Mining Convex Polygon Patterns with Formal Concept Analysis

Aimene Belfodil[a], **Sergei O. Kuznetsov**[b], **Céline Robardet**[a], **Mehdi Kaytoue**[a]
[a] Univ Lyon, INSA Lyon, CNRS, LIRIS, F-69621, LYON, France
[b] National Research University Higher School of Economics, Moscow, Russia
firsname.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 for capturing high density areas in any pattern mining task. Our contribution is three-fold: (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, that 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 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 of a given type 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 [Ganter and Wille, 1999], going beyond the formalization of hyper-rectangles given by [Kuznetsov, 2005] in the early 90's 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 polygons 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 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. 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 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, 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 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 noted W_{m_i}. $m_i(g) = w$ means that w is the value of attribute m_i for object $g \in G$. Figure 1(right) plots 5 objects with 2 attributes on the Euclidean space.

Interval pattern. A numerical pattern is a box (hyper-rectangle) 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 $d = \langle [a_i, b_i]\rangle_{1\leq i\leq |M|}$ 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.

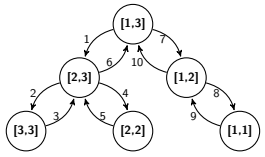
Interval pattern search space. The search space of interval patterns is the finite set D of all interval vectors $\langle [a_i, b_i]\rangle_{1\leq i\leq |M|}$ with $a_i, b_i \in W_{m_i}$. The size of the search space is given by: $|D| = \prod_{i\in[\![1,|M|]\!]}(|W_{m_i}| \times (|W_{m_i}|+1)/2)$.

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 maximal patterns 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 $d \in (D, \sqcap)$

$$d^{\square} = \{g \in G | d \sqsubseteq \delta(g)\} \quad A^{\square} = \bigsqcap_{g\in A} \delta(g)$$

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,



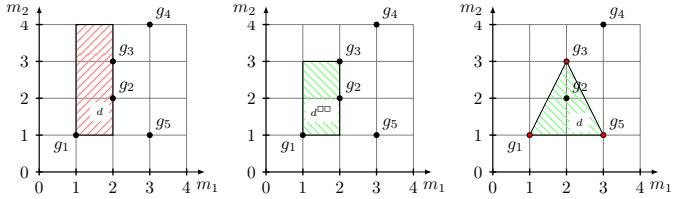


Figure 1: Non-closed (left) and closed (middle) interval pattern; convex polygon pattern (right)

which is the minimal bounding box of all objects in G ($G^{\square}$). Then, as shown on the figure aside, 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 at the attribute m_i, only attributes $m_j >= m_i$ can be refined in further steps from the generated pattern.

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.

Spatial attribute and convex polygon pattern. A spatial attribute m takes values in $\mathbb{R}^2$. A convex polygon pattern is a *compact convex polygon* represented by a sequence of h points in *counterclockwise order (ccw)* $d = [p_i]_{1\leq i\leq h}$ where $p_i \in \mathbb{R}^2$. For all subsets of points $e \subseteq d$ of size 3, the points of e are *not collinear*. An object g is in the image of a convex polygon pattern d when g is in the closed area formed by the polygon d. The support of d, denoted by $sup(d)$, is the set of object $g \in G$ in the image of d. Figure 1 (right) shows the convex polygon pattern $d = \langle (1, 1), (3, 1), (2, 3)\rangle$ whose support is $\{g_1, g_2, g_3, g_5\}$.

Convex polygon patterns search space. D is given by the set of all possible *convex polygon* that can be constructed using every subset of objects $A \subseteq G$.

Before presenting the *pattern structure* for *convex polygon patterns*, we define the notion of *convex hull*. Given a finite subset $E \subseteq \mathbb{R}^2$, the *convex hull* of E denoted by $ch(E)$ is the smallest convex set that contains E which is the closed area of a *convex polygon*. Given a convex

polygon P, we denote by $\overline{P}$ the *extreme points* of P. $\overline{P}$ is the minimal set in $\mathbb{R}^2$ where $ch(\overline{P}) = P$. Note that, when $P = ch(E)$, $\overline{ch(E)}$ is a subset of E. In Figure 1 (right), the hatched triangle represents the convex hull of $\{g_1, g_2, g_3, g_5\}$ whose extreme points are $\{g_1, g_3, g_5\}$.

Extreme points form what we call a $\mathcal{V}$-representation (vertex representation). Another representation of convex hull (called $\mathcal{H}$-representation) consists of intersection of h closed half-planes where h is the number of extreme points of the convex hull. Formally: $ch(E) = \{x \in \mathbb{R}^2 \mid Ax \leq b\}$ with $b \in \mathbb{R}^h$ and $A = (a_{i,j})_{1 \leq i \leq h, 1 \leq j \leq 2} \in \mathbb{M}_{h,2}(\mathbb{R})$. ch is a monotone, extensive and idempotent application from $\mathbb{R}^2$ into $\mathbb{R}^2$: i.e., a closure operator.

Convex polygon pattern structure. We define a *convex polygon pattern structure* as follows: $(G, (D, \sqcap), \delta)$ where G is the set of objects described by one spatial attribute m, $(D, \sqcap)$ the semi-lattice of object descriptions (convex polygons) where $D = \{\overline{ch(A)} | A \subseteq G\}$ (note that $\overline{ch(A)}$ here produces the sequence of extreme points in *counterclockwise order*). $\delta : G \to D$ is a mapping that associates to each object $g \in G$, $\delta(g) = \langle m(g) \rangle$ (a degenerate polygon with a singleton point). The *infimum* $\sqcap$ is defined as follows. Let $c = \langle p_i \rangle_{i \in [\![1,h]\!]}$ and $d = \langle q_i \rangle_{i \in [\![1,k]\!]}$ 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 $d \in (D, \sqcap)$: $d^{\square} = \{g \in G \mid g \subseteq ch(d)\}$ and $A^{\square} = \overline{ch(A)}$. A pair (A, d) such that $A^{\square} = d$ and $d^{\square} = A$ is a concept. Both A and d are closed under $(.)^{\square\square}$. A contains all points, while d only extreme points.

Mining convex polygons with constraints. There are 2^n convex polygons in the worst case (n points taken from 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 ($|d| \leq \delta_{compl}$), minimal support ($|A| \geq \delta_{supp}$), minimal/maximal perimeter ($perim(d) \geq$ or $\leq \delta_{perim}$), and minimal/maximal area ($area(d) \geq$ or $\leq \delta_{area}$). 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

We propose three algorithms for mining convex polygon patterns. The first one enumerates object subsets in a bottom-up way with closures. The next one considers a top-down enumeration and does not compute 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 *CloseByOne* 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 the empty set. A total order on G is provided, e.g. $g_1 < g_2 < .....$ 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 will serve as a baseline.

4.2 Updating a Delaunay triangulation

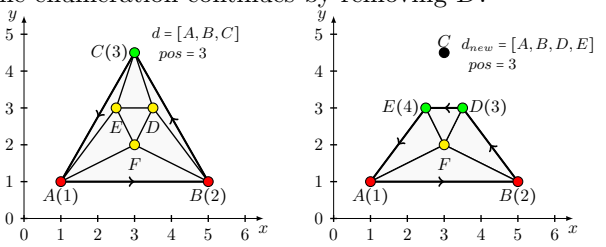
The enumeration starts from the most general pattern $G^{\square}$. Minimal changes are applied to obtain the next pattern to ensure the correctness and completeness of the enumeration: all convex polygons and only them are visited. A minimal change to a polygon d consists in removing an extreme point. There are $|d|$ of such minimal changes for a polygon d for obtaining next smaller polygons $e = d \backslash \{d[i]\}$ s.t. $d \sqsubset e$, $i = 1, ..., |d|$. 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 actually 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.

Algorithm DelaunayEnum**.** 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 $\mathcal{O}(|P| \cdot log(|P|))$. 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 compute $DT(d \setminus \{d[i]\})$ from $DT(d)$. During this computing, one can easily update 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 enumerates *Delaunay triangulations* dt and maintains at each step the sequence of extreme points d. Algorithm 1 shows the pseudo-code. It starts from pattern $G^{\square} = \overline{ch(G)}$ (line 3). 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 10), the new description d_{new} (sequence of extreme points) (line 11) and the new support c_{new} by rremoving objects with value p from c (line 12). For non-redundancy, removal of extreme points before the last removed extreme point p are not allowed (argument *pos* in Algorithm 1) 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 (*counterclockwise*). 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 1 DelaunayEnum

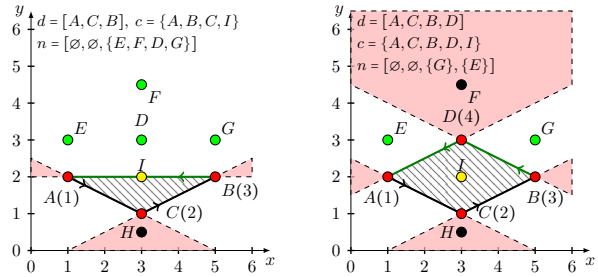
```
1: procedure DelaunayEnum(G)
2:     dt ← Delaunay(G)
3:     d ← dt̄                                   ▷ dt̄ = G□ = ch(G)
4:     traverse(G,d,dt,1)
5: end procedure
6: procedure traverse(c,d,dt,pos)
7:     Print(c,d)                               ▷ visit pattern concept (c, d)
8:     for i ∈ pos, ..., |d| do
9:         p ← d[i]
10:        dt_new ← Delaunay_remove(dt,p)
11:        d_new ← dt_new‾                      ▷ where d_new[1] = d[1] if i > 1
12:        c_new ← c\⟨p⟩□                       ▷ [p]□ = {g ∈ G | m(g) = p}
13:        traverse(c_new,d_new, dt_new,i)
14:    end for
15: end procedure
```

4.3 Enumerating simpler shapes first

We consider the pattern search space D differently. Until now, elements of $(D, \sqsubseteq)$ were ordered w.r.t. polygon inclusion. Now we consider $(D, \bar{\sqsubseteq})$, where polygons are ordered w.r.t. extreme points inclusion: $c \,\bar{\sqsubseteq}\, d \Leftrightarrow c \supseteq d$. Intuitively, the i^{th} level of $(D, \bar{\sqsubseteq})$ (level 0 being $\varnothing$), considers all polygons with i extreme points ($i = 3$ are the triangles, $i = 4$ the quadrilaterals, ...). Enumerating simpler shapes first is interesting as they are easier to interpret and less stick to the data points.

In order to enumerate all the polygons w.r.t the new partial order $\bar{\sqsubseteq}$ in *bottom-up* fashion, 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 pattern for sure (no need to compute the convex hull). The intuition is given inductively.

- *Case 0.* Consider the points $(G, \leq)$ as an arbitrary total order where, without loss of generality and for the sake of simplicity, elements of G are pairwise distinct. Let the empty pattern $d = [\,]$ (0 extreme point), it is sure that every $A \in G$ forms a pattern (point).
- *Case 1.* Consider now the pattern $d = [A]$ (1 extreme point), every object B where $B > A$ will for sure produce a pattern $[A, B]$ (a segment).
- *Case 2.* Let the pattern $d = [A, B]$ (2 extreme points), the only objects $C \in G$ that are candidates are: *(i)* $C > B$ and *(2)* A, B and C are not collinear. We distinguish between two sets of candidate points: those that are in the *open lower half-plane of the oriented segment* AB *denoted by* $n(AB) = AB^-$ (forming a *ccw-triangle* ACB), and those that are in the *open upper half-plane of the oriented segment* AB *denoted by* $n(BA) = BA^- = AB^+$ (forming *ccw-triangle* ABC). Note that, points which are on the segment AB denoted by AB^0, will belong to the support of the pattern AB. In the figure below (left), $n(AB) = AB^- = \{C, H\}, n(BA) = AB^+ = \{D, E, F, G\}$ and $AB^0 = \{A, B, I\}$



- *Case k.* Consider now the general case where the current number of extreme points is k with $k \geq 2$. Let $d = [A_i]_{1 \leq i \leq k}$ the current description which support is c. In the general case, the candidate points set of the face A_jA_{j+1} is $n(A_jA_{j+1}) = A_jA_{j+1}^- \cap A_{j-1}A_j^+ \cap A_{j+1}A_{j+2}^+$ (note that here, $j + 1 = 1$ if $j = k$ and $j - 1 = k$ if $j = 1$). Consider $N \in n(A_iA_{i+1})$ a candidate point for the face A_iA_{i+1} (with $N > A_j \;\forall j \in \{1, ..., k\}$) , the new pattern after insertion of N is $d_{new} = \{A_1, ..., A_i, N, A_{i+1}, ..., A_k\}$ which support is $c_{new} = c \cup A_iN^0 \cup NA_{i+1}^0 \cup I$ *s.t.* $I = A_iA_{i+1}^- \cap A_iN^+ \cap NA_{i+1}^+$ is the set of objects contained in the open area enclosed by the triangle A_iNA_{i+1}. Clearly, after the insertion of N, one need to update the candidate points of some faces of the new description d_{new}. It is evident that only faces related to A_i and A_{i+1} will be updated ($A_{i-1}A_i$, A_iN, NA_{i+1} and $A_{i+1}A_{i+2}$). They will be updates as below:
  - $n(A_iN) = A_{i-1}A_i^+ \cap A_iN^- \cap NA_{i+1}^+$

- $n(NA_{i+1}) = A_{i+1}A^+_{i+2} \cap NA^-_{i+1} \cap A_iN^+$
- $n(A_{i-1}A_i) = n(A_{i-1}A_i) \cap A_iN^+$
- $n(A_{i+1}A_{i+2}) = n(A_{i+1}A_{i+2}) \cap NA^+_{i+1}$

It is easy to verify that $n(A_iN) = n(A_iA_{i+1}) \cap A_iN^- \cap NA^+_{i+1}$ and $n(NA_{i+1}) = n(A_iA_{i+1}) \cap NA^-_{i+1} \cap A_iN^+$. Note that for the particular case ($k = 2$), we have $A_{i-1}A_i$ is the same as $A_{i+1}A_{i+2}$, thus $n(A_{i-1}A_i) = n(A_{i-1}A_i) \cap A_iN^+ \cap NA^+_{i+1}$. We can see the candidate points as a vector which length is $|d|$ and where $n[i]$ denote candidate points of the face A_iA_{i+1}. In the figure above (right), we have $d = [A, C, B]$, $c = \{A, B, C, I\}$ and $n = [\varnothing, \varnothing, \{E, F, \underline{D}, G\}]$. When we add $\underline{D}$ between B and A we obtain the new description $d_{new} = [A, C, B, \underline{D}]$, the new support will become $c_{new} = \{A, B, C, \underline{D}, I\}$ and $n_{new} = [\varnothing, \varnothing, \{G\}, \{E\}]$. Note that F is no longer a candidate point for the new description.

Algorithm ExtremePointsEnum**.** The algorithm follows almost the same enumeration principle than ExtCbO. The difference lies in maintaining the list of object candidates that can be added to a pattern d to generate $e \sqsubset d$. The other difference is that the two first levels are enumerated in BFS manner. This allows to build the segment index S that stores for each distinct object pair A_iA_j, the support $A_iA^0_j$ (objects in segment A_iA_j) and its candidates $A_iA^-_j$ and $A_iA^+_j$. 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 ($k \geq 2$). Algorithm 2 gives the pseudo-code of ExtremePointsEnum.

Algorithm 2 ExtremePointsEnum

```
1: Let (G, ≤) be the 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(G)
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
```

5 Experiments

We report an experimental study of the different algorithms, carried out on 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`.

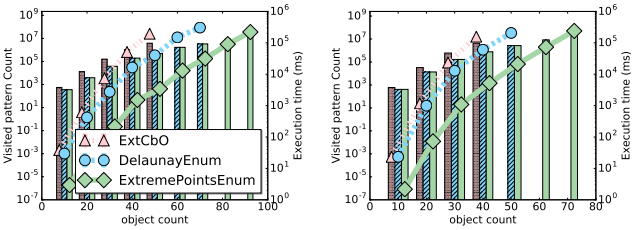


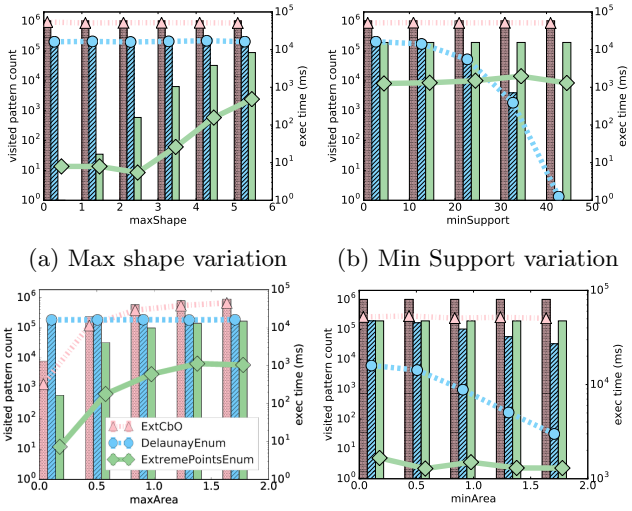
Figure 4: Polygon pattern enumeration comparison

Mining polygon patterns. We consider the task of mining polygon patterns and compare our three algorithms without any constraint: ExtCbO, DelaunayEnum and ExtremePointsEnum. Figure 4 plots for each one their run times and the number of pattern candidates they needed to generate. Datasets consist in n objects drawnfrom 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's 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 to 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-know property in pattern mining). DelaunayEnum enumerates convex polygons in a top-down fashion w.r.t to 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 5 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 are removed). As such, depending on the constraints the user is interested in, one algorithm may be preferred to another.

Intervals vs. polygons. Our main motivation for introducing convex polygon patterns is to discover shapes



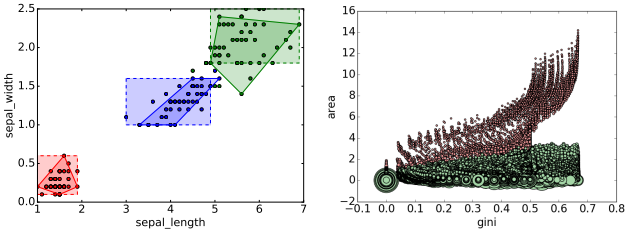
(a) Max shape variation (b) Min Support variation

(c) Max Area variation (d) Min Area variation

Figure 5: Run time and generated pattern count for our three algorithms when introducing constraints

with high density and area, and possibly with a high class homogeneity (e.g., low Gini). Figure 6 (left) considers the IRIS dataset (Sepal length vs. Sepal Width). It presents the three most frequent polygons that have a 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 6 (right) 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.



(a) Top-3 patterns (b) Gini, area and density

Figure 6: Comparing interval and polygon patterns

Sampling polygon patterns with MCTS. Our algorithms are clearly not scalable as the number of polygons is too important. 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 chose 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 eventually 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.

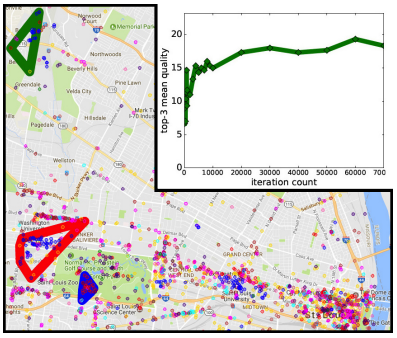


Figure 7: Top three homogeneous polygons on Foursquare places from Saint-Louis (Missouri, USA)

[Falher *et al.*, 2015] propose several spatial datasets (cities) containing *Foursquare* users check-in places of different kinds ("College & university", "Art & Entertainement", ...). We randomly chose the city of Saint-Louis (Missouri, USA) containing $3,464$ points. Running our algorithms on this dataset is impossible. The MCTS sampling however quickly returns patterns of high quality: Figure 7 displays average the 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 7 presents the 3 best patterns which are homogeneous zones with a large support. Designing a quality measure Q can be achieved in many ways (e.g., involving density).

6 Conclusion

In pattern mining, hyper-rectangles cannot always properly capture interesting areas. We formally defined convex polygons 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.

Acknowledgment. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

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] A. A. Bendimerad, M. Plantevit, and C. 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] G. Bosc, J.-F. Boulicaut, C. Raïssy, and M. Kaytoue. Anytime Discovery of a Diverse Set of Patterns with Monte Carlo Tree Search. *ArXiv e-prints*, September 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. *Automatic Documentation and Mathematical Linguistics*, 27(5):400–412, 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.

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

[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.