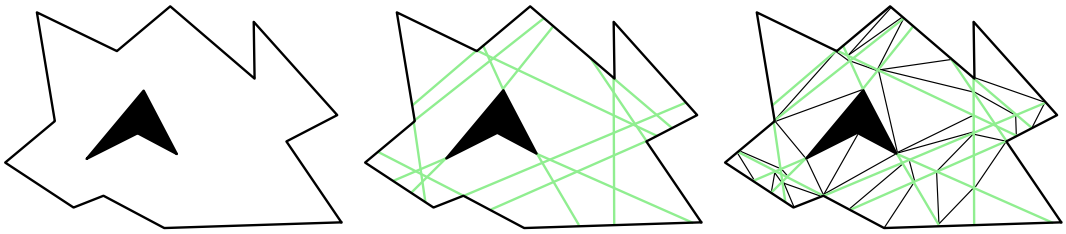

Figure 1 Polygon P (left), extensions of P (middle), and extension partition of P (right).

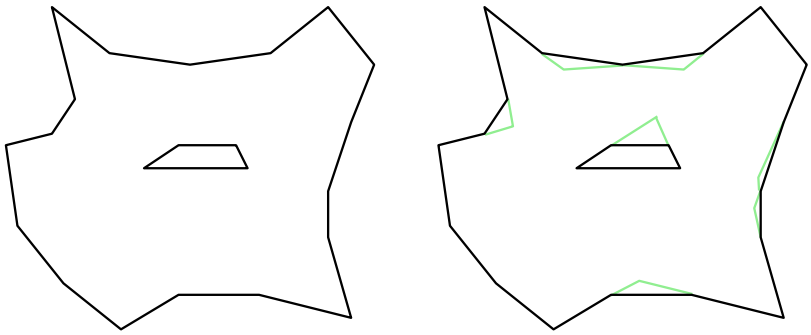

Figure 2 Polygon P (left) and `concave-chain triangulation` with no grid (right).

In the simplest case, we use a Delaunay triangulation (constrained by the edges of the polygon). We prefer a Delaunay triangulation over an arbitrary triangulation because it leads to fat triangles, which we intuitively assume to create better pieces for the convex cover. The main issues of using a Delaunay triangulation of P as partition are that its vertices are restricted to the corners of P and that the pieces can be too coarse to merge into valid convex pieces; see Figure 10 for an example for which this leads to a suboptimal cover. Thus, the question is: which Steiner points should we introduce to obtain better solutions?

Consider a directed edge e of P and suppose that the interior of P is to the left of e. We define the *extension* of e to be the maximal directed segment s such that $e \subseteq s \subseteq P$; see Figure 1 (middle). Note that a piece Q of a convex cover can contain an interior point of e only if Q does not contain a point to the right of s. Thus, intuitively it makes sense to include pieces of the cover that are bounded by s. This intuition is captured by the *extension partition*, which is the constrained Delaunay triangulation of the extensions of all edges of P; see Figure 1 (right).

Unfortunately, the extension partition can have quadratic size in the number of vertices of P. This renders an extension partition infeasible for some large instances. To reduce the size of the extension partition in these cases, we try to restrict to important extensions. As described above, we add the extension of an edge e to facilitate the creation of a piece that covers the interior of e. This is a local property and we hence prefer shorter extensions as they capture the local structure but hopefully lead to fewer intersections. We did so in two ways: (i) given a length threshold, we only include the extensions that have length below this threshold, and (ii) pick extensions with probabilities that decrease with the length of an extension.

Finally, consider the case of long concave chains in polygons, i.e., a sequence of edges that have concave inner angles; see Figure 11 for an example. Note that if we have a concave chain of k edges, then we also need at least k pieces to cover it as no interior of two edges can be covered by the same convex piece (assuming that the concave chain does not turn so

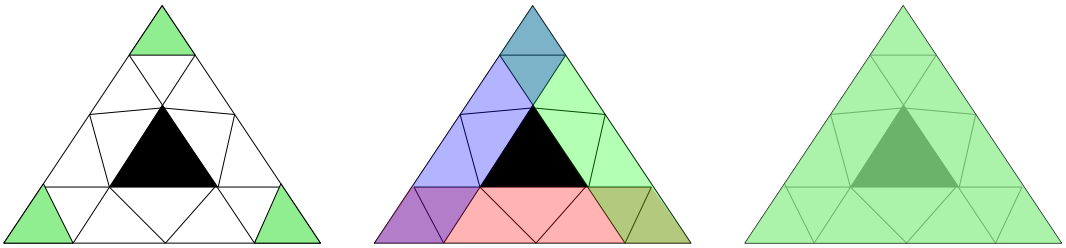

Figure 3 For the set of green triangles (left), every pairwise convex hull is contained in P (middle), but the convex hull of all the green triangles is not (right).

much that some edges face each other). To create a triangulation that captures this property, we do the following: we add a small triangle to each edge of the concave chain, which is formed by the extensions of its neighboring edges, see Figure 2. This in particular enables our algorithm to later create pieces that cover edges of multiple concave chains, see Figure 11.

2.2 Step 2: visibility graph

In order to create a convex cover, we first want to determine which triangles we can potentially combine to form pieces for the cover. Given a triangulation $\mathcal{T}$ of the polygon P and two triangles $p, q \in \mathcal{T}$, we say that q is *fully visible* from p if every point in p sees all of q, in other words, the convex hull of $p \cup q$ is contained in P. We say that q is *partially visible* if every point in p sees some point in q (but not necessarily the same). We define the visibility graph $G = (\mathcal{T}, E)$, which contains an edge pq if q is fully visible from p. We can compute G naively by checking for each pair $p, q \in \mathcal{T}$ whether its convex hull is contained in P. However, the running time $\Omega(|\mathcal{T}|^2)$ renders this impractical. A simple observation comes in handy here: For any triangle $q \in \mathcal{T}$ fully visible from $p \in \mathcal{T}$, there exists a path from p to q in the dual graph[1] of $\mathcal{T}$ using only triangles that are partially visible from p. Thus, instead of checking all pairwise visibilities, we can simply perform a BFS on the dual graph from each of the triangles. When doing BFS from a triangle p, we only add neighbours of triangles that are partially visible from p to the BFS queue, and we stop exploring around triangles that are not even partially visible. While this significantly reduces the running time in practice, it can still be too expensive. For further speedup, we resort to building a subgraph of G by only adding neighbour triangles that are fully visible from p to the BFS queue. The region we explore will thus be bounded by triangles that are not fully visible. To speed up the visibility graph construction, we engineered fast visibility checks to be described in Section 3.

2.3 Step 3: compute cover

We employ the following three steps to compute a convex cover using the visibility graph.

Compute vertex clique cover

We first compute a vertex clique cover (VCC) on the visibility graph. The problem of finding a minimum VCC is one of Karp's classical NP-hard problems, and there exists no $n^{1-\varepsilon}$-approximation algorithm for any $\varepsilon > 0$ unless P = NP. However, there exist

[1] The dual graph of a partition is defined as follows: the vertex set consists of the triangles of the partition and there is an edge between two vertices iff the two corresponding triangles touch.

implementations that compute small solutions on practical instances. Namely, Chalupa [4] presented a randomized clique-growing approach that was subsequently used as a subroutine by Strash and Thompson [11] in their state-of-the-art solver `ReduVCC` that uses sophisticated reduction rules with a branch-and-reduce approach.

Fix cover

Recall that a clique C corresponds to a set of triangles that are pairwise fully visible. We would like to use the convex hull Q of the triangles as a piece in our convex cover, but Q may not be contained in P; see Figure 3 for a simple example. While this rarely happens on the Challenge instances (see Section 4.3), we nonetheless have to post-process such pieces to obtain a feasible convex cover.

Let us first give the following lemma which characterizes the situation where Q is not contained in P.

▶ **Lemma 1.** *Let Q be the convex hull of the triangles corresponding to a clique C. Then Q is disjoint from the unbounded connected component of the complement of P, and if Q intersects a hole of P, then the hole is contained in Q.*

Proof. Since Q is the convex hull of triangles for which each pair has full visibility, the boundary of Q is contained in P. Hence, any connected component in the complement of P that intersects Q is also contained in Q. It follows that the unbounded region of the complement of P cannot intersect Q, and if a hole intersects Q, then the hole is contained in Q. ◀

We fix an invalid piece Q as follows: Pick any hole H that invalidates Q and consider an arbitrary line ℓ intersecting H. Let L be one of the two half-planes bounded by ℓ. Now partition the triangles as $C = C_1 \cup C_2$, where triangles in C_1 intersect L and triangles in C_2 are disjoint from L. Taking the convex hulls of C_1 and C_2, we get two new pieces Q_1 and Q_2, and we claim that both are disjoint from H. We apply this procedure recursively to Q_1 and Q_2 (always reducing the number of holes intersected by at least one) until all newly created pieces are valid; see Figure 4. The following lemma states that the described procedure indeed eliminates the intersections with the hole H.

▶ **Lemma 2.** *Following the process described above, the new pieces Q_1 and Q_2 are both disjoint from the hole H.*

Proof. Suppose for contradiction that Q_i intersects H for some $i \in \{1, 2\}$. By Lemma 1, if Q_i intersects H, then Q_i contains H. The polygon Q_2 is disjoint from the half-plane L and H intersects L, so Q_2 cannot contain H. To see that Q_1 cannot contain H, note that the connected components of the complement $\ell \setminus H$ contains two unbounded rays ℓ_1 and ℓ_2, one on each side of H. There must be triangles Δ_1 and Δ_2 in C_1 such that Δ_i intersects ℓ_i, since otherwise, Q_1 would not contain H. It then follows that there is no full visibility between Δ_1 and Δ_2, as the line ℓ between them is obstructed by H. This is a contradiction. ◀

Make cover minimal

At this point, we may end up with a non-minimal cover, i.e., there may exist redundant pieces; see Figure 5. To make the solution minimal, we iterate over the pieces and remove them from the cover if their removal does not invalidate it.

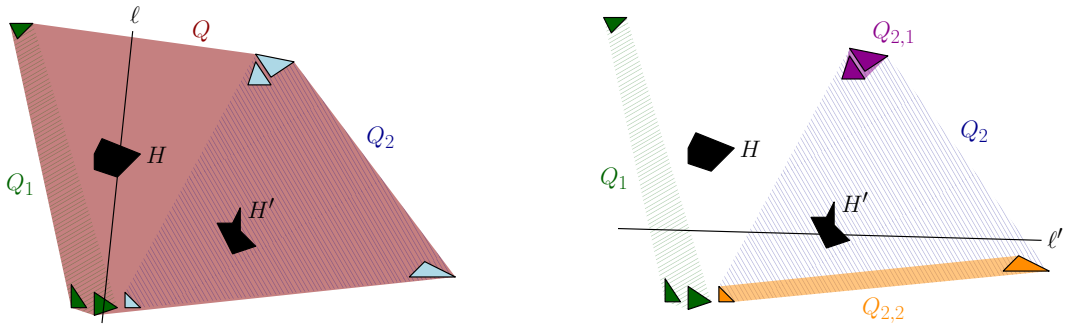

Figure 4 The shown triangles form a clique, but the resulting piece Q is invalid due to holes H and H'. We eliminate the holes by splitting Q into Q_1 and Q_2 (left), and then splitting Q_2 into $Q_{2,1}$ and $Q_{2,2}$ (right), using lines ℓ and ℓ' through H and H', respectively.

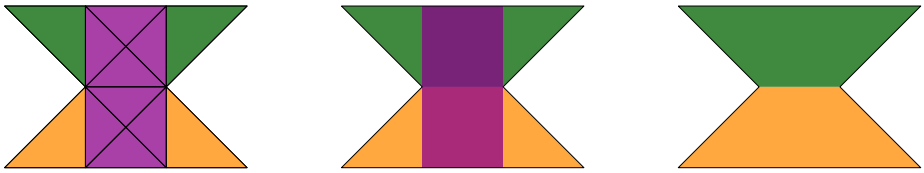

Figure 5 Extension partition with suboptimal clique cover (left), the corresponding convex cover (middle), and a convex cover without the unnecessary purple polygon (right).

3 Fast polygon containment

One of the most frequent sub-problems that we have to solve in our approach is to decide whether a given convex polygon Q is contained in the input polygon P, or in the visibility polygon of a point in P. We need to answer this question in both step (2) and (3): In step (2), we need to decide whether the convex hull of two triangles is contained in P. In step (3), we need to check whether the convex hull of the triangles forming a clique is contained in P, and if not, we recursively split and check for containment as described in Section 2.3. We employ various methods for such containment queries, which we explain in the following subsections.

3.1 Containment in constant time

When constructing the visibility graph, we can often decide if an edge is present or not in constant time using the following lemma, which is illustrated in Figure 6.

▶ **Lemma 3.** *Consider two triangles T_1 and T_2 of a triangulation of P. Let E be the set of one, two or three edges of T_2 that are not on the boundary of the convex hull of $T_1 \cup T_2$. If along each edge $e \in E$ there is a neighbouring triangle of T_2 which is fully visible from T_1, then T_2 is also fully visible from T_1. If an edge in E is on the boundary of P, then T_2 is not fully visible from T_1.*

Proof. Suppose that along each edge in E, there is a neighbouring triangle which is fully visible from T_1. Consider points $p \in T_1$ and $q \in T_2$. The segment pq can be split into a segment connecting T_1 with a neighbour T of T_2, and a segment contained in $T \cup T_2$. Both of these are contained in P because T_1 fully sees T.

If, on the other hand, an edge $e \in E$ is on the boundary of P, then there are points $p \in T_1$ and $q \in T_2$ such that the segment pq crosses e in the interior, and thus part of pq is

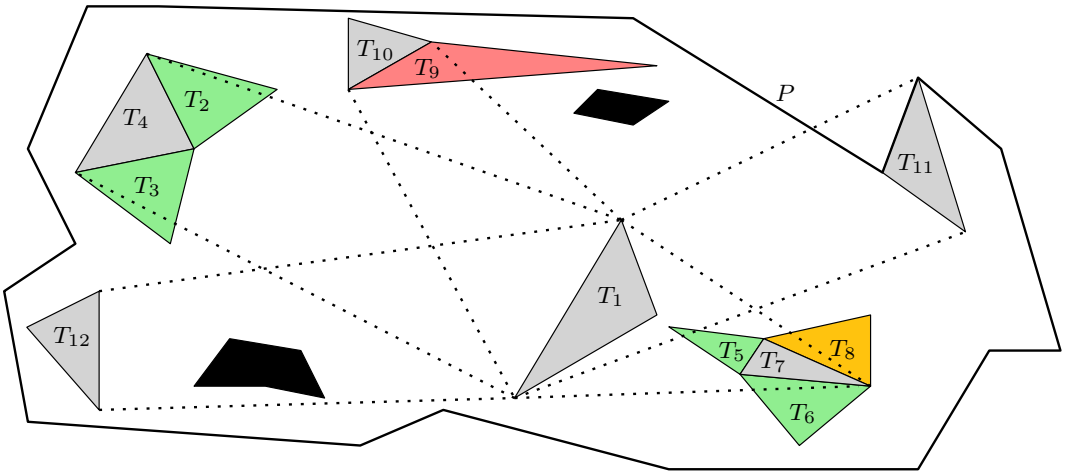

Figure 6 A polygon P with two holes (black) and some triangles from a triangulation shown. Green triangles are known to be fully visible from T_1, red ones are known to be not fully visible from T_1 and for orange ones it is unknown. We conclude that T_4 is fully visible, as T_2 and T_3 are. We cannot conclude right now that T_7 is fully visible as T_8 is unknown. We cannot conclude that T_{10} is fully visible since T_9 is not. We conclude that T_{11} is not fully visible since an edge not on the boundary of the convex hull is on the boundary of P. We cannot conclude that T_{12} is *not* fully visible, since the one edge not on the boundary of the convex hull is not on the boundary of P.

not in P, so T_2 is not fully visible from T_1. ◀

We use the lemma to construct many edges in the visibility graph from a triangle T_1. As explained in Section 2.2, the order in which we consider the other triangles is a BFS order of the dual graph of the triangulation starting from T_1. A BFS order usually has the property that if the relevant neighbours of a triangle T_2 are fully visible, then these neighbours are considered before T_2. Our algorithm can then conclude that T_2 is fully visible.

3.2 Containment in visibility polygons

When we are unable to use the constant-time check from Section 3.1 to conclude whether two triangles are fully visible from each other, we resort to the following simple lemma instead.

▶ **Lemma 4.** *Let Q_1 and Q_2 be two interior-disjoint convex polygons contained in P. Let L_a and L_b be the two outer common tangents of Q_1 and Q_2. For $i \in \{1,2\}$ and $c \in \{a,b\}$, let p_{ic} be a vertex of Q_i that supports L_c. Then the convex hull of $Q_1 \cup Q_2$ is inside P if and only if p_{1a} fully sees Q_2 and p_{2b} fully sees Q_1.*

Proof. Let T_a be the triangle with corners p_{1a}, p_{2a}, p_{2b} and T_b the triangle with corners p_{1b}, p_{2b}, p_{1a}; see Figure 7. Note that the convex hull of $Q_1 \cup Q_2$ is $Q_1 \cup T_b \cup Q_2 \cup T_a$. If p_{1a} fully sees Q_2, then $Q_2 \cup T_a$ is in the visibility polygon from p_{1a}, so $Q_2 \cup T_a \subset P$. Likewise, if p_{2b} fully sees Q_1, then $Q_1 \cup T_b \subset P$. If both hold, the convex hull of $Q_1 \cup Q_2$ is in P, as stated. ◀

We use the lemma in the special case where both Q_1 and Q_2 are triangles of our triangulation. Suppose that we do a BFS from Q_1 before Q_2 when constructing the visibility graph, as described in Section 2.2. We construct the visibility polygon V_1 from p_{1a} (as well as visibility polygons from the other corners of Q_1). Since V_1 is star-shaped, we can check in

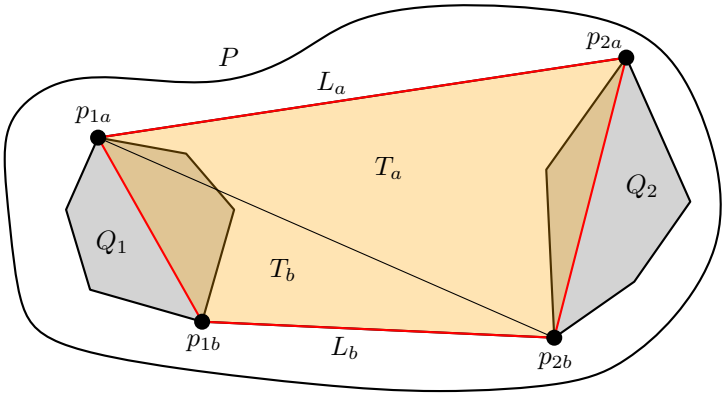

Figure 7 Illustration of Lemma 4: The convex hull of Q_1 and Q_2 is contained in P if and only if p_{1a} sees the quadrilateral F.

$O(\log |V_1|)$ time whether the three corners of Q_2 are contained in V_1 with a binary search-like algorithm. If so, we know that $Q_2 \subset V_1$ if and only if no edge of V_1 intersects the interior of Q_2. This is checked using range and segment trees, as described in Section 3.3.

Let V_2 be the visibility polygon from p_{2b}. We check that Q_1 is in V_2 in an analogous way (if we concluded that $Q_2 \subset V_1$). Storing visibility polygons from the corners of all triangles would require too much memory, so we only store them for the triangle from which we are currently performing a BFS. We will get the answer as to whether $Q_1 \subset V_2$ when we do the BFS from Q_2. Until then, it will be unknown whether or not there should be an edge between Q_1 and Q_2 in the visibility graph.

Note that instead of checking that $Q_2 \subset V_1$ and $Q_1 \subset V_2$, we could instead check directly that the quadrilateral $F = T_a \cup T_b$ is in V_1. However, as Q_1 and Q_2 are often small, far-apart triangles, the bounding rectangle of F may be much larger than those of Q_1 and Q_2. Hence, the use of range and segment trees (where we query the range and segment trees with the bounding rectangle of the object with which we want to check for intersections) would be much less efficient.

3.3 Containment using range and segment trees

We briefly explain here how we check whether the convex hull Q of a clique is contained in P in step (3). Since Q is the convex hull of a set of triangles contained in P (that form a clique in the visibility graph), we note that Q is contained in P if and only if no vertex of P is in the interior of Q and no edge of P enters the interior of Q. Let B be the axis-aligned bounding rectangle of Q. Using range and segment trees, we can find the vertices and segments of P intersecting B in output sensitive $O(\log^2 n + k)$ time, where k is the number of returned vertices or segments. We can then check whether the returned vertices are in the interior of Q and whether the edges enter the interior of Q using binary search-like algorithms. In order to construct the visibility graph in step (2), we likewise use range and segment trees when checking whether a triangle Q is contained in a visibility polygon V, as outlined in Section 3.2.

4 Evaluation

In this section we give additional insights into our implementation and the convex covers that it produces on the Challenge instances.

4.1 Implementation and data

Our code is written in C++ and compiled using GCC 11.3 with `-O3` optimization turned on. We use CGAL [12] for all geometric primitives with a Kernel that uses a number type that saves numbers as fractions and performs exact computations. For the partitioning and to construct the visibility graph, we use the triangulation, visibility, convex hull and range tree packages of CGAL [8, 9, 14, 10]. To compute the vertex cover, we use ReduVCC [11]. We show different types of instances of the problem set in Figure 8.

4.2 Examples

An important part of our approach is the choice of the partition, so in the following we discuss some of the effects that the different partitions mentioned in Section 2.1 have. While the Delaunay triangulation is fast to compute, the extension partition creates partitions with significantly more pieces and thus slows down our approach. To justify this kind of partition, it must lead to significantly better solutions. Figure 10 shows a cover of the same instance using the Delaunay partition and the extension partition — one can clearly see that the extension partition better adapts to the geometry of the input polygon. For example, consider the long horizontal light blue piece in the solution using the extension partition. A piece covering the same region cannot be assembled using the Delaunay partition: Any set of triangles from the Delaunay partition whose convex hull covers the blue piece spreads out too much vertically and is therefore not contained in the polygon.

Some Challenge instances have long concave chains on the boundary of P. Recall that the midpoint of each edge of such a chain has to be part of a distinct piece in the convex cover. To allow for creation of pieces that combine segments from multiple concave chains, we locally triangulate long concave chains instead of creating extensions. See Figure 11 for the effect that this partition has; especially, instead of only constructing pieces that contain parts of two different concave chains, with this partition we create pieces that contain parts of three concave chains.

Let us give an example of the adaptivity of our approach: In Figure 12 we show our solution for one of the `maze` instances. In this instance we want to use many long and thin pieces to cover the grid. However, at some places in the polygon there are some holes missing, and there we also want to use larger pieces to cover these areas. Our approach constructs such a solution, see Figure 12.

Finally, we show a selection of solutions on different instances in Figure 9. We want to highlight some properties of these solutions. In the solution at the top middle (`octa`) of Figure 9, we can observe that many pieces cover otherwise uncovered parts in different places of the polygon. In other words, the solution very much exploits that the pieces can overlap. We can see a similar behavior in the bottom left solution (`mc`). There our algorithm creates a lot of long pieces: instead of covering each of the "rooms" in the polygon with one large polygon and the remaining regions with small polygons (as one might at first do when constructing a cover for an instance like this), it elegantly connects non-convex features in different places of the polygon and thereby achieves a smaller number of pieces.

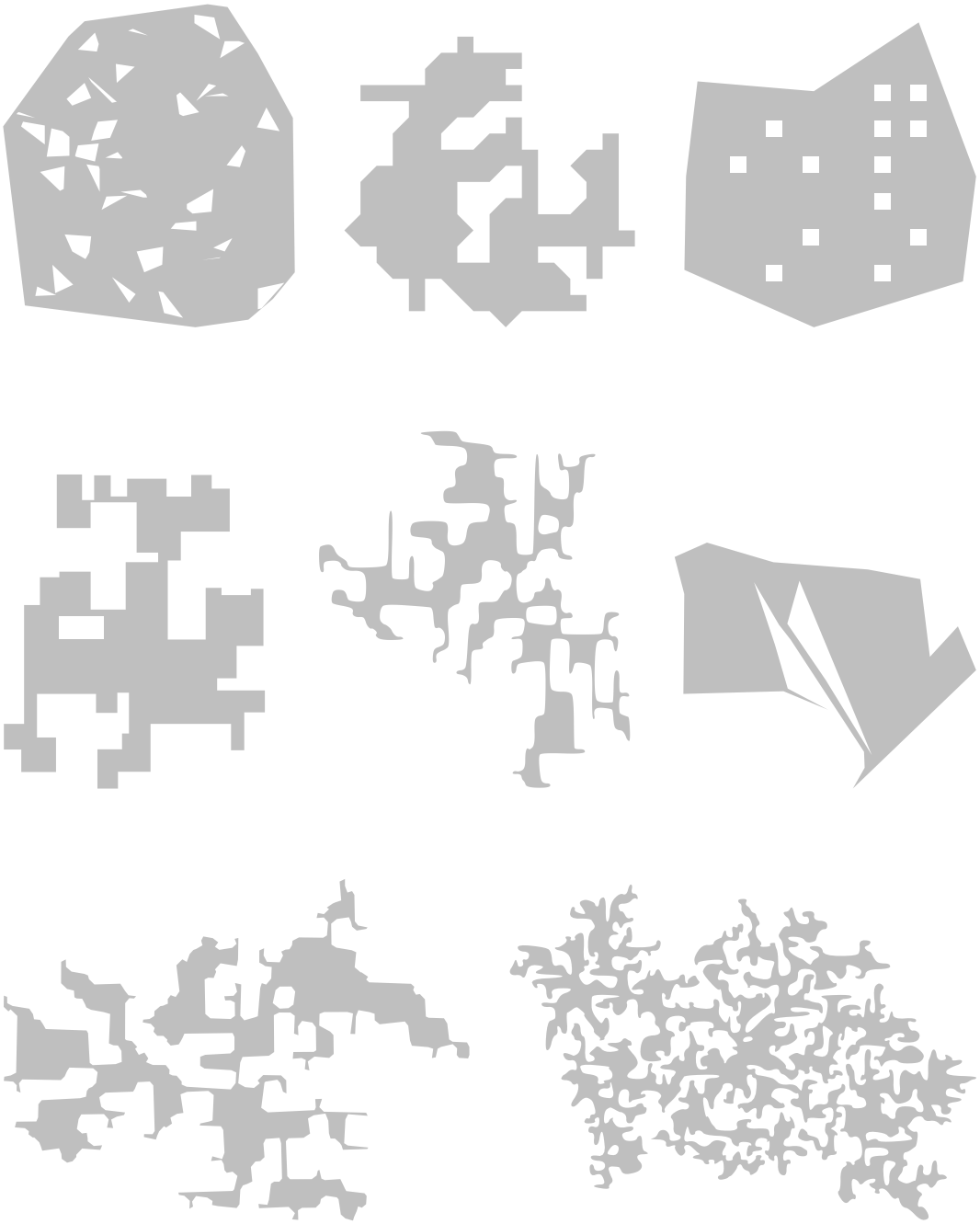

Figure 8 Examples of instance types `cheese`, `octa`, `maze`, `iso`, `smr`, `fpg`, `mc` and `smo` from left to right and top to bottom.

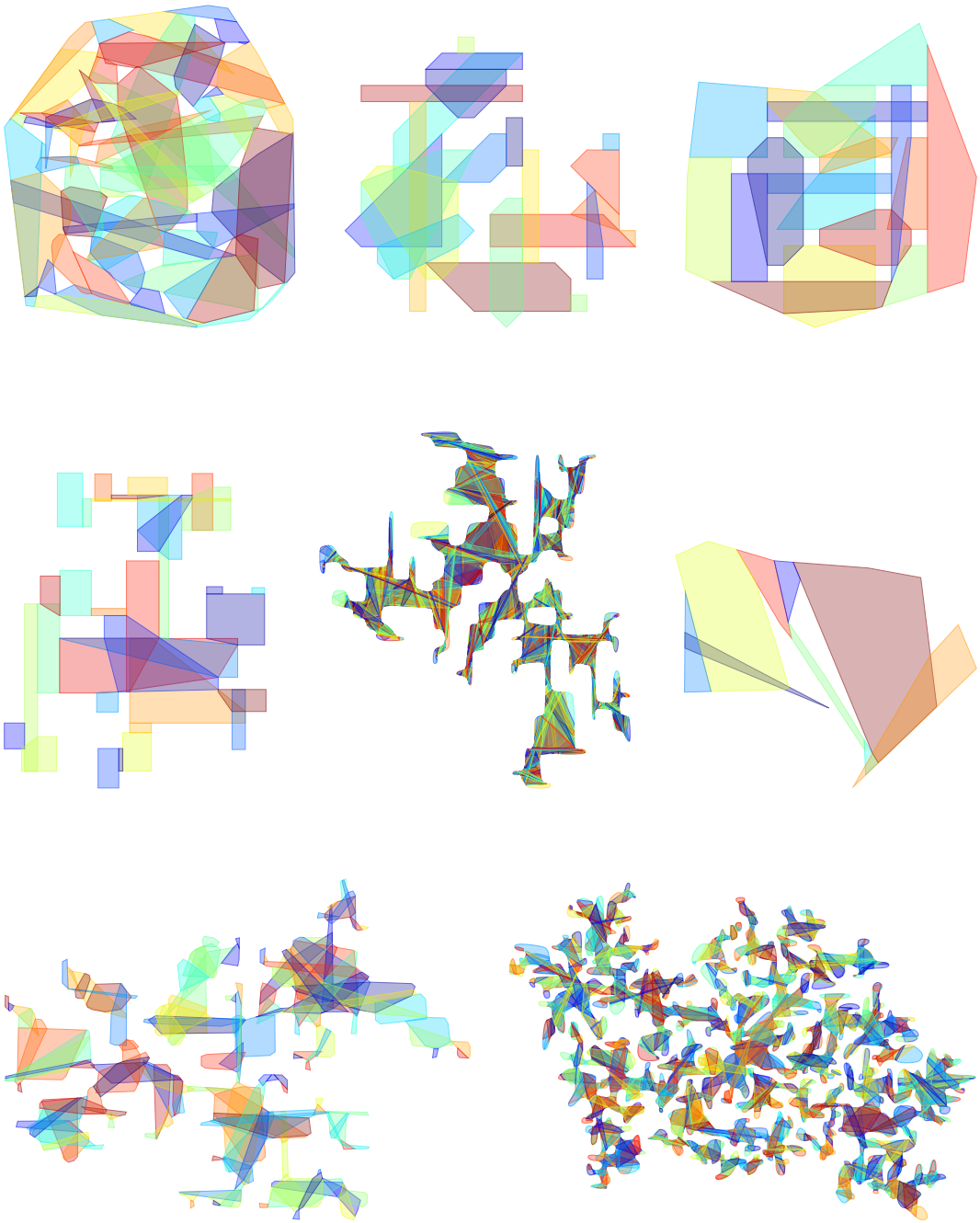

Figure 9 Solutions to the examples from figure 8. Solutions sizes are 65 for `cheese`, 22 for `octa`, 18 for `maze`, 33 for `iso`, 1312 for `smr`, 9 for `fpg`, 152 for `mc` and 1179 for `smo` from left to right and top to bottom.

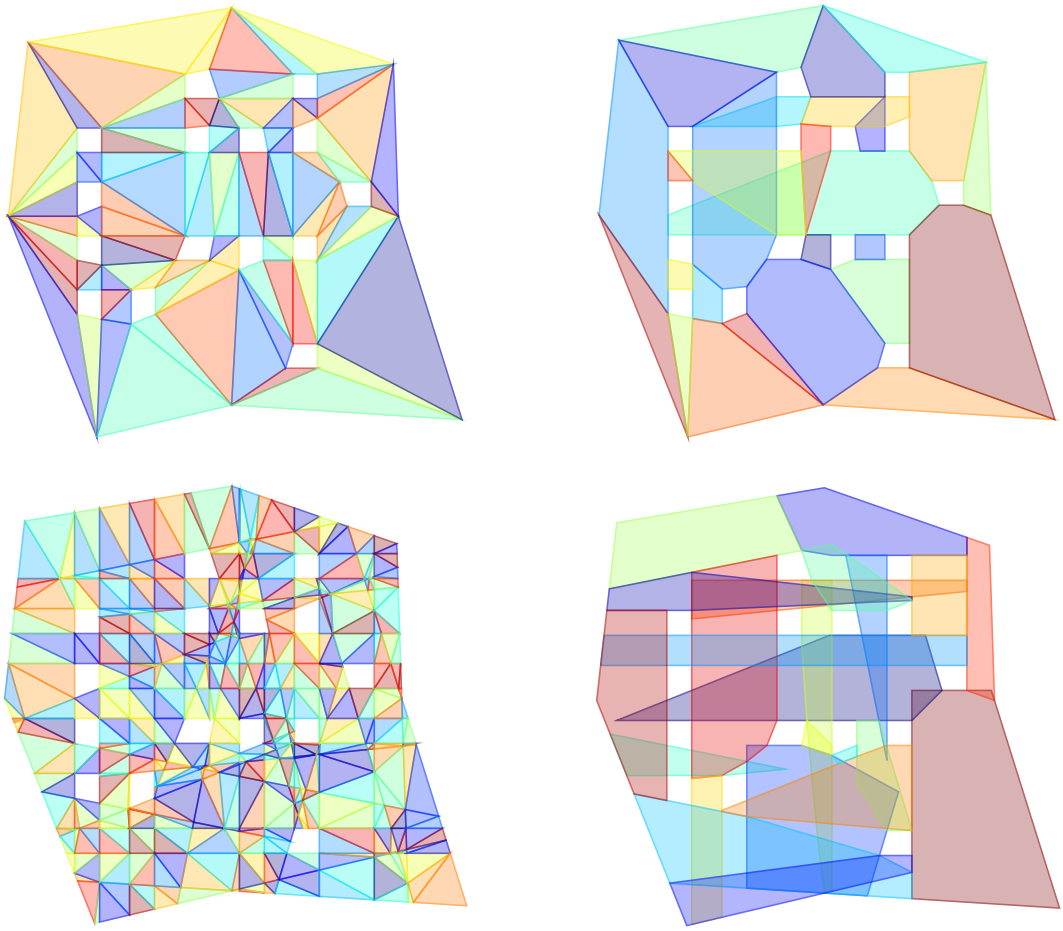

Figure 10 The Delaunay triangulation (top left) and the resulting cover of size 27 (top right). The extension partition (bottom left) and the resulting cover of size 23 (bottom right).

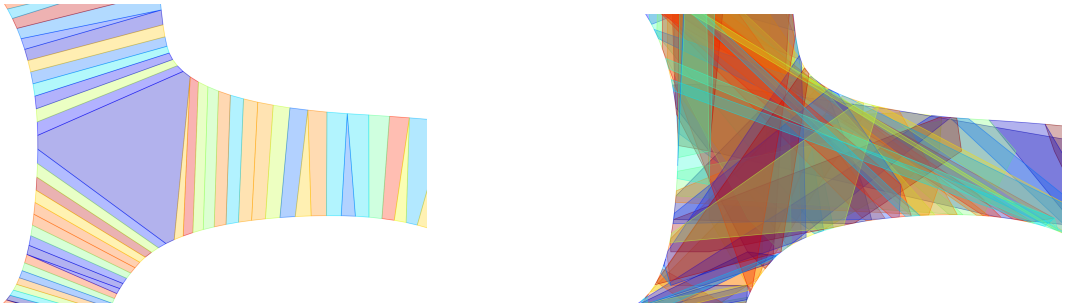

Figure 11 Part of a polygon with multiple long concave chains. While the cover using a Delaunay triangulation almost exclusively creates pieces adjacent to only two concave chains (left), local triangulation allows for creation of pieces that contain parts of all three concave chains (right).

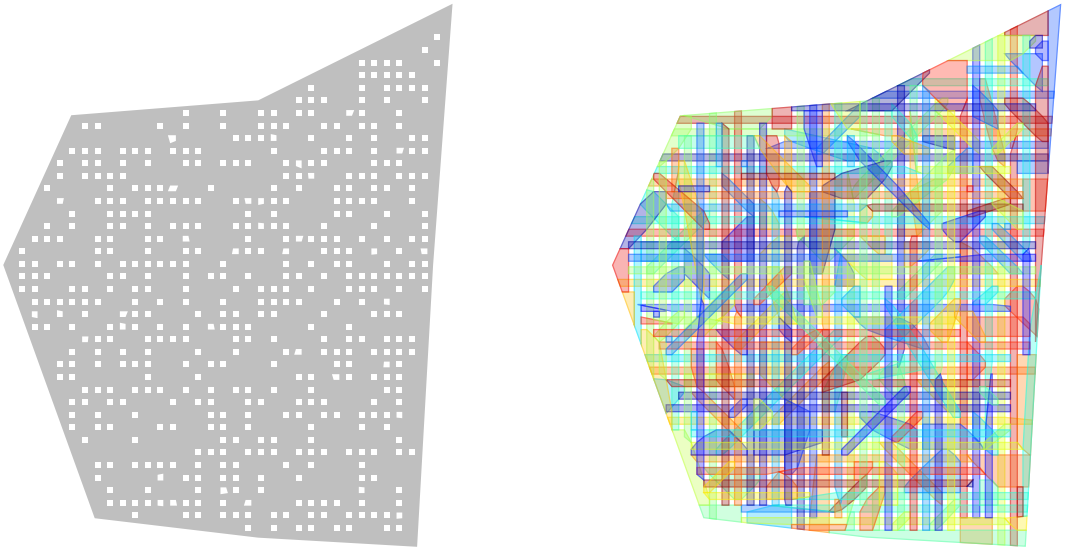

Figure 12 The best solution found to a `maze` instance. A lot of long horizontal and vertical pieces are used.

4.3 Experiments

For this section, we selected a subset of the instances for more thorough experiments and subsequently only refer to these. See the sizes and types in the plots of Figure 13. Recall that we compute an intermediate, potentially infeasible solution via a vertex clique cover that is subsequently fixed. We argue above that we expect that only few cliques have to be fixed on practical instances. Indeed, on all except the `cheese` instances, the solution size increased by at most 6 pieces when fixing cliques, while most small instances did not have any invalid clique. However, the largest increase in solution size was for the largest `cheese` instances with an increase of 110 pieces. Our algorithm may create redundant cliques that are removed in a post-processing step, so it is interesting to consider how much this post-processing reduces the size of the solution. This decrease in pieces is very much dependent on the instance: While for `octa` the maximal decrease was 2 pieces, it was 83 pieces for `cheese` instances and all larger `cheese` instances saw significant improvements.

For the competition and our experiments we used a server with two Intel Xeon E5-2690 v4 CPUs with 14 cores (28 threads) each, and a total of 504GB RAM. All reported running times are single-threaded. The bottleneck of our approach is the visibility graph computation discussed in Section 2.2. To better understand the trade-off between running time, memory usage, and solution quality with respect to the choice of partition, we conduct experiments comparing Delaunay triangulation and extension partition; see Figure 13. The extension partition introduces a large overhead in running time and memory consumption compared to the Delaunay triangulation, but it reduces the solution size by a significant fraction. While for the extension partition the visibility graph computation clearly dominates the running time, for the Delaunay triangulation it only makes up 32.8% of the running time on average.

5 Conclusion

We developed a practical algorithm for the computation of convex covers of polygons (potentially with holes), with the aim of minimizing the number of convex pieces. The

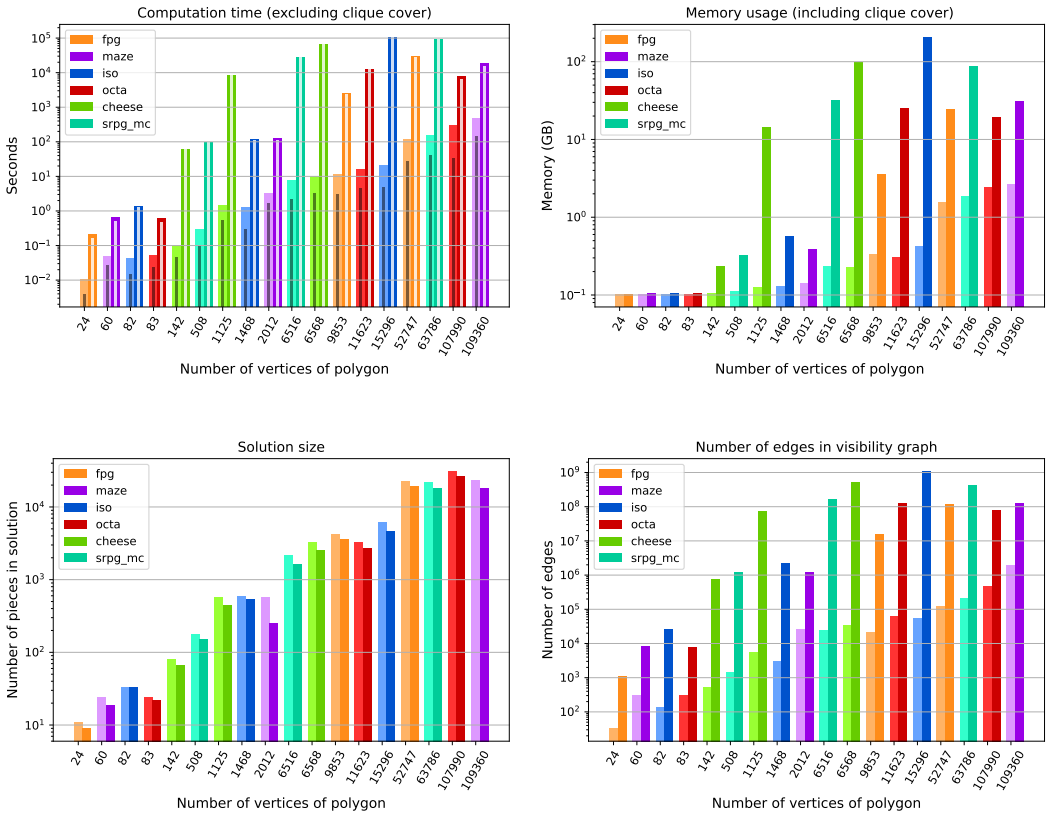

Figure 13 Experiments with the Delaunay triangulation (left bars) and the extension partition (right bars) as underlying partitions for a selected set of instances. We measure the running time (top left; thin bars showing the running time of the visibility graph computation), memory usage (top right), solution size (bottom left), and number of edges in the visibility graph (bottom right).

algorithm follows a three-step approach, where we first compute a partition, then extract a visibility graph of this partition, and finally solve a vertex clique cover problem on this graph. We showed that this approach is powerful and versatile. An intuitive explanation for the success of this approach is that, depending on the instance, we can use different types of convex partitions which are tailored to the structure of the instance. Additionally, we could leverage already existing engineered software for the vertex clique cover problem.

References

1 Mikkel Abrahamsen. Covering polygons is even harder. In *Symposium on Foundations of Computer Science (FOCS)*, pages 375–386, 2021. doi:10.1109/FOCS52979.2021.00045.

2 Mikkel Abrahamsen and Dan Halperin. Ten problems in geobotics, 2024. doi:10.48550/arXiv.2408.12657.

3 Mikkel Abrahamsen, William Bille Meyling, and André Nusser. Constructing concise convex covers via clique covers (CG challenge). In *International Symposium on Computational Geometry (SoCG 2023)*, pages 66:1–66:9, 2023. doi:10.4230/LIPICS.SOCG.2023.66.

4 David Chalupa. Construction of near-optimal vertex clique covering for real-world networks. *Comput. Informatics*, 34(6):1397–1417, 2015. URL: http://www.cai.sk/ojs/index.php/cai/article/view/1276.

5 Joseph C. Culberson and Robert A. Reckhow. Covering polygons is hard. *J. Algorithms*, 17(1):2–44, 1994. doi:10.1006/jagm.1994.1025.

6 Guilherme Dias da Fonseca. Shadoks approach to convex covering (CG challenge). In Erin W. Chambers and Joachim Gudmundsson, editors, *39th International Symposium on Computational Geometry, SoCG 2023, June 12-15, 2023, Dallas, Texas, USA*, volume 258 of *LIPIcs*, pages 67:1–67:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.SoCG.2023.67, doi:10.4230/LIPICS.SOCG.2023.67.

7 Sándor P. Fekete, Phillip Keldenich, Dominik Krupke, and Stefan Schirra. Minimum coverage by convex polygons: The CG:SHOP Challenge 2023, 2023. doi:10.48550/arXiv.2303.07007.

8 Michael Hemmer, Kan Huang, Francisc Bungiu, and Ning Xu. 2D visibility computation. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgVisibility2.

9 Susan Hert and Stefan Schirra. 2D convex hulls and extreme points. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgConvexHull2.

10 Gabriele Neyer. dD range and segment trees. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgSearchStructures.

11 Darren Strash and Louise Thompson. Effective data reduction for the vertex clique cover problem. In *Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 41–53, 2022. doi:10.1137/1.9781611977042.4.

12 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html.

13 Peter Werner, Alexandre Amice, Tobia Marcucci, Daniela Rus, and Russ Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. *CoRR*, abs/2310.02875, 2023. URL: https://doi.org/10.48550/arXiv.2310.02875.

14 Mariette Yvinec. 2D triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.2 edition, 2023. URL: https://doc.cgal.org/5.5.2/Manual/packages.html#PkgTriangulation2.