Fixed Parameter Algorithms for Minimum Weight Partitions

Christian Borgelt[†], Magdalene Grantson[*], and Christos Levcopoulos[*]

Abstract

In this paper we propose to solve two hard geometric optimization problems: We describe a fixed parameter algorithm for computing the minimum weight triangulation (MWT) of a simple polygon with $(n-k)$ vertices on the perimeter and k hole vertices in the interior, that is, for a total of n vertices. We show that the MWT can be found in time at most $O(n^4 4^k k)$, and thus in time polynomial in n if $k \leq O(\log n)$. We implemented our algorithm in Java and report experiments backing our analysis.

Given a convex polygon with $(n-k)$ vertices on the perimeter and k hole vertices in the interior, that is, for a total of n vertices, we also describe a fixed parameter algorithm for computing the minimum weight convex partition (MWCP) of the input. We show that the MWCP problem can be found in at most $O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$ time, and thus at $O(n^3)$ time if k is constant, and in time polynomial in n if $k = O(\frac{\log n}{\log \log n})$. Our results for the MWCP problem hold also for the more general case where the input is an n-vertex PSLG and k is the total number of holes and/or reflex vertices inside the convex hull.

1 Introduction

We propose to solve two hard geometric optimization problems in this paper:

1. We consider the problem of finding the *minimum weight triangulation* (MWT) of a simple polygon with $n-k$ vertices on the perimeter and k hole vertices in the interior, i.e., for a total of n vertices. In this case, a MWT is a maximal set of non-intersecting edges with minimum total edge length, all of which lie inside the polygon. The problem of finding the MWT of a set S of points can be reduced to this problem by finding the convex hull of S, which is then treated as a (convex) polygon, while all vertices not on the convex hull are treated as hole vertices. The MWT problem has several applications [17, 15].

2. We also consider the problem of finding the *minimum weight convex partition* (MWCP) of a convex polygon with $n-k$ vertices on the perimeter and k hole vertices in the interior, i.e., for a total of n vertices. In this case, the MWCP is a convex partition such that the total edge length is minimised. The MWCP has applications in computer graphics [17], image processing [15], database systems [13], and data compression [17].

It is known that the MWCP of G is NP-hard [11]. With respect to a convex polygon with a single hole vertex or a convex polygon with two hole vertices, we [7] gave $O(n)$ and $O(n^2)$ time algorithms, respectively. With respect to n-vertex polygons without holes, Keil [9] developed a dynamic programming algorithm that runs in $O(n^2 N^2 \log n)$ time, where N is the number of concave vertices. Later Agarwal, Flato, and Halperin [1] improved this time bound to $O(n^2 N^2)$.

The complexity status of the MWT problem has been open since 1975, when it was included in a list of problems neither known to be NP-complete or solvable in polynomial time [5]. Recently, however, it was reported that the MWT problem is NP-hard [16]. For $k = 0$ (no holes), however, a MWT can be found by dynamic programming in time $O(n^3)$ [6, 12].

Recent attempts to give exact algorithms for computing a MWT and a MWCP exploit the idea of developing a so-called *fixed parameter algorithm*. Such an algorithm has a time complexity of $O(n^c \cdot f(k))$, where n is the input size, k is a (constrained) parameter, c is a constant independent of k, and f is an arbitrary function [3].

W.r.t. a MWT of a simple polygon with holes the total number n of vertices is the size of the input and we may choose the number k of hole vertices as the constrained parameter. An algorithm based on such an approach was presented in [10] and analyzed to run in $O(n^5 \log(n)\, 6^k)$ time. In this paper we describe a fixed parameter algorithm inspired by a basic observation in this algorithm, but deviating in several respects. Due to improvements in both the algorithm and its analysis, we are able to show that the time needed to find a MWT of a polygon with hole vertices is at most $O(n^4 4^k k)$. In addition, we implemented our algorithm in Java and performed experiments backing our analysis.

W.r.t. a MWCP of a convex polygon with holes the total number n of vertices is the size of the input and we may choose the number k of hole vertices as the constrained parameter. In this paper, we describe a

[*]Department of Computer Science, Lund University, Box 118, 221 Lund, Sweden, {magdalene,christos}@cs.lth.se, [†]Department of Knowledge Processing and Language Engineering, University of Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany, borgelt@iws.cs.uni-magdeburg.de

fixed parameter algorithm for computing the MWCP of a convex polygon G with $n-k$ vertices on the perimeter and k hole vertices in the interior, that is, for a total of n vertices. We show that our algorithm can solve the problem in at most $O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$ time, and thus at $O(n^3)$ time if k is constant, and in time polynomial in n if $k = O(\frac{\log n}{\log \log n})$. Our results for the MWCP problem hold also for the more general case where the input is an n-vertex PSLG and k is the total number of holes and/or reflex vertices inside the convex hull.

The paper is structured as follows. In Section 2 we discuss our results for the MWT problem. In section 3 we discuss our results for the MWCP problem.

The full versions of both algorithms can be found in [7] and [8] respectively.

2 A Fixed Parameter Algorithm for the MWT Problem

In this Section we will discuss our results for the MWT problem.

2.1 Preliminaries and Basic Idea

Following [2], we call a polygon with holes a *pointgon* for short. We denote the set of $(n-k)$ perimeter vertices by $V_p = \{v_1, v_2, \ldots, v_{n-k}\}$, assuming that they are numbered in counterclockwise order starting at an arbitrary vertex. The set of k hole vertices we denote by $V_h = \{v_{n-k+1}, v_{n-k+2}, \ldots, v_n\}$. The set of all vertices is denoted by $V = V_p \cup V_h$, the pointgon formed by them is denoted by G.

Definition 1 *A vertex $u \in V$ is said to be* lexicographically smaller *than a vertex $v \in V$, written $u \prec v$, iff (1) the x-coordinate of u is smaller than the x-coordinate of v or (2) the x-coordinate of u is equal to the x-coordinate of v, but the y-coordinate of u is smaller than the y-coordinate of v.*

W.l.o.g. we assume that the vertices in V_h are in lexicographical order, i.e., $\forall i; n-k < i < n : v_i \prec v_{i+1}$. (Otherwise we can sort and renumber them.)

Definition 2 *A path in a pointgon G, i.e., a sequence of vertices from V, is called* lexi-monotone *iff it is either lexicographically increasing or lexicographically decreasing. A* separating lexi-monotone path *(or simply a* separating path*) is a lexi-monotone path with start and end vertices on the perimeter of G (i.e. vertices in V_p) and a (possibly empty) sequence of hole vertices (i.e. vertices in V_h) in the middle, which does not intersect the perimeter of G.*

With these definitions, the core idea of our algorithm (as well as the core idea of the algorithm in [10]) is based on the following observation:

Observation 1 *Let $v \in V_p$ be an arbitrary vertex on the perimeter of a pointgon G. Then in every triangulation T of G there exists:* either *a separating path π starting at v* or *two perimeter vertices v_c and v_{cc} that are adjacent to v and that together with v form a triangle without any hole vertices in its interior.*

As a consequence, we can try to find the MWT of a given pointgon G with a recursive procedure that considers possible splits of G into at most two subpointgons (using the above observation). The MWT is then obtained as the minimum over all these splits.

Formally, we can describe the solution procedure as follows: Let G be a given pointgon and $v \in V_p$ an arbitrary vertex on the perimeter of G. Let $\Pi(G, v)$ be the set of all separating paths of G starting at v. If $\pi \in \Pi(G, v)$ is a separating path, let $|\pi|$ be the length of π and $L(G, \pi)$ and $R(G, \pi)$ the sub-pointgons to the left and to the right of π, respectively. Furthermore, let v_c and v_{cc} be the perimeter vertices that are adjacent to v in clockwise and counterclockwise direction, respectively. Then the weight of a MWT of G can be computed recursively as

$$\min \Big\{ \min_{\pi \in \Pi(G,v)} \{ \mathrm{MWT}(L(G,\pi)) + \mathrm{MWT}(R(G,\pi)) - |\pi| \}, \; \mathrm{MWT}(R(G,(v_{cc},v_c))) + |(v,v_{cc})| + |(v,v_c)| \Big\}.$$

The first term in the outer minimum considers all splits by separating lexi-monotone paths. The second term in the outer minimum refers to the special path (v_{cc}, v_c) that "cuts off" v from the rest of the pointgon if the triangle (v, v_{cc}, v_c) does not contain any hole vertices. Although the above recursive formula only computes the weight of a MWT, it can easily be extended to yield the edges of a MWT by returning the set of edges that is added in order to achieve a triangulation for each recursive call. The union of these sets of edges for the term that yields the minimum weight is a MWT for the original pointgon G.

2.2 Dynamic Programming

To apply dynamic programming, we have to identify the different subproblems that we meet in the recursion, and we have to find a representation for them. The core idea here is: if in the recursion we prefer to use the same vertex v for attaching separating paths as in the preceding split, every subproblem we encounter can be described by one or two lexi-monotone paths that start at the same vertex v (which we call an *anchor* of the subproblem) and a coherent piece of the perimeter of the input pointgon. An analysis of this statement, providing a proof, will be given later.

We represent a subproblem by an index word over an alphabet with n characters, which uniquely identifies each subproblem. This index word has the general form (v, π_{cc}, π_c) and describes a counterclockwise

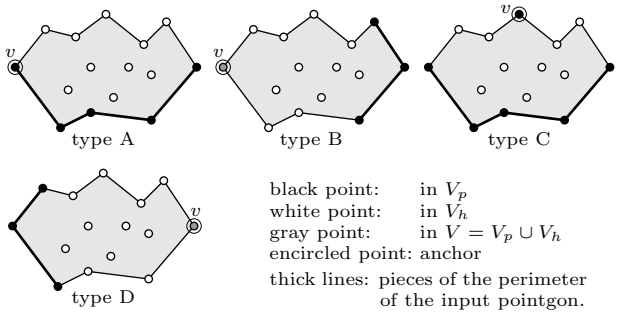

Figure 1: The four types of pointgons we encounter.

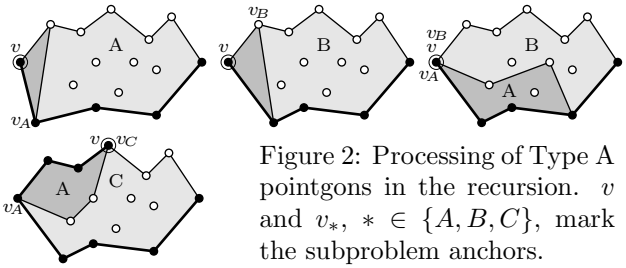

Figure 2: Processing of Type A pointgons in the recursion. v and v_*, $* \in \{A, B, C\}$, mark the subproblem anchors.

walk along the perimeter of the subproblem. The first element is the anchor v, which may be a perimeter vertex or a hole vertex of the input pointgon and thus can have n possible values. π_{cc} and π_c describe the sequences of hole vertices of the input pointgon that are on the separating paths. All elements of π_{cc} and π_c are in V_h—with the possible exception of the last elements, which may be perimeter vertices s_{cc} and s_c, respectively. The vertices in a coherent perimeter piece between the end vertices s_{cc} and s_c (if such a perimeter piece exists) are not part of the subproblem representation, but are left implicit.

Instead of pure dynamic programming, we use *memorized tree recursion* based on a trie structure, which is accessed through the index word representing a subproblem. In each recursive call, we first access the trie structure in order to find out whether the solution to the current subproblem is already known. If it is, we simply retrieve and return the solution. Otherwise we carry out the split computations and in the end store the found solution in the trie.

2.3 Types of Pointgons

Apart from the input pointgon, which is of neither of these types, we encounter four types of sub-pointgons (see Figure 1; this set differs from the one used in [10]):

Type A pointgons have only one separating path starting at the anchor v, which must be on the perimeter of the input pointgon. The vertices on the path are lexicographically increasing. There is also a coherent perimeter piece of the input pointgon.

Type B pointgons are bounded by two separating paths starting at the anchor v, which may be either a perimeter vertex or a hole vertex of the input pointgon. The vertices on both paths are lexicographically increasing. There may or may not be a coherent perimeter piece of the input pointgon.

Type C pointgons are bounded by two separating paths starting at the anchor v, which must be a perimeter vertex of the input pointgon. One of them is lexicographically increasing, the other decreasing. There is a perimeter piece of the input pointgon.

Type D pointgons are bounded by two separating paths starting at the anchor v, which may be either a perimeter vertex or a hole vertex of the input pointgon. The vertices on both paths are lexicographically decreasing. There must be a perimeter piece of the input pointgon, which contains at least two vertices.

The general principle of the choice of the anchor of a subproblem is that it is the leftmost vertex on the separating path if there is just one path, and the vertex that is on both paths if there are two separating paths. If there are two vertices that are on both paths (because they share both start and end vertex), we choose the leftmost of the two. For the input pointgon, we choose the lexicographically smallest perimeter vertex as the anchor.

For the input pointgon, regardless of whether the path starts at the anchor or cuts off the anchor, we obtain a sub-pointgon of type A for the subproblems. The other types of pointgons can only be created in deeper levels of the recursion. In the following we consider how these types of pointgons are treated in the recursion in our algorithm and thus also prove that these are the only types of pointgons that occur.

Type A: The different splits of a type A pointgon are sketched in Figure 2. On the very left a path "cutting off" the anchor, which is seen as leading from the counterclockwise neighbor of v to its clockwise neighbor, can be merged with the existing separating path to give a new type A pointgon. Otherwise, we obtain a type B pointgon (second sketch). For a path starting at the anchor, we distinguish whether it is lexicographically increasing (third sketch) or decreasing (fourth sketch, note the different anchor). In the former case, we obtain one type A and one type B pointgon, which receive the same anchor as the original pointgon. In the latter case, we obtain one type A pointgon, with its anchor at the end of the new separating path, and one type C pointgon, with its anchor equal to that of the original pointgon. Note that all cases may also occur mirrored at a horizontal axis, which should also be kept in mind for the other types.

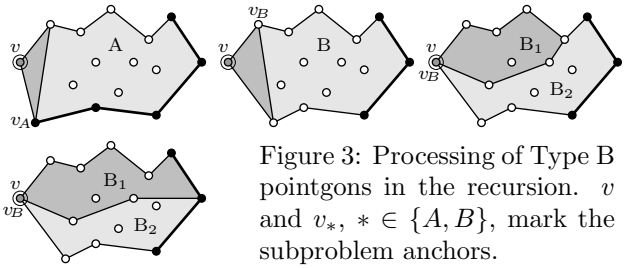

Figure 3: Processing of Type B pointgons in the recursion. v and v_*, $* \in \{A, B\}$, mark the subproblem anchors.

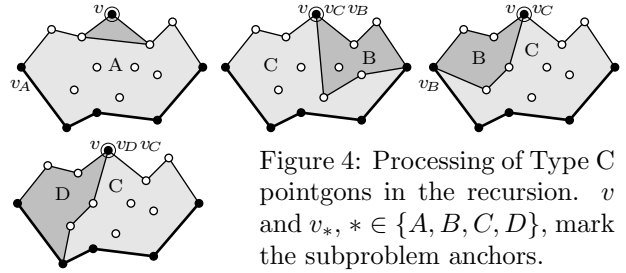

Figure 4: Processing of Type C pointgons in the recursion. v and v_*, $* \in \{A, B, C, D\}$, mark the subproblem anchors.

Type B: Type B pointgons behave similarly to type A pointgons (see Figure 3). Again we have to check whether a type A pointgon can result (first sketch). Otherwise we get a type B pointgon with an anchor that is one end of the cutting path (second sketch). For separating paths starting at the anchor only type B pointgons can result (third and fourth sketch), since both separating paths are increasing.

Type C: Type C pointgons (which do not appear in [10]) are the most complicated case (see Figure 4). If the anchor is "cut off", we only have one separating path, so the anchor is set to its starting vertex and we obtain a type A pointgon (first sketch). If a separating path is attached to the anchor, we have to distinguish whether it is lexicographically increasing or decreasing. Increasing paths are simpler, leading to a split into one type C and one type B pointgon (second sketch). If the path is lexicographically decreasing, we have to check whether there is a perimeter piece of the input pointgon with at least two vertices. If there is not, we obtain one type B pointgon, with its anchor at its leftmost vertex, and one type C pointgon, which maintains the anchor of the original pointgon (third sketch). Otherwise we obtain one type D and one type C pointgon, both of which receive the anchor of the original pointgon (fourth sketch).

Type D: Type D pointgons behave symmetrically to type B pointgons. When the anchor is "cut off" we also have to check whether a type A pointgon results (first sketch). Otherwise we get a type D pointgon with an anchor that is one end of the cutting path (second sketch). For separating paths starting at the anchor either one type B and one type D pointgon (third sketch), namely if one perimeter piece is empty, or two type D pointgons result (fourth sketch).

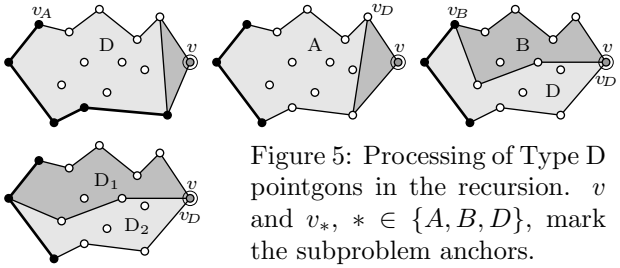

Figure 5: Processing of Type D pointgons in the recursion. v and v_*, $* \in \{A, B, D\}$, mark the subproblem anchors.

Comparison to [10]: Our approach gives rise to a different set of sub-pointgons. An important advantage of our approach is that it uses a coherent scheme for processing the sub-pointgons, which is strictly based on either "cutting off" the anchor or attaching a lexi-monotone path to the anchor (Section 2.1). In contrast to this, the approach of [10] needs an additional split type (when processing a type 1/type A pointgon).

2.4 Analysis

To estimate the time complexity of our algorithm, we group the subproblems and analyze the groups separately. The groups are defined by the number of hole vertices the sub-pointgon has on its perimeter.

So consider the number of subproblems with l, $0 \leq l \leq k$, hole vertices on the perimeter. The worst case is that we have three perimeter vertices of the input pointgon, namely the anchor and the two ends of the separating paths. This gives us a factor of n^3. Next we have to choose l of the k input hole vertices, for which we have $\binom{k}{l}$ possibilities, and then we have to distribute the chosen hole vertices on the two paths, for which there are 2^l possibilities. As a consequence we have in the worst case $O(n^3\binom{k}{l}2^l)$ possible sub-pointgons with l holes on the perimeter.

Given a sub-pointgon with l holes on the perimeter, there are at most $k - l$ holes left to form a separating path and at most n end points. This gives us a maximum of $n2^{k-l}$ possible paths. For each path, we have to check whether it intersects the perimeter of the sub-pointgon. This check can exploit a preprocessing step in which we determine for each edge that could be part of a separating path whether it intersects the perimeter of the input pointgon or not. The resulting table has a size of at most n^2. With this table we can check in $O(k - l)$ whether a given separating path intersects a (possibly existing) perimeter piece. We also have to check for an intersection with the at most two already existing separating paths, which contain at most $l + 2$ edges. By exploiting that all paths are lexi-monotone, this check can be carried out in $O(k)$. Once a path is found to be valid, the sub-pointgons have to be constructed by collecting their at most $k + 3$ defining vertices, and their

$n-k$	k	time in seconds	time/$n^4 4^k k$
3	1	0.008 ± 0.000	$7.813 \cdot 10^{-6}$
6	1	0.009 ± 0.000	$9.371 \cdot 10^{-7}$
9	2	0.039 ± 0.003	$8.324 \cdot 10^{-8}$
12	3	0.071 ± 0.004	$7.305 \cdot 10^{-9}$
15	4	0.121 ± 0.020	$9.067 \cdot 10^{-10}$
18	5	0.370 ± 0.090	$2.582 \cdot 10^{-10}$
21	6	1.365 ± 0.539	$1.045 \cdot 10^{-10}$
24	7	5.402 ± 2.209	$5.100 \cdot 10^{-11}$
27	8	25.335 ± 10.449	$3.220 \cdot 10^{-11}$
30	9	90.641 ± 34.399	$1.661 \cdot 10^{-11}$

Table 1: Results obtained from our Java implementation. All results are averages over 20 runs.

solutions have to be looked up. Both operations take $O(k)$ time. Finally the length of the path has to be computed, which takes $O(k-l)$ time. Thus processing one path takes in all $O(k)$ time.

Therefore the overall time complexity is $O(n^4 4^k k)$.

2.5 Implementation

As already mentioned, we implemented our algorithm in Java. Example results for different numbers of holes and perimeter vertices (averages over 20 runs, convex pointgons) are shown in Table 1. The test system was an Intel Pentium 4C@2.6GHz with 1GB of main memory running S.u.S.E. Linux 10.0 and Sun Java 1.5.0_03. To check our estimate of the time complexity, we computed the ratios of the measured execution times to the theoretical values. As can be seen, these ratios are decreasing for increasing values of n and k, indicating that the theoretical time complexity is actually a worst case, while average results in practice are considerably better. The source code of our implementation can be downloaded at http://fuzzy.cs.uni-magdeburg.de/~borgelt/pointgon.html.

3 A Fixed Parameter Algorithm for the MWCP Problem

In this Section we will discuss our results for the MWCP problem.

3.1 Preliminaries

We consider as input a convex polygon with $(n-k)$ vertices on the perimeter and k hole vertices, thus a total of n input vertices. We call such a convex polygon with holes a *convex pointgon* for short. We denote the set of perimeter vertices by $V_p = \{v_0, v_1, \ldots, v_{n-k-1}\}$, assuming that they are numbered in counterclockwise order starting at an arbitrary vertex. The set of hole vertices we denote by $V_h = \{v_{n-k}, v_{n-k+1}, \ldots, v_{n-1}\}$. The set of all vertices is denoted by $V = V_p \cup V_h$, the convex pointgon formed by them is denoted by G.

Given a convex pointgon G a *p-edge* is an edge from a hole vertex to any vertex on the perimeter.

Fact 1 *Given a convex pointgon G , apart from edges going between hole vertices, at most 3 p-edges incident to a hole vertex are sufficient to induce a convex partition.*

Proof. At most three p-edges suffice for each hole vertex because if a hole vertex is incident to four p-edges, there must be one that can be removed without introducing a concavity at the hole vertex. Removing it also does not introduce a concavity at the perimeter, because the polygon we consider is convex. □

From the proof in Fact 1, we observe a decisive difference between a p-edge and an edge going between hole vertices. That is, removing an edge going between hole vertices so that no concavity is introduced at one end point may very well introduce a concavity at the other end point and thus we cannot remove it.

From Fact 1 we deduce a more special case:

Fact 2 *For any MWCP of G there are at most three p-edges incident to a hole vertex.*

Proof. At most three p-edges are needed for each hole vertex because of the same reasoning in the proof of Fact 1. Moreover removing the forth p-edge decreases the weight for the case of the MWCP problem. □

Given a convex polygon with k vertex holes, at most 3 p-edges incident to each hole vertex are needed to induce a minimum convex partition. Thus a maximum of at most $3k$ p-edges can be in the solution. Given at most $3k$ p-edges we can also check whether they intersect in constant time since k is constant. A simple and obvious way to solve the problem is to use a brute-force approach of examining all the possible ways of selecting the $3k$ p-edges. For each possible $3k$ p-edges we consider all possible combinations of non-crossing edges going between hole vertices. There are at most $O(2^3 \cdot 59)^k \cdot k^{-6})$ such combinations [4, 18]. Selecting the best over all combinations solves the problem in $O(n^{3k} \cdot (2^3 \cdot 59)^k \cdot k^{-6})$ time.

Given a convex pointgon G, we consider partitioning G into so called *v-pieces*. These v-pieces serve the purpose to divide the problem into subproblems such that a subproblem denotes a piece of G containing a single hole vertex. The v-pieces alone do not necessarily yield a convex partition. The subproblems representing v-pieces are solved and the results are combined. The idea is that for any given convex partition **CP**, the v-pieces and **CP** are compatible if and

only if for each hole vertex $v_h \in V_h$, the p-edges incident to v_h in **CP** lie only in one or several v-pieces of v_h.

Definition 1 A v-piece (v_h, v_1, v_2) is a part of a given polygon P with holes, where $v_h \in V_h$ is a hole vertex and $v_1 \in V_p$ and $v_2 \in V_p$ are (not necessarily distinct) perimeter vertices (the so-called *vital points* of the v-piece). It is the part bounded by the two p-edges (v_h, v_1) and (v_h, v_2), the so-called *vital edges*, and the part of the perimeter of G that is traversed by a counterclockwise walk from v_1 to v_2. If a v-piece (v_h, v_1, v_2) contains no other hole vertices, we call it a *legal v-piece* (or just a *v-piece*, for short). If a v-piece (v_h, v_1, v_2) contains other hole vertices, we call it an *illegal v-piece*.

Note that the v-piece (v_h, v_1, v_2) is not the same as the v-piece (v_h, v_2, v_1). That is, the order of the vertices v_1 and v_2 is important, since the part of the perimeter of G that bounds the v-piece is defined by a *counter-clockwise* walk from the first vertex to the second. Therefore (v_h, v_1, v_2) and (v_h, v_2, v_1) are complements of each other w.r.t. the convex pointgon G; their union is the whole convex pointgon G.

The idea of the algorithm is to partition the perimeter of G into (legal) v-pieces, such that each hole vertex h has at most 3 v-pieces. Three v-pieces suffice, because no more than three p-edges per hole vertex are needed to achieve a convex partition (see Fact 1). If there were more than 3 v-pieces, at most three can contribute p-edges to the solution. Therefore we can remove all v-pieces that do not contribute without changing the solution. The partition should be such that the v-pieces of a coonvex pointgon G put together contain the entire perimeter of G (but not necessarily the entire polygon P). Adjacent v-pieces, that is, v-pieces that have at least one perimeter vertex in common, should belong to different hole vertices.

The core idea of our algorithm is that in order to find a MWCP it suffices to search all possible partitions into v-pieces, where the partition is such that two v-pieces associated with the same whole do not share a vital point, but all v-pieces together cover the perimeter. To show this, we only have to show that any convex partition has a compatible partition into v-pieces, from which it can be derived, so that we do not "miss" any convex partition by searching only partitions into v-pieces. One can do this by initially constructing an v-piece partition by placing one vital point at the end of each p-edge, associating it with the hole vertex the p-edge leads to. We turn this into a list, by starting at a perimeter vertex and following the perimeter counterclockwise, numbering vertices and collecting vital points. If several vital points are located at the same perimeter vertex, we order them according to the order in which the corresponding p-edges are met on a traversal of the perimeter shrunk by some small ϵ. That is we follow a route inside the perimeter which is at a distance ϵ from the perimeter. Next we remove from this list consecutive vital points that are associated with the same hole vertex to satisfy the condition stated in the definition above. The resulting list induces a partition into v-pieces that is compatible with the convex partition. Since we did not restrict the convex partition in any way, this procedure enables us to find a compatible partition into v-pieces for all convex partitions.

Note that the partition into v-pieces generated as described in the preceding paragraph is valid, that is, there are no intersecting vital edges. To see this, consider any two consecutive (with respect to the perimeter) p-edges $\{(v_h, v), (v_h, v')\}$, $v_h \in V_h$, $v, v' \in V_p$. If no p-edge is incident to the perimeter between v and v', then (v_h, v) and (v_h, v') must belong to the same hole-free convex polygon in the partition. Hence the v-piece of v_h, which contains v can extend to any vertex between v and v', including v'.

It is shown in [7] (the full version) that given k hole vertices in a convex polygon P, there are at most $E = \max(2k-2, 1)$ v-pieces in the minimum convex partition. Note however that the precise number of v-pieces is not crucial for the complexity result, it only helps to refine it a little bit. A rougher approximation of its number would suffice to get similar asymptotic upper bounds.

In order to generate all possible partitions into v-pieces (which we have to search), we proceed as described in the following sections.

3.2 Preprocessing Phase

We consider all possible combinations of non-crossing edges going between hole vertices, i.e., all non-crossing super-graphs on the k hole vertices. The total number of such graphs is at most $O((2^3 \cdot 59)^k \cdot k^{-6})$ [4, 18].

As shown pointed out above, it can be shown that given a convex pointgon G, at most 3 v-pieces incident to a hole vertex are sufficient achieve a convex partition (See [7] as well for further explanation). Therefore for each non-crossing graph on the k hole vertices, we allocate a label $i \in \{1, 2, 3\}$ to each non-convex hole vertex, which is meant to indicate how many v-pieces that hole vertex would have in a convex partition if it can be constructed. There are no more than 3^k such labelings for any non-crossing graph.

Since we know the upper bound $E = \max(2k-2, 1)$ on the total number of v-pieces the k hole vertices may have in an MWCP [7], we discard all labelings where the total number of v-pieces is greater that E. To process one labeling, we allocate unique names to each hole's v-pieces as follows: A v-piece name is a tuple (v_h, x), where $v_h \in V_h$ is a hole vertex and $x \in \{a, b, c\}$ distinguishes between the (at most) three v-pieces the

hole vertex v_h has.

We then consider the arrangement of lines, such that for each pair of hole vertices we have a line containing them. We look at all intersections of this arrangement with the perimeter. There are $O(k^2)$ such intersections. Therefore the intersections of these lines with the perimeter partition the perimeter into $O(k^2)$ pieces. We will refer to each such piece as a 'topologically homogeneous perimeter piece' (or 'homogeneous piece' for short). Assume we let the homogeneous pieces of the perimeter $CH(P)$ to be $\{CH(P) = C_0, C_1, \ldots, C_\mu\}$ in counterclockwise order.

For each vital edge (see Definition 1) of a v-piece there can be $O(k^2)$ possibilities as to which homogeneous piece the vital edge should be incident to. However, since the v-pieces are contiguous, that is, neighbouring v-pieces share a vital point, it suffices to allocate one tag C_j, $j = 1, 2, \ldots, \mu$, to each v-piece name (v_h, x), instead of one tag for each vital edge. We define that the tag C_j represents the homogeneous piece the *most clockwise* vital edge of (v_h, x) is incident to. To find the homogeneous perimeter piece the most *counterclockwise* vital edge of a v-piece is incident to, we retrieve the tag of the neighboring v-piece that shares the vital point this vital edge goes to.

A v-piece name assigned to a homogeneous piece C_j is a tuple $((v_h, x), C_j)$. There are $O(k^{4k-4} \cdot 2^{2k-2})$ such assignments to be considered for a given labeling L, since there are at most $2k-2$ v-piece names and each v-piece name can be assigned to $O(k^2)$ homogeneous pieces.

For each homogeneous assignment to v-piece name, we then take an arbitrary point on the perimeter of each homogeneous piece and connect it to all hole vertices which should have a vital edge going to it. Let E' be the set of edges (line segments) created in this way, for all homogeneous pieces. We do this to:
(1) Check for possible edge intersections. (We have no intersections if and only if the set E' together with all edges of the non-crossing supergraph does not lead to any intersections.) If there are edge intersections, we discard the current labeling. (2) Find an ordering Φ of the v-piece names of the current labeling L, corresponding to the counterclockwise ordering in which the vital edges of the v-pieces will appear on the shrunk perimeter if a convex partition is to be constructed. This step takes $O(k \log k)$ time, because it is basically a sorting operation.

Before we start the dynamic programming, which determines whether it is possible to place the p-edges on the perimeter vertices we know: 1) How many v-pieces are associated to each hole vertex. 2) The homogeneous piece a vital edge of a v-piece should go to. 3) The counterclockwise ordering Φ of the v-piece names. 4) All edges connecting hole vertices.

3.3 Dynamic Programming Phase

For each ordering Φ of the v-piece names, we look at coherent subsequences of $1 \leq j \leq 2k-2$ v-piece names. We consider and solve each subsequence of j v-piece names with the condition that either a hole vertex has all its v-piece names (at most three) in this subsequence, or none of them. We will refer to such subsequences as *valid coherent subsequences*. Let C be a coherent piece of the polygon starting at perimeter vertex l and containing m vertices. If we have the most clockwise vital edge for a v-piece i at vertex l and the most counterclockwise vital edge for a v-piece j at $(l+m)$ mod n (the other end of the chain), we want to find the optimal way to place the p-edges on the way counterclockwise from l to $(l+m)$ mod n in order to minimise the length of the corresponding convex partition, if such a convex partition exists. To be precise, we specify each subproblem as a 4-tuple (i, j, l, m), where $i \in 1 \ldots \max(2k-2, 1)$ indicates the position of the first element of the coherent subsequence in the counterclockwise ordered list of v-piece names, $j \in 1 \ldots \max(2k-2, 1)$ represents the number of v-piece names of the coherent subsequence, $l \in 0 \ldots n-k-1$ represents the vertex v_l where the considered perimeter piece starts at, $m \in 1 \ldots n-k$ represents the number of vertices on the considered perimeter piece.

We start with smaller subproblems which are later used to solve larger subproblems, that is, we consider subproblems in the order of increasing j and m.

For each subproblem we store the length of the minimum convex partition if it is possible to obtain a convex partition. Otherwise we store ∞ indicating a convex partition cannot be constructed.

See [7] for how the type of subproblems are solved.

3.4 Analysis

We considered all possible combinations of non-crossing super-graphs on the k hole vertices. There are $O((2^3 \cdot 59)^k \cdot k^{-6})$ number of such non-crossing super-graphs [7]. It takes $O((2^3 \cdot 59)^k \cdot k^{-6})$ time to enumerate all such non-crossing super-graphs [7]. For each non-crossing super-graph G', we considered all the possible labelings of the number of v-pieces corresponding to each hole vertex. There are not more than 3^k such labelings for each non-crossing super-graph. For each labeling L, we considered all possible homogeneous assignments. There are $O(k^{4k-4} \cdot 2^{2k-2})$ such assignments to be considered for a given labeling L. For each assignment, we then obtained the counterclockwise ordering Φ of the corresponding v-piece names in $O(k \log k)$ time. For the ordering Φ of an assignment Π of a labeling L, we then check for possible edge intersections in time polynomial in k and then run the dynamic programming algorithm which takes $O(n^3 k^2)$ time at each time it is called.

This is because, the memory requirement in the worst case is dominated by the $O(n^2 \cdot k^2)$ space for the table entries. We solve directly each subproblem in time $O(n)$ with the help of the smaller subproblems. That is why each call of the dynamic programming algorithm takes $O(n^3k^2)$ time. We call the dynamic programming algorithm at most $O(k^{4k-10} \cdot 2^{13k})$ times. Thus the time taken to solve the MWCP problem is $O((2^3 \cdot 59)^k \cdot (k^{-6}) \cdot 3^k \cdot (k^{4k-4} \cdot 2^{2k-2}) \cdot n^3k^2) = O(n^3 \cdot k^{4k-8} \cdot 2^{13k})$.

It is straightforward to generalize this result to the case where we have as input a PSLG G and k is the total number of holes and/or reflex vertices inside the convex hull of G.[1]

References

[1] P.K. Agrawal, E. Flato, and D. Halperin. Polygon Decomposition for Efficient Construction of Minkowski Sums. *Computational Geometry* 21(1-2):39–61. Elsevier Science, Amsterdam, Netherlands 2002

[2] O. Aichholzer, G. Rote, B. Speckmann, and I. Streinu. The Zigzag Path of a Pseudo-Triangulation. *Proc. WADS*, LNCS 2748:377–388. Springer-Verlag, Berlin, Germany 2003

[3] R. Downey and M. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, NY, USA 1999

[4] A. Garcia, M. Noy, and J. Tejel. Lower Bounds on the Number of Crossing-free Subgraphs of K_N. *Computational Geometry, Theory and Applications* 16:211–221. Elsevier Science, Amsterdam, Netherlands 2000

[5] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to Theory of NP-Completeness.* Freeman, New York, NY, USA 1979

[6] P.D. Gilbert. New Results in Planar Triangulations. *Report R-850.* University of Illinois 1979

[7] M. Grantson. *Fixed-Parameter Algorithms and Other Results for Optimal Convex Partitions.* Licentiate thesis, LU-CS-TR:2004-231, ISSN 1650-1276 Report 152. Lund University, Sweden 2004

[8] M. Grantson, C. Borgelt and C. Levcopoulos. A Fixed Parameter Algorithm for Minimum Weight Triangulation: Analysis and Experiments. Technical Report LU-CS-TR:2005-234, ISSN 1650-1276 Report 154. Lund University, Sweden 2005

[9] J. Keil. *Decomposing a Polygon into Simpler Components.* Ph.D. thesis (Report 163/8). Univ. of Toronto, Toronto, Canada 1983

[10] M. Hoffmann and Y. Okamoto. The Minimum Triangulation Problem with Few Inner Points. *Proc. IWPEC*, LNCS 3162:200–212. Springer-Verlag, Berlin, Germany 2004

[11] J. Keil. Decomposing a Polygon into Simpler Components. *SIAM Journal on Computing* 14:799–817. Society of Industrial and Applied Mathematics, Philadelphia, PA, USA 1985

[12] G.T. Klincsek. Minimal Triangulations of Polygonal Domains. *Annals of Discrete Mathematics* 9:121–123. ACM Press, New York, NY, USA 1980

[13] E. Lodi, F. Luccio, C. Mugnai, and L. Pagli. On Two-Dimensional Data Organization, Part I. *Fundaments Informaticae* 2:211–226. Polish Mathematical Society, Warsaw, Poland 1979

[14] A. Lubiw. The Boolean Basis Problem and How to Cover Some Polygons by Rectangles. *SIAM Journal on Discrete Mathematics* 3:98–115. Society of Industrial and Applied Mathematics, Philadelphia, PA, USA 1990

[15] D. Moitra. Finding a Minimum Cover for Binary Images: An Optimal Parallel Algorithm. *Algorithmica* 6:624–657. Springer-Verlag, Berlin, Germany 1991

[16] W. Mulzer and G. Rote. Minimum-weight Triangulation is NP-hard. *Proc. 22nd Ann. Symp. on Computational Geometry.* to appear, 2006

[17] D. Plaisted and J. Hong. A Heuristic Triangulation Algorithm. *Journal of Algorithms* 8:405–437. Academic Press, San Diego, CA, USA 1987

[18] F. Santos and R. Seidel. A Better Upper Bound on the Number of Triangulations of a Planar Point Set. *arXiv:math.CO/0204045 v2* 2002

[1] The complexity of our algorithm increases as if every reflex vertex were a hole vertex; i.e. k would then be the sum of all holes and the number of reflex vertices on the outer perimeter.