# Mining Convex Polygon Patterns with Formal Concept Analysis

**Aimene Belfodil**[ab], **Sergei O. Kuznetsov**[c], **Céline Robardet**[a], **Mehdi Kaytoue**[a]
[a] Univ Lyon, INSA Lyon, CNRS, LIRIS UMR 5205, F-69621, LYON, France
[b] Mobile Devices Ingenierie, 100 Avenue Stalingrad, 94800, Villejuif, France
[c] National Research University Higher School of Economics, Moscow, Russia
firstname.lastname@insa-lyon.fr , skuznetsov@hse.ru

## Abstract

Pattern mining is an important task in AI for eliciting hypotheses from the data. When it comes to spatial data, the geo-coordinates are often considered independently as two different attributes. Consequently, rectangular shapes are searched for. Such an arbitrary form is not able to capture interesting regions in general. We thus introduce convex polygons, a good trade-off between expressivity and algorithmic complexity. Our contribution is threefold: (i) We formally introduce such patterns in Formal Concept Analysis (FCA), (ii) we give all the basic bricks for mining convex polygons with exhaustive search and pattern sampling, and (iii) we design several algorithms, which we compare experimentally.

## 1 Introduction

Highly focused on over the past 20 years, pattern mining, the task of discovering interesting generalizations of object descriptions (subsets, subsequences, subgraphs, etc) has become a mature subfield in AI for knowledge discovery purposes [Giacometti *et al.*, 2013]. When objects are given with a class label, discriminant patterns enable to elicit hypotheses from the data and to build intelligible classifiers [Zimmermann and Nijssen, 2014].

Dealing with numerical data, and especially spatio-temporal data is still challenging. Algorithms supporting the correct, complete and non-redundant enumeration of particular shapes, say geometrical, have surprisingly not attracted much research interest [Aggarwal and Han, 2014]. Generally, numerical attributes (even spatial) are discretized, either in a pre-processing or on-the-fly during the execution (run) of a pattern enumeration algorithm (e.g. [Grosskreutz and Rüping, 2009]), which has the consequence of considering geo-coordinates attributes independently and thus rectangular shape patterns occur. Still, such patterns were successfully used for mining numerical data (e.g. with most of the subgroup discovery approaches [Duivesteijn *et al.*, 2016]), and spatial data (such as urban and mobility data see e.g. [Bendimerad *et al.*, 2016] and [Kaytoue *et al.*, 2017]).

The problem with rectangular shapes can be observed on the right hand side figure. Each object gives a POI (Point Of Interest) of a given type (Hotel, Restaurant, University, ...) and position. An interesting pattern is understood as a geographical area for which there is a sufficient number of points, high density, and a high proportion of objects of the same type. The candidate areas could have any shape. Rectangles, as being the products of intervals, have edges parallel to the plane axes: they may enclose both dense and sparse regions. Arbitrary polygons stick too much to the data and are hard to interpret. We consider convex polygons, a good trade-off for capturing high density areas.

Our contribution introduces a new type of patterns, convex polygons, with FCA tools [Ganter and Wille, 1999], going beyond the formalization of hyper-rectangles given by [Kuznetsov, 2005] in the early 1990s and introduced in pattern mining by [Kaytoue *et al.*, 2011]. We thus make precise the well-known notion of Galois connection as well as several polygon enumeration techniques and associated algorithms, using several concepts from computational geometry. We introduce several polygon constraints and experiment with our algorithms. We show that polygons give a better trade-off between area, density and homogeneity for mining spatial data. These findings give the basic bricks for any pattern mining algorithm dealing with, among others, a spatial attribute. The major problem with polygons is their worst-case exponential number (in number of input points). This is probably why they were not used in pattern mining until now: Exhaustive enumerations fail at considering large datasets even with 100 objects. We finally show that embedding our enumeration techniques in a recent pattern sampling technique (Monte Carlo Tree Search) enables us to discover high quality patterns very quickly in large datasets.

In what follows, Section 2 recalls how to properly define and enumerate hyper-rectangles with FCA. Section 3 defines polygon patterns, while Section 4 details our algorithms. Before to conclude, in Section 5 we present a wide range of experiments to support our claims.

## 2 Interval patterns

We recall first the formalization of interval patterns, or hyper-rectangles, in terms of FCA, for understanding next our formalization of convex polygon patterns.

**Numerical dataset.** A numerical dataset is given by a set of objects $G$, a set of numerical attributes $M = \{m_i\}_{1\leq i\leq |M|}$, where the range of $m_i \in M$ is a finite set denoted by $W_{m_i}$, $m_i(g) = w$ means that $w$ is the value of attribute $m_i$ for object $g \in G$. Figure 1 plots 5 objects with 2 attributes on the Euclidean plane.

**Interval pattern.** An interval pattern is a box (hyper-rectangle) with sides *parallel to coordinate axes* formally defined as the Cartesian product of intervals $d = \langle [a_i, b_i]\rangle_{1\leq i\leq |M|}$. An object $g$ is in the image of an interval pattern $d$ when $m_i(g) \in [a_i, b_i]\ \forall i \in [\![1, |M|]\!]$. The support of $d$, denoted by $sup(d)$, is the set of objects in the image of $d$.

**Interval pattern search space.** The search space of interval patterns is the finite set $D$ of all interval vectors $\langle [a_i, b_i]\rangle_{1\leq i\leq |M|}$ with $a_i, b_i \in W_{m_i}$. The size of the search space is given by: $|D| = \prod_{i\in[\![1,|M|]\!]}(|W_{m_i}| \times (|W_{m_i}|+1)/2)$.

Many patterns in $D$ have exactly the same support: different hyper-rectangles contain the same objects. To avoid this redundancy, a closure operator can be defined, and only closed patterns which are unique for a given support are retained. This is achieved thanks to the formalism of pattern structures introduced by [Ganter and Kuznetsov, 2001]. An *interval pattern structure* is given by $(G, (D, \sqcap), \delta)$ where $G$ is the set of objects, $(D, \sqcap)$ the semi-lattice of object descriptions (boxes) and $\delta : G \to D$ a mapping that associates to each object $g \in G$, a vector of numerical intervals in the form of one-point interval $\delta(g) = \langle [m_i(g), m_i(g)]\rangle_{i\in[\![1,|M|]\!]}$. Elements of $D$ are called *patterns* and are ordered as follows: $c \sqcap d = c \Leftrightarrow c \sqsubseteq d$. The *infimum* $\sqcap$ is defined as follows. Let $c = \langle [a_i, b_i]\rangle_{i\in[\![1,|M|]\!]}$ and $d = \langle [e_i, f_i]\rangle_{i\in[\![1,|M|]\!]}$ we have $c \sqcap d = \langle [min(a_i, e_i), max(b_i, f_i)]\rangle_{i\in[\![1,|M|]\!]}$ which is the minimal bounding box of the two boxes $c$ and $d$. The two following operators $(.)^{\square}$, with $A \subseteq G$ and $d \in (D, \sqcap)$

$$d^{\square} = \{g \in G \mid d \sqsubseteq \delta(g)\} \quad A^{\square} = \sqcap_{g\in A}\, \delta(g)$$

form a Galois connection between $(2^G, \subseteq)$ and $(D, \sqsubseteq)$. $(.)^{\square\square}$ is a closure operator, i.e., monotone, extensive, idempotent. Closed patterns, i.e., such that $d = d^{\square\square}$ are smallest patterns/rectangles for a given support. We note that, for any description (box) $d \in D$, $d^{\square\square}$ represents the *minimal bounding box* of the support of $d$ ($d^{\square}$, objects enclosed by $d$). Figure 1 (left; middle) shows an example where $G = \{g_1, ..., g_5\}$, we have $d = \langle [1, 2], [1, 4]\rangle$ which support is $d^{\square} = \{g_1, g_2, g_3\}$. However $d$ is not closed. Indeed, $d^{\square\square} = \{g_1, g_2, g_3\}^{\square} = \langle [1, 2], [1, 3]\rangle$ which represents its closure (the minimal bounding box of $\{g_1, g_2, g_3\}$).

**Interval patterns enumeration.** Consider first data with a single attribute. To enumerate all interval patterns in $(D, \sqcap)$, [Kaytoue *et al.*, 2011] started from the top pattern, which is the minimal bounding box of all objects in $G$ ($G^{\square}$). Then, as shown in Figure 2, at every step of the algorithm two minimal changes are applied (minimal left change (`minLeftChange`) and minimal right change (`minRightChange`)). To ensure a non-redundant generation, `minLeftChange` are not allowed after `minRightChange`. For $|M|$ numerical attributes, the algorithm is the same with two differences: (1) It considers a total order on the set of attributes $M$; (2) When a minimal change is applied to the attribute $m_i$, only attributes $m_j >= m_i$ can be refined in further steps from the generated pattern.

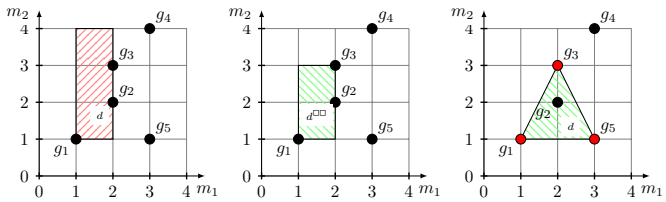



Figure 1: Non-closed (left) and closed (middle) interval pattern; convex polygon pattern (right).

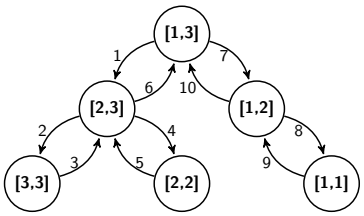



Figure 2: Depth-first traversal of $(D_{m_1}, \sqcap)$.

**Closed interval patterns enumeration.** To enumerate only closed patterns in $(D, \sqcap)$ (minimal bounding boxes), [Kaytoue *et al.*, 2011] adapted *CloseByOne* of [Kuznetsov, 1993]. Consider a pattern $d$ generated by a change at attribute $m_j \in M$. Its closure is given by $d^{\square\square}$. If $d^{\square\square}$ differs from $d$ for some attributes $m_i \in M$ such as $m_i < m_j$, then $d^{\square\square}$ has already been generated: the algorithm backtracks. Otherwise, it continues the enumeration from the closed pattern $d^{\square\square}$.

## 3 Convex polygon patterns

The choice of convex polygons instead of arbitrary ones is motivated by the fact that (i) it generalizes intervals (convex) and (ii) it is a natural way to avoid non-convex polygons which would over-fit the data.

### 3.1 Preleminaries

**Convex polygon.** A *convex polygon* $P$, represented as a sequence of points $[p_1, ..., p_h]$ in $\mathbb{R}^2$, is a simple polygon (not self-intersecting) where all interior angles are strictly less than $\pi$. The ordered point sequence $[p_1, ..., p_h]$ is denoted by $\overline{P}$ and is given in *counterclockwise order (ccw)*. Points $p_i$ are called *extreme points* and *oriented line segments* $p_i p_{i+1}$ following this order are called *edges* of the polygon $P$ (where $i + 1 = 1$ if $i = h$ and $i - 1 = h$ if $i = 1$).

Note that an oriented line segment $AB$ subdivides $\mathbb{R}^2$ in 4 regions: (i) $AB^+$ (resp. (ii) $AB^-$) the open upper (resp. lower) half-plane of $AB$ (*i.e.* $Q \in AB^+$ implies that $ABQ$ is a triangle in *ccw* (resp. *cw*) order), (iii) $AB^0$ the set of points on the segment $AB$ and (iv) the points that are collinear with $A$ and $B$ but outside $AB$.

Let $q \in \mathbb{R}^2$. An edge $p_ip_{i+1}$ is said to be *visible from* $q$ (or $q$ sees $p_ip_{i+1}$) iff $q \in p_ip_{i+1}^-$. A point $q$ is in the enclosed area of the polygon $P$ iff $q$ does not see any edge of $P$ ($q \in p_ip_{i+1}^+ \forall i \in \{1, ..., h\}$). Conventionally, $\varnothing$, points and segments are considered as convex polygons.

**Convex hull.** Given a finite set $E \subseteq \mathbb{R}^2$, the *convex hull* of $E$ denoted by $ch(E)$ is the smallest convex set that contains $E$, which is a *convex polygon*. Note that $\overline{ch(E)} \subseteq E$. In other words, extreme points of the *convex hull* of $E$ are in $E$. Application $ch : \mathbb{R}^2 \to \mathbb{R}^2$ is monotone, extensive, and idempotent, i.e., a closure operator.

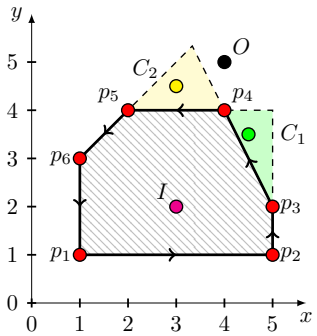


Figure 3: Convex polygon and edges visibilities.

**Example.** Figure 3 shows a polygon $P$ where $\overline{P} = [p_1, p_2, p_3, p_4, p_5, p_6]$. The point $I$ is in the area enclosed by the polygon $P$. The point $C_1$ sees only the edge $p_3p_4$. Conversely, the green triangle represents all possible points that are only visible by the edge $p_3p_4$. The point $O$ sees the two edges $p_3p_4$ and $p_4p_5$.

### 3.2 Convex polygon pattern structure

**Spatial attribute.** A spatial attribute $m$ takes values in $\mathbb{R}^2$. Given a set of objects $G$, $m(g) = p$ means that $p \in \mathbb{R}^2$ is the value of attribute $m$ for object $g \in G$. Analogically, for a subset $A \subseteq G$, $m(A) = \{m(g) \mid g \in A\} \subset \mathbb{R}^2$. For the sake of simplicity, we will consider now a dataset $G$ having only *one spatial attribute* $m$. For ease of notations, $g$ and $m(g)$ ($G$ and $m(G)$) will be sometime confused in what follows.

**Convex polygon pattern.** A convex polygon pattern $d$ is a convex polygon $[p_i]_{1 \le i \le h}$ given in *ccw* order. An object $g$ is in the image of a convex polygon pattern $d$ when $m(g)$ is in the enclosed area formed by the polygon $d$. The support of $d$, denoted by $sup(d)$, is the set of objects $g \in G$ in the image of $d$. In Figure 1 (right), $d = [g_1, g_5, g_3]$ is a convex polygon pattern which support is $sup(d) = \{g_1, g_2, g_3, g_5\}$.

**Convex polygon pattern structure.** It is defined as follows: $(G, (D, \sqcap), \delta)$ where $G$ is the set of objects described by one spatial attribute $m$, $(D, \sqcap)$ is the semi-lattice of object descriptions (convex polygons) where $D = \{\overline{ch(m(A))} | A \subseteq G\}$. The mapping $\delta : G \to D$ takes each object $g \in G$ to $\delta(g) = [m(g)]$ (a degenerate polygon with a singleton point). The *infimum* $\sqcap$ is defined as follows. Let $c$ and $d$ be two descriptions in $D$, we have $c \sqcap d = \overline{ch(c \cup d)}$. Elements of $(D, \sqcap)$ are ordered as follows: $c \sqsubseteq d \Leftrightarrow c \sqcap d = c \Leftrightarrow d \subseteq ch(c)$. Thus, the Galois operators $(.)^\square$ are defined as follows, with $A \subseteq G$ and $d \in (D, \sqcap)$: $d^\square = \{g \in G \mid m(g) \in ch(d)\}$ and $A^\square = \overline{ch(m(A))}$. A pair $(A, d)$ s.t. $A^\square = d$ and $d^\square = A$ is a concept. Both $A$ and $d$ are closed under $(.)^{\square\square}$. The set $A$ contains all objects, while $d$ contains only extreme points in *ccw* order.

**Mining convex polygons with constraints.** There are $2^n$ convex polygons in the worst case ($n$ points on a circle). Not all of them are interesting, we thus define several constraints a pattern shall respect. The problem is then, given a set of points in $\mathbb{R}^2$, to find all polygon pattern concepts $(A, d)$ respecting one or more of the following constraints, such as shape complexity ($|d| \le \delta_{compl}$), minimal support ($|A| \ge \delta_{supp}$), minimal/maximal perimeter ($perim(d) \ge$ or $\le \delta_{perim}$), and minimal/maximal area ($area(d) \ge$ or $\le \delta_{area}$). We also wish patterns that maximize a class homogeneity (e.g. low Gini) and density ($|A|/area(d)$).

By using minimal support we avoid considering polygons with too few points. The density has to be understood as a relative support (support normalized by area). The number of extreme points characterizes the complexity of the polygon in terms of interpretation (point, segment, triangle, quadrilateral, ...). The simpler the form the better by principle of parsimony: Minimizing the shape complexity may also avoid over-fitting the data when searching for discriminant patterns. Controlling perimeter and area is also important: It allows one to express different types of patterns (e.g. avoiding or forcing thin polygons, with a large perimeter and small surface).

## 4 Algorithms

Let $G$ be a finite subset of $\mathbb{R}^2$. We propose three algorithms for mining convex polygon patterns. The first one enumerates object subsets in a bottom-up way (from $\varnothing$ to $G$) with closures. The next one considers a top-down enumeration and does not compute costly closures. The last one considers a bottom-up enumeration of polygons by shape complexity (points, then segments, then triangles...) and performs the best.

### 4.1 Enumerating point sets

As the operators $((.)^\square, (.)^\square)$ form a Galois connection, the set of all pattern concepts is given by $\mathcal{C} = \{(A^{\square\square}, A^\square), \forall A \subseteq G\}$. The $A^\square$ are precisely the polygon patterns and can thus be obtained by enumerating closed object subsets with the generic algorithm *Close-ByOne (CbO)* of [Kuznetsov, 1993] and its modern efficient software realization [Andrews, 2015]. The principle is the following. The lattice $(2^G, \subseteq)$ is explored with a DFS starting from $\varnothing$. A total order on $G$ is provided,

e.g. $g_1 < g_2 < ... < g_n$. A new object set $A$ is generated from a previous one by adding next object $g$ w.r.t. $<$ and $(A^{\square\square}, A^{\square})$ is the resulting pattern concept. The concept is discarded (DFS backtrack) if an object smaller than $g$ w.r.t. $<$ is added. It is proven that this algorithm outputs the correct, complete and non-redundant collection of closed patterns. Namely, ExtCbO (Algorithm 1) is the version of CbO where concepts are computed w.r.t. increasing generality order, i.e., by adding one object at a time. Algorithm ExtCbO will serve as a baseline.

**Algorithm 1** ExtCbO

```
1: Let (G, ≤) be a totally-ordered finite subset of ℝ²
2: procedure ExtCbO( )
3:     traverse(∅,∅,1)
4: end procedure
5: procedure traverse(c,d,pos)          ▷ c is the extent, d the intent
6:     Print(c,d)                       ▷ visit pattern concept (c, d)
7:     for i ∈ pos, ..., |G| and g_i ∉ c do
8:         c_new ← c ∪ {g_i}
9:         d_new ← c_new^□              ▷ Compute convex hull
10:        c_new ← d_new^□
11:        if ∀j ∈ [[1, i]] : g_j ∉ c → g_j ∉ c_new then
12:            traverse(c_new,d_new,i+1)
13:        end if
14:    end for
15: end procedure
```

## 4.2 Updating a Delaunay triangulation

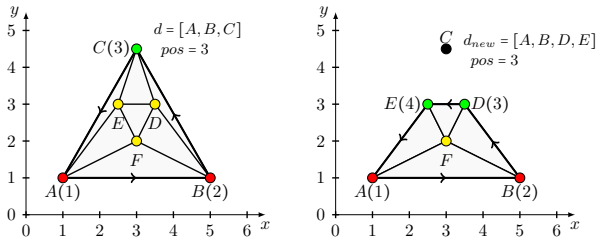
The enumeration starts from the most general pattern $G^{\square}$. Minimal changes are applied to obtain the next pattern to ensure the correctness and completeness of the enumeration: all convex polygons and only them are visited. Let us consider the convex polygon pattern $d = [p_1, ..., p_i, ..., p_{|d|}]$. A minimal change to $d$ consists in removing an extreme point (as done with interval patterns). There are $|d|$ of such minimal changes for a polygon $d$ for obtaining next smaller polygons $e = d\backslash\{d[i]\}$ s.t. $d \sqsubset e$, $i \in [\![1, |d|]\!]$. When a new pattern $e$ is generated, we need to compute its convex hull to discover its extreme points $d_{new} = e^{\square\square}$ and continue the enumeration.

This algorithm is again an instance of *CloseByOne*, but follows a top-down enumeration. Sadly, it still forces to compute the convex hull $e^{\square}$ at each step. Fortunately, it can be avoided with a *Delaunay triangulation*.

**Delaunay triangulation.** The *Delaunay triangulation* of a set of points $P$ denoted by $DT(P)$ is a partition of the *convex hull* $ch(P)$ into $\mathcal{O}(n)$ triangles such that: *(i)* Triangles vertices are in $P$, *(ii)* No point in $P$ is inside the circumcircle of any triangle in $DT(P)$ besides the vertices of the triangle. The complexity of computing $DT(P)$ is $\mathcal{O}(|P| \cdot log(|P|))$ [Zalik, 2005].

**Algorithm** DelaunayEnum**.** The key idea is the following. Efficiently computing $e^{\square} = (d\backslash\{d[i]\})^{\square}$, that is the next closed pattern obtained from removing the extreme point $i$ of $d$, is equivalent to computing $DT(d \setminus \{d[i]\})$ from $DT(d)$. During this computation, one can easily updates the sequence of extreme points. [Devillers, 1999] answered this problem with an efficient algorithm ($\mathcal{O}(k \cdot log(k))$ complexity), where $k$ is the number of points sharing an edge with $p$ (neighbor points).

Our algorithm DelaunayEnum (Algorithm 2) enumerates *Delaunay triangulations* $dt$ and maintains at each step the sequence of extreme points $d$. It starts from pattern $G^{\square} = \overline{ch(G)}$ (line 4). Consider a step in the enumeration where $d$ is the current pattern, $dt$ the current triangulation and $c$ the current image of $d$. We remove successively extreme points $p \in d$ and we update the Delaunay triangulation $dt_{new}$ based on $dt$ (line 11), the new description $d_{new}$ (sequence of extreme points) (line 12) and the new support $c_{new}$ by removing objects with value $p$ from $c$ (line 13). To avoid redundancy (avoid visiting the same pattern twice), removal of extreme points before the last removed extreme point $p$ are not allowed (argument *pos* in Algorithm 2) in further steps. In order to do that simply, when the extreme point sequence $d$ is updated upon extreme point $i$ removal, we only insert the new extreme points (extreme points not in $d$) at the position $i$ while keeping the same order (*counter-clockwise*). The figure below (from left to right) shows an example of how the algorithm updates the description $d = [A, B, C]$ upon removal of the extreme point $C$ (position 3) and produces the description $d_{new} = [A, B, \underline{D}, \underline{E}]$ where $\underline{D}$ and $\underline{E}$ denote the new extreme points that replaced $C$. The enumeration continues by removing $D$.




**Algorithm 2** DelaunayEnum

```
1: Let G be a finite subset of ℝ²
2: procedure DelaunayEnum( )
3:     dt ← Delaunay(G)
4:     d ← dt̄                          ▷ dt̄ = G^□ = ch(G)
5:     traverse(G,d,dt,1)
6: end procedure
7: procedure traverse(c,d,dt,pos)
8:     Print(c,d)                      ▷ visit pattern concept (c, d)
9:     for i ∈ pos, ..., |d| do
10:        p ← d[i]
11:        dt_new ← Delaunay_remove(dt,p)
12:        d_new ← dt_new^-            ▷ d_new[1] = d[1] or d_new[|d_new|] = d[|d|]
13:        c_new ← c\[p]^□             ▷ [p]^□ = {g ∈ G | m(g) = p}
14:        traverse(c_new,d_new, dt_new,i)
15:    end for
16: end procedure
```

## 4.3 Enumerating simpler shapes first

Until now, elements of $(D, \sqsubseteq)$ were enumerated w.r.t. polygon inclusion order. Now we consider the *poset* $(D, \bar{\sqsubseteq})$, where polygons are ordered w.r.t. extreme points inclusion: $c \bar{\sqsubseteq} d \Leftrightarrow c \supseteq d$. Intuitively, the $i^{th}$ level of $(D, \bar{\sqsubseteq})$ contains all polygons with $i$ extreme points ($i = 3$ are the triangles, $i = 4$ the convex quadrilaterals, ...). Enumerating simpler shapes first is interesting as they are easier to interpret and less stick to the data points

(prevent overfitting). Note that the new order $\bar{\sqsubseteq}$ will change only the enumeration order.

In order to enumerate all the polygons w.r.t. the new order $\bar{\sqsubseteq}$ in a *bottom-up* fashion as simpler shapes are preferred, one can simply adapt ExtCbO. Indeed, polygons are represented as extreme points sets which can be enumerated starting from $\varnothing$. However, it requires to compute at each step the convex hull of the set of points (closure) which is extremely slow. To avoid this, we propose an enumeration technique relying on basic geometry: points used to extend a pattern produce a *new closed pattern* for sure. That is to say: (i) there is no need to compute the convex hull and (ii) there is no need to discard a pattern as done in ExtCbO since every generated pattern is new. This can be done thanks to *pattern candidate maintenance* as explained below.

In what follows, without loss of generality and for the sake of simplicity, elements of $G$ are pairwise distinct. Moreover, for any oriented line segment $AB$: $AB^+$, $AB^-$ and $AB^0$ will denote finite sets where only points of $G$ are considered (rather than $\mathbb{R}^2$).

**Pattern candidate points array.** Consider a convex pattern $d = [p_1, ..., p_h]$ where $\forall i \in [\![1, h]\!] : p_i \in G$. We define a same-size array $n_d$ called *candidate points array* where $n_d[i]$ gives the set of points that are *visible from and only from* the edge $p_i p_{i+1}$. Formally, if $h \geq 2$:

$$n_d[i] = p_i p_{i+1}^- \cap \left(\bigcap_{j \neq i} p_j p_{j+1}^+\right)$$

In case of $h = 1$, we have $n_d[1] = G \backslash \{p_1\}$. The proposition below gives a simpler formula for $n[i]$ when $h \geq 2$:

**Proposition 1.** *If $h \geq 2$, $\forall i \in [\![1, h]\!]$ we have:*

$$n_d[i] = p_i p_{i+1}^- \cap p_{i-1} p_i^+ \cap p_{i+1} p_{i+2}^+$$

**Extending a pattern and candidate maintenance.** We have the following proposition:

**Proposition 2.** *$\forall i \in [\![1, h]\!]$, $\forall q \in n_d[i]$, we have:*

$$d \sqcap [q] = [p_1, p_i, q, p_{i+1} ..., p_h]$$

*In other words, extending a pattern $d$ by adding a candidate point $q$ creates a new pattern $e = d \sqcap [q]$ s.t. $e \bar{\sqsubseteq} d$.*

Proposition 3 gives a method to compute the support and the candidate points of the new pattern $e$:

**Proposition 3.** *If $h \geq 2$, $\forall i \in [\![1, h]\!]$, $\forall q \in n_d[i]$, if $e = d \sqcap [q]$ and $n_e$ its candidate points array, we have:*

$$\begin{aligned}
e^{\square} &= d^{\square} \cup [p_i, q, p_{i+1}]^{\square} \\
n_e[i] &= n_d[i] \cap p_i q^- \cap q p_{i+1}^+ \\
n_e[i+1] &= n_d[i] \cap q p_{i+1}^- \cap p_i q^+ \\
n_e[i-1] &= n_d[i-1] \cap p_i q^+ \\
n_e[i+2] &= n_d[i+1] \cap q p_{i+1}^+ \\
n_e[i+k] &= n_d[i+k-1] \text{ if } 3 \leq k \leq h-i+1 \\
n_e[i-k] &= n_d[i-k] \text{ if } 2 \leq k < i
\end{aligned}$$

*Note that $[p_i, q, p_{i+1}]^{\square}$ is the subset of objects enclosed in the area formed by the triangle $[p_i, q, p_{i+1}]$. Formally: $[p_i, q, p_{i+1}]^{\square} = p_i q^0 \cup q p_{i+1}^0 \cup p_i p_{i+1}^0 \cup (p_i p_{i+1}^- \cap p_i q^+ \cap q p_{i+1}^+)$.*

Propositions 1 and 2 are direct results from planar geometry [Overmars and van Leeuwen, 1981], while Proposition 3 is a direct reformulation of Proposition 1.

Figure 3 shows a description $d = [p_1, p_2, p_3, p_4, p_5, p_6]$ with candidate points $n_d = [\varnothing, \varnothing, \{C_1\}, \{C_2\}, \varnothing, \varnothing]$. More generally, every point that falls in the green (resp. yellow) zone is a candidate point for the edge $p_3 p_4$ (resp. $p_4 p_5$). However, the point $O$ is not a candidate point for any edge. Indeed, $O$ sees two edges $p_3 p_4$ and $p_4 p_5$.

**Algorithm** ExtremePointsEnum**.** The algorithm ExtremePointsEnum (Algorithm 3) follows almost the same enumeration principle as that of ExtCbO. The difference lies in maintaining the list of object candidates that can be added to a pattern $d$ to generate patten $e \sqsubset d$. The other difference is that the two first levels are enumerated in BFS manner. This allows one to build the segment index $S$ that for each distinct object pair $g_i g_j$ of $G$ stores the support $g_i g_j^0$ (objects in segment $g_i g_j$) and its candidates $g_i g_j^-$ and $g_i g_j^+$. Note that the segment index $S$ can be seen as a *strictly upper triangular matrix*. The algorithm continues in a *DFS-fashion* to enumerate higher levels ($k \geq 2$). To prevent redundancy, an arbitrary total order on $G$ is provided, e.g. $g_1 < g_2 < ... < g_n$. When extending a pattern $d = [p_1, ..., p_h]$, which candidate points array is $n_d$, with a point $q \in n_d[i]$, only points $q$ s.t. $p_j < q \ \forall j \in [\![1, h]\!]$ are considered.

**Algorithm 3** ExtremePointsEnum

```
1: Let (G, ≤) be a totally-ordered set of pairwise distinct points
2: Let S be the segment index
3: procedure TRAVERSE(c, d, n, pos)
4:     Print(c,d)                                  ▷ visit pattern concept (c, d)
5:     for i ∈ 1, ..., |d| do
6:         for j ∈ n[i] and j >= pos do   ▷ j ≥ pos for non redundancy
7:             s1 ← S[d[i]][j]                     ▷ first new segment d[i] → j
8:             s2 ← S[j][d[i + 1]]   ▷ second new segment j → d[i + 1]
9:             d_new ← [d[1], ..., d[i], j, d[i + 1], ..., d[|d|]]
10:            c_new ← c ∪ s1^0 ∪ s2^0 ∪ (n[i] ∩ s1^+ ∩ s2^+)
11:            n_new ← [n[1], ..., n[i], n[i], n[i + 1], ..., n[|d|]]
12:            n_new[i] ← n[i] ∩ s1^- ∩ s2^+
13:            n_new[i + 1] ← n[i] ∩ s2^- ∩ s1^+
14:            n_new[i − 1] ← n[i − 1] ∩ s1^+
15:            n_new[i + 2] ← n[i + 1] ∩ s2^+
16:            TRAVERSE(c_new, d_new, n_new, j + 1)
17:        end for
18:    end for
19: end procedure
20: procedure ExtremePointsEnum( )
21:    Print(∅,∅)                  ▷ visit the empty pattern concept (∅, ∅)
22:    Enumerate in BFS-fashion all distinct points G
23:    Compute segment index S
24:    for i ∈ 1, ..., |P| − 1 do
25:        for j ∈ i + 1, .., |P| do
26:            s ← S[i, j]
27:            TRAVERSE(s^□, [i, j], [s^-, s^+], j+1)
28:        end for
29:    end for
30: end procedure
```

**Detailed example.** Figure 5 gives a detailed step-by-step partial enumeration of convex patterns related to $G$ w.r.t. $\bar{\sqsubseteq}$ partial order. In each subfigure, red points are *extreme points*, yellow points are part of the support of the pattern but not extreme points, green points are candidate points and black points are not candidates (points located in the red zones).

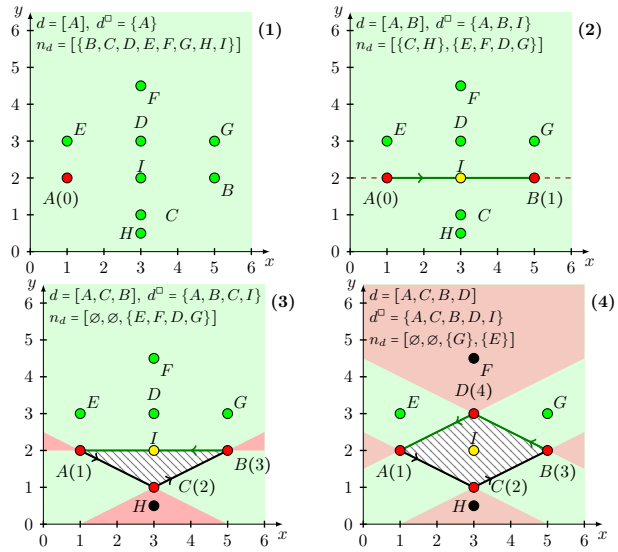



Figure 5: ExtremePointsEnum algorithm step-by-step enumeration.

## 5 Experiments

We report an experimental study of the different algorithms, carried out on a machine equipped with Intel Core i7-2600 CPUs 3.4 Ghz machine with 16 GB RAM. Algorithms are implemented in Java. All materials are available on `https://github.com/BelfodilAimene/MiningConvexPolygonPatterns`.

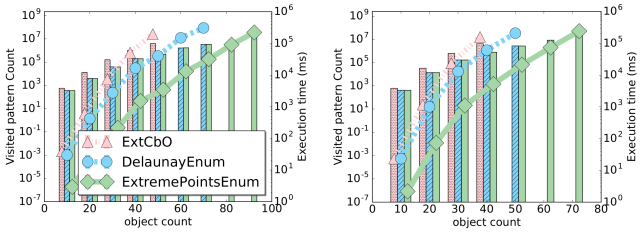
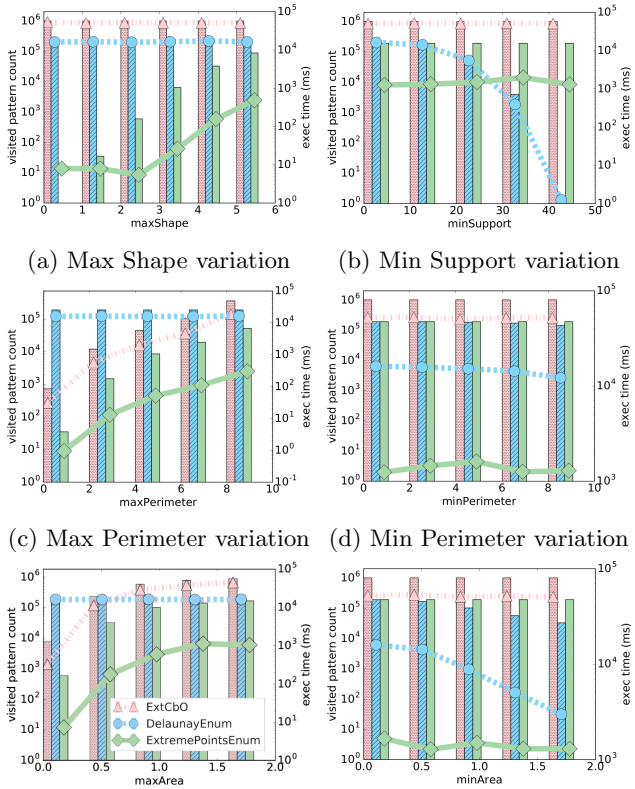



Figure 6: Polygon pattern enumeration performance comparison. IRIS sepal-length × sepal-width (left). IRIS petal-length × petal-width (right).

**Mining polygon patterns.** We compare our three algorithms without any constraint: ExtCbO, DelaunayEnum and ExtremePointsEnum. Figure 6 plots for each one their run times and the number of pattern candidates they generated. Datasets consist in $n$ objects drawn from the IRIS dataset uniformly from the three different classes for the attributes sepal-length and sepal-width (or petal-lentgh and petal-width). First, notice that ExtCbO generates a lot of candidates discarded by the canonicity test (redundant), while the two others generate each pattern only once. It follows that ExtCbO is from one to two orders of magnitude slower (it is the only one computing closures). Interestingly, ExtremePointsEnum is faster than DelaunayEnum as it does not require to compute and update a Delaunay triangulation (even when the state-of-the-art [The CGAL Project, 2016] is used).

**Impact of the constraints.** ExtCbO enumerates convex polygons in a bottom-up fashion w.r.t. inclusion. It can thus only handle maximum perimeter and area constraints that are monotone (the proof is given, e.g. by [Bogomolny, 2017]): when a pattern is generated and does not satisfy the constraint, the algorithm backtracks (a well-known property in pattern mining). DelaunayEnum enumerates convex polygons in a top-down fashion w.r.t. inclusion. It can thus naturally prune w.r.t. minimal support, area and perimeter. ExtremePointsEnum enumerates patterns by inclusion but also from simpler to more complex shapes (extreme points inclusion). It can thus handle maximum shape complexity, perimeter and area constraints.

Figure 7 reports the run time of our three algorithms on the IRIS Sepal length vs. Sepal Width dataset when introducing each constraint separately and varying the associated threshold (min. and max. perimeter are computed as they have the same behavior as min. and max. area, respectively). It also reports the number of generated patterns: The lower, the better. Some algorithms output more patterns as they cannot efficiently handle the constraints (such invalid patterns need to be removed during post-processing). As such, depending on the constraints the user is interested in, one algorithm may be preferred to another.




(a) Max Shape variation (b) Min Support variation

(c) Max Perimeter variation (d) Min Perimeter variation

(e) Max Area variation (f) Min Area variation

Figure 7: Run time and generated patterns count for our three algorithms when introducing constraints.

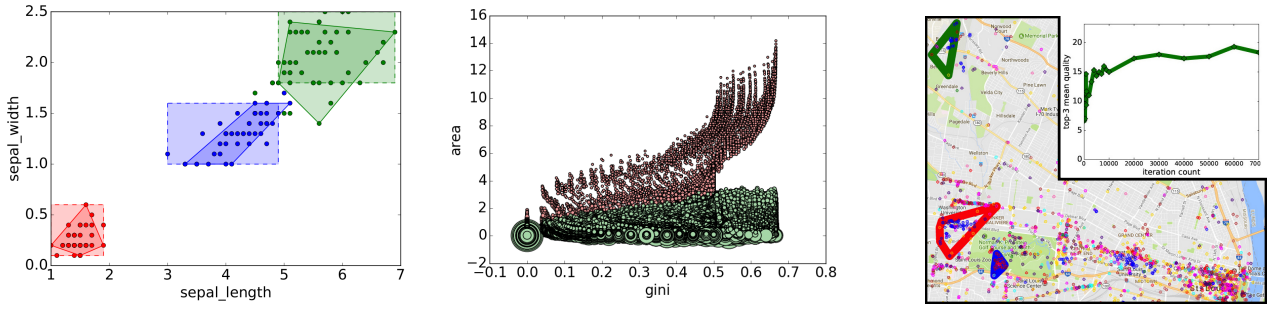



Figure 8: Comparing interval and convex polygon top-3 patterns (Left). Comparing interval and convex polygon patterns gini, area and density (Center). Top-3 homogeneous convex polygon patterns on Foursquare places from Saint-Louis (Missouri, USA) (Right).

**Intervals vs. polygons.** Our main motivation for introducing convex polygon patterns is to discover shapes with high density and area, and possibly with high class homogeneity (e.g., low Gini). Figure 8 (left) considers the IRIS dataset (Sepal length vs. Sepal Width). It presents the three most frequent polygons that have null Gini, and either 3 or 4 extreme points for a fair comparison: convex polygons better stick to the data without extremely over-fitting.

We also compare interval and polygon patterns in several datasets and plot their area, density and Gini. Figure 8 (center) plots all discovered patterns area (Y), Gini (X), and density (point diameter). It appears that convex polygons enable to find shapes with higher density, yet smaller area, over the same Gini range. Rectangles with high area are exactly those that we want to avoid for spatial data: they have high chances to enclose both zones of high and low density, and high impurity.

**Sampling polygon patterns with MCTS.** Our algorithms are clearly not scalable as their complexity depends on the number of polygons which can be very large. We propose to sample the pattern search space with a recent technique introduced by [Bosc *et al.*, 2016] relying on Monte Carlo Tree Search [Browne *et al.*, 2012]. Without entering into the details, MCTS requires a proper enumeration technique: we choose ExtremePointsEnum as it is generally the most efficient. MCTS iteratively draws a random pattern, following a path of ExtremePointsEnum: the best pattern quality measure is returned as a reward. This reward is used to update a memory (the Monte Carlo tree) and drives the search for the next iterations (thanks to the upper confidence bound, a formula that expresses a trade-off between exploration and exploitation of the search space). If given enough budget (maximum number of iterations allowed), MCTS will perform an exhaustive search. The quality of a pattern $p$ is given by $Q(p) = 2 \cdot max\{sup(p,i)\} - |sup(p)|$ where $sup(p,i)$ is made of the points of $sup(p)$ belonging to the $i^{th}$ class. This choice favors polygons with high support and class homogeneity, but one could choose any other measure.

[Falher *et al.*, 2015] propose several spatial datasets (cities) containing *Foursquare* users check-in places of different kinds ("College & university", "Art & Entertainement", ...). We randomly choose the city of Saint-Louis (Missouri, USA) containing 3 464 points. Running our exhaustive algorithms on this dataset is impossible. The MCTS sampling however quickly returns patterns of high quality: Figure 8 (right) displays the average pattern quality measure of the best patterns (after a redundancy post-processing as done by [Bosc *et al.*, 2016]): a plateau is reached with a maximum budget of $20K$ iterations (about 2 minutes). Note that each point corresponds to a different execution with a different budget. Figure 8 (right) presents the top-3 patterns, which are homogeneous zones with a large support. Designing a quality measure can be achieved in many ways (e.g., involving density).

## 6 Conclusion

In pattern mining, hyper-rectangles cannot always properly capture interesting areas although they are widely used. We formally defined convex polygon patterns by means of FCA and proposed three enumeration techniques. Although these algorithms do not scale, they serve as a basis for efficient pattern sampling approaches, which are shown to be scalable to very large pattern spaces and data. Tuning MCTS sampling techniques remains to be a subject of our further study. We also plan to extend our approach to other convex forms, like circles and ellipses, as well as to apply this approach in machine learning settings (pattern-based classifier).

**Acknowledgment.** This work has been partially supported by the European project GRAISearch (FP7-PEOPLE-2013-IAPP) and the ANRt French program. Sections 2 and 3 were written by Sergei O. Kuznetsov supported by the Russian Science Foundation under grant 17-11-01294 and performed at National Research University Higher School of Economics, Russia. The authors would like to warmly thank David Coeurjolly, Eric Guérin, Clément Jamin, Mikhail Vyalyi and Adnene Belfodil for interesting discussions on computational geometry among others.

## References


[Aggarwal and Han, 2014] Charu C. Aggarwal and Jiawei Han, editors. *Frequent Pattern Mining*. Springer, 2014.

[Andrews, 2015] Simon Andrews. A 'Best-of-Breed' approach for designing a fast algorithm for computing fixpoints of Galois Connections. *Inf. Sci.*, 295:633–649, 2015.

[Bendimerad *et al.*, 2016] Anes Bendimerad, Marc Plantevit, and Céline Robardet. Unsupervised Exceptional Attributed Sub-Graph Mining in Urban Data. In *IEEE 16th International Conference on Data Mining (ICDM)*, pages 21–30, 2016.

[Bogomolny, 2017] Alexander Bogomolny. Cut The Knot: Perimeters of Convex Polygons, One within the Other (http://www.cut-the-knot.org/m/Geometry/PerimetersOfTwoConvexPolygons.shtml), 2017.

[Bosc *et al.*, 2016] Guillaume Bosc, Chedy Raïssi, Jean-François Boulicaut, and Mehdi Kaytoue. Any-time diverse subgroup discovery with monte carlo tree search. *CoRR*, abs/1609.08827, 2016.

[Browne *et al.*, 2012] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[Devillers, 1999] Olivier Devillers. On Deletion in Delaunay Triangulations. In Victor Milenkovic, editor, *Proceedings of the Fifteenth Annual Symposium on Computational Geometry, Miami Beach, Florida, USA, June 13-16, 1999*, pages 181–188. ACM, 1999.

[Duivesteijn *et al.*, 2016] Wouter Duivesteijn, Ad J. Feelders, and Arno Knobbe. Exceptional Model Mining. *Data Mining and Knowledge Discovery*, 30(1):47–98, 2016.

[Falher *et al.*, 2015] Géraud Le Falher, Aristides Gionis, and Michael Mathioudakis. Where Is the Soho of Rome? Measures and Algorithms for Finding Similar Neighborhoods in Cities. In *ICWSM 2015*, pages 228–237, 2015.

[Ganter and Kuznetsov, 2001] Bernhard Ganter and Sergei O. Kuznetsov. Pattern Structures and Their Projections. In *International Conference on Conceptual Structures (ICCS)*, volume 2120 of *Lecture Notes in Computer Science*, pages 129–142. Springer, 2001.

[Ganter and Wille, 1999] B. Ganter and R. Wille. *Formal Concept Analysis*. Springer, 1999.

[Giacometti *et al.*, 2013] Arnaud Giacometti, Dominique Haoyuan Li, Patrick Marcel, and Arnaud Soulet. 20 years of pattern mining: a bibliometric survey. *SIGKDD Explorations*, 15(1):41–50, 2013.

[Grosskreutz and Rüping, 2009] Henrik Grosskreutz and Stefan Rüping. On subgroup discovery in numerical domains. *Data Min. Knowl. Discov.*, 19(2):210–226, 2009.

[Kaytoue *et al.*, 2011] Mehdi Kaytoue, Sergei O. Kuznetsov, and Amedeo Napoli. Revisiting Numerical Pattern Mining with Formal Concept Analysis. In *IJCAI*, pages 1342–1347, 2011.

[Kaytoue *et al.*, 2017] Mehdi Kaytoue, Marc Plantevit, Albrecht Zimmermann, Anes Bendimerad, and Céline Robardet. Exceptional contextual subgraph mining. *Machine Learning*, pages 1–41, 2017.

[Kuznetsov, 1993] Sergei O. Kuznetsov. A Fast Algorithm for Computing All Intersections of Objects in a Finite Semi-lattice. *Nauchno-Tekhnicheskaya Informatsiya*, ser. 2(1):17–20, 1993.

[Kuznetsov, 2005] Sergei O. Kuznetsov. Galois Connections in Data Analysis: Contributions from the Soviet Era and Modern Russian Research. In *Formal Concept Analysis, Foundations and Applications*, volume 3626 of *Lecture Notes in Computer Science*, pages 196–225. Springer, 2005.

[Overmars and van Leeuwen, 1981] Mark H. Overmars and Jan van Leeuwen. Maintenance of Configurations in the Plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981.

[The CGAL Project, 2016] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.9 edition, 2016.

[Zalik, 2005] Borut Zalik. An efficient sweep-line Delaunay triangulation algorithm. *Computer-Aided Design*, 37(10):1027–1038, 2005.

[Zimmermann and Nijssen, 2014] Albrecht Zimmermann and Siegfried Nijssen. Supervised Pattern Mining and Applications to Classification. In *Frequent Pattern Mining*, pages 425–442. Springer, 2014.