

On Making Skyline Queries Resistant to Outliers

Hélène Jaudoin, Pierre Nerzic, Olivier Pivert and Daniel Rocacher

Abstract This paper deals with the issue of retrieving the most preferred objects (in the sense of Skyline queries, i.e., of Pareto ordering) from a collection involving outliers. Indeed, many real-world datasets, for instance from ad sales websites, contain odd data and it is important to limit the impact of such odd data (outliers) on the result of skyline queries, and prevent them from hiding more interesting points. The approach we propose relies on the notion of fuzzy typicality and makes it possible to compute a graded skyline where each answer is associated with both a degree of membership to the skyline and a typicality degree. A GPU-based parallel implementation of the algorithm is described and experimental results are presented, which show the scalability of the approach.

1 Introduction

In this paper, a qualitative view of preference queries is considered, namely the *Skyline* approach introduced in Börzsönyi et al. (2001). Given a set of points in a space, a skyline query retrieves those points that are not dominated by any other in the sense of Pareto order. When the number of dimensions on which preferences are expressed gets high, many tuples may become incomparable. Several approaches have been proposed to define an order for two incomparable tuples, based on the number of other tuples that each of the two tuples dominates (notion of k -representative dominance proposed in Lin et al. (2007)), on a preference order over the attributes (see for instance the notions of k -dominance and k -frequency introduced in Chan et al. (2006a, b)),

H. Jaudoin (✉) · P. Nerzic · O. Pivert · D. Rocacher
Irisa, 6 rue Kerampont, Lannion, France
e-mail: hjaudoin@irisa.fr

P. Nerzic
e-mail: nerzic@univ-rennes1.fr

O. Pivert
e-mail: pivert@irisa.fr

D. Rocacher
e-mail: rocacher@irisa.fr

or on a notion of representativity ((Tao et al. 2009) redefines the approach proposed by Lin et al. (2007) and proposes to return only the most representative points of the skyline, i.e., a point among those present in each cluster of the skyline points). Other approaches fuzzify the concept of skyline in different ways, see e.g. Goncalves and Tineo (2007) and Hadjali et al. (2011). See also Rojas et al. (2014) where the authors define soft skylines by relaxing the dominance relation. Here, we are concerned with a different problem, namely that of the possible presence of *exceptional* points, also known as outliers, in the dataset over which the skyline is computed. Such exceptions may correspond to noise or to the presence of *nontypical* points in the collection considered. The impact of such points on the skyline may obviously be important if they dominate some other, more representative ones.

At least two strategies can be considered to handle outliers. The former consists in removing anomalies by adopting cleaning procedures. However, the task of automatically distinguishing between odd points and simply exceptional points is not always easy. Another solution is to define an approach that is *tolerant to outliers*, that highlights representative points of the database and that points out the possible outliers. Literature about outlier detection is very abundant as shown by recent surveys (Hodge and Austin 2004; Niu et al. 2011; Zimek et al. 2012; Zhang 2013) and recent publications in the data mining area (Gupta et al. 2013; Gabel et al. 2013; Zimek et al. 2013; Ji et al. 2013). The approaches may be categorized into three classes: (i) those that aim to isolate data distant from another *normal* data, (ii) those that aim to detect if a new observation is normal or abnormal, (iii) those that exclusively focus on modelling normality and assess if a new observation is likely to be normal.

The estimation of the best approach for detecting outliers is out of the scope of this paper. In the tolerant skyline approach we propose, we use a simple detection technique based on the notions of frequency and distance. More precisely we adopt the fuzzy notion of typicality (Zadeh 1984) in order to identify non-typical, thus exceptional points. We revisit the definition of a skyline and show that it (i) makes it possible to retrieve the dominant points without discarding other potentially interesting ones, and (ii) constitutes a flexible tool for distinguishing between the answers.

The remainder of the paper is structured as follows. Section 2 provides a refresher about skyline queries and motivates the approach. Section 3 presents the principle of exception-tolerant skyline queries, based on the fuzzy concept of typicality. Section 4 deals with implementation aspects whereas Sect. 5 presents experimental results obtained on a real-world dataset. Finally, Sect. 6 recalls the main contributions and outlines perspectives for future work.

2 Refresher About Skyline Queries and Motivations

2.1 Skyline Queries

Let $\mathcal{D} = \{D_1, \dots, D_d\}$ be a set of d dimensions. Let us denote by $\text{dom}(D_i)$ the domain associated with dimension D_i . Let $\mathcal{S}$ be a subset of $\text{dom}(D_1) \times \dots \times \text{dom}(D_d)$, p and q two points of $\mathcal{S}$, and $\succ_i$ a preference relation on D_i . One says that p *dominates* q on $\mathcal{D}$ (p is better than q according to Pareto order), denoted by $p \succ_{\mathcal{D}} q$, iff

$$\forall i \in [1, d] : p_i \succeq_i q_i \text{ and } \exists j \in [1, d] : p_j \succ_j q_j$$

A skyline query on $\mathcal{D}$ applied to a set of points $\mathcal{S}$, whose result is denoted by $\text{SKY}_{\mathcal{D}}(\mathcal{S})$, according to preference relations $\succ_i$, produces the set of points that are not dominated by any other point of $\mathcal{S}$:

$$\text{SKY}_{\mathcal{D}}(\mathcal{S}) = \{p \in \mathcal{S} \mid \nexists q \in \mathcal{S} : q \succ_{\mathcal{D}} p\}$$

Depending on the context, one may try, for instance, to maximize or minimize the values of $\text{dom}(D_i)$, assuming that $\text{dom}(D_i)$ is a numerical domain.

In order to illustrate the principle of the approach we propose, let us consider the dataset *Iris* (Fisher 1936), graphically represented in Fig. 1.

The vertical axis corresponds to the attribute *sepal width* whereas the horizontal axis is associated with *sepal length*. The skyline query:

```
select * from iris
skyline of sepalength max, sepalwidth max
```

looks for those points that maximize the dimensions *length* and *width* of the sepals (the circled points in Fig. 1).

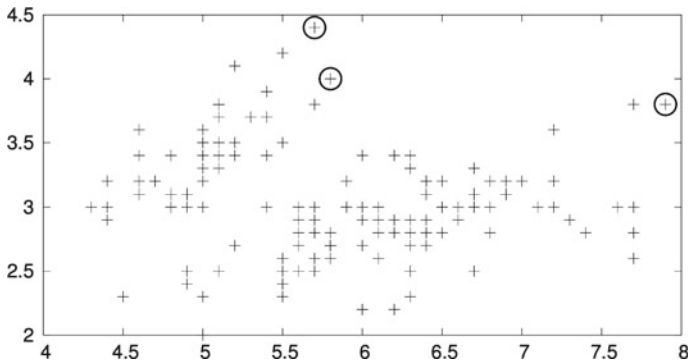


Fig. 1 The dataset *Iris*

In this dataset, the points form two groups that respectively correspond to the intervals $[4, 5.5]$ and $[5.5, 7]$ on attribute *length*. By definition, the skyline points are on the border of the region that includes the points of the dataset. However, these points are very distant from the areas corresponding to the two groups and are thus not very representative of the dataset. It could then be interesting for a user to be able to visualize the points that are “almost dominant”, closer to the clusters, then more representative of the dataset. A way to make such points visible without discarding extrema, while allowing to discriminate them, is to use a gradual view of representativity. The notion of typicality discussed in the next section makes it possible to reach that goal.

2.2 Computing a Fuzzy Set of Typical Values

The concept of typicality has been studied by numerous authors, both in cognitive psychology (Rosch and Mervis 1975; Hampton 1988; Osherson and Smith 1997) and in fuzzy logic, see e.g. Zadeh (1982), Friedman et al. (1995) and Yager (1997). In the following, we consider Zadeh’s interpretation (Zadeh 1987), but any other interpretation of typicality could be used without drastically altering the general principle of the approach. In Zadeh (1987), x is defined as a typical element of a fuzzy set A iff (i) x has a high degree of membership to A and (ii) most of the elements of A are similar to x . In the case where A is a classical set, this definition becomes: x belongs to A and most of the elements of A are similar to x . In the same spirit, in Dubois and Prade (1984), the authors define a typicality indice based on similarity and frequency. We adapt their definition as follows. Let us consider a set $\mathcal{E}$ of points. We say that a point is all the more typical as it is close to many other points. The proximity relation considered is based on Euclidean distance. We consider that two points p_1 and p_2 are close to each other if $d(p_1, p_2) \leq \tau$ where τ is a predefined threshold. In the experiment performed on the dataset *Iris*, we used $\tau = 0.5$. The frequency of a point is defined as:

$$F(p) = \frac{|\{q \in \mathcal{E}, d(p, q) \leq \tau\}|}{|\mathcal{E}|} \quad (1)$$

This degree is then normalized into a typicality degree in $[0, 1]$:

$$typ(p) = \frac{F(p)}{\max_{q \in \mathcal{E}} \{F(q)\}} \quad (2)$$

We will also use the following notations:

$$\text{TYP}(\mathcal{E}) = \{typ(p)/p \mid p \in \mathcal{E}\}$$

$$\text{TYP}_\gamma(\mathcal{E}) = \{p \mid p \in \mathcal{E} \text{ and } typ(p) \geq \gamma\}.$$

$\text{TYP}(\mathcal{E})$ represents the fuzzy set of points that are somewhat typical of the set $\mathcal{E}$ while $\text{TYP}_\gamma(\mathcal{E})$ gathers the points of $\mathcal{E}$ whose typicality is over the threshold γ .

Example 1 Let us consider the following multiset (of cardinality $n = 30$):

$$E = \langle 1/0, 1/3, 1/4, 4/5, 7/6, 5/7, 3/8, 5/9, 2/12, 1/23 \rangle$$

where k/e means that element e has k copies in E . Using Formulas (1) and (2) with $\tau = 2$, we get:

$$\text{TYP}(\mathcal{E}) = \{0.04/0, 0.25/3, 0.54/4, 0.75/5, 0.83/6, \\ 1/7, 0.83/8, 0.54/9, 0.08/12, 0.04/23\}$$

and

$$\text{TYP}_{0.5}(\mathcal{E}) = \{4, 5, 6, 7, 8, 9\}.\diamond$$

An excerpt of the typicality degrees computed over the *Iris* dataset is presented in Table 1.

Remark 1 This typicality-based interpretation of outliers is close to the approach used in DBSCAN (Ester et al. 1996). Typical points (relative to thresholds τ and γ) correspond to *core* points in DBSCAN. However, as we will see, the fuzzy skyline definition introduced in Sect. 3.2 relies on a *gradual* view of outliers where no threshold γ is applied.

Table 1 Excerpt of the *Iris* dataset with associated typicality degrees

Length	Width	Frequency	Typicality
7.4	2.8	0.0600	0.187
7.9	3.8	0.0133	0.0417
6.4	2.8	0.253	0.792
6.3	2.8	0.287	0.896
6.1	2.6	0.253	0.792
7.7	3.0	0.0467	0.146
6.3	3.4	0.153	0.479
6.4	3.1	0.293	0.917
6.0	3.0	0.320	1.000

3 Principle of the Exception-Tolerant Skyline

As explained in the introduction, our goal is to revisit the definition of the skyline so as to take into account the typicality of the points in the database, in order to control the impact of exceptions or anomalies. Thus, three variants of the classical skyline are defined hereafter:

- $\text{SKY}_{\mathcal{D}}^1$ that returns all sufficiently typical points of $\mathcal{S}$ that are not dominated by sufficiently typical points,
- $\text{SKY}_{\mathcal{D}}^2$ that returns all points of $\mathcal{S}$ that are not dominated by sufficiently typical points, and
- $\text{SKY}_{\mathcal{D}}^3$ that returns a fuzzy set of $\mathcal{S}$, where each point is associated with a membership degree which is a function of the typicality of the points that dominate it.

3.1 Boolean View

The first idea is to restrict the computation of the skyline to a subset of $\mathcal{E}$ that corresponds to sufficiently typical points. The corresponding definition is:

$$\text{SKY}_{\mathcal{D}}^1(\mathcal{S}) = \{p \in \text{TYP}_{\gamma}(\mathcal{S}) \mid \nexists q \in \text{TYP}_{\gamma}(\mathcal{S}) \text{ such that } q \succ_{\mathcal{D}} p\} \quad (3)$$

Such an approach obviously reduces the cost of the processing since only the points that are typical at least to the degree γ are considered in the calculus. However, this definition does not make it possible to discriminate the points of the result since the skyline obtained is a crisp set. Figure 2 illustrates this behavior and shows the maxima (circled points) obtained when considering the points that are typical to a degree ≥ 0.7 (represented by crosses).

Another drawback of this definition is to exclude the nontypical points altogether, even though some of them could be interesting answers. A more cautious definition consists in keeping the nontypical points while computing the skyline and transform Eq. (3) into:

$$\text{SKY}_{\mathcal{D}}^2(\mathcal{S}) = \{p \in \mathcal{S} \mid \nexists q \in \text{TYP}_{\gamma}(\mathcal{S}) \text{ such that } q \succ_{\mathcal{D}} p\} \quad (4)$$

Figure 3 illustrates this alternative solution. It represents (circled points) the objects from the *Iris* dataset that are not dominated by any item typical to the degree $\gamma = 0.7$ at least (represented by crosses).

With Eq. (3), the nontypical points are discarded, whereas with Eq. (4), the skyline is larger and includes nontypical extrema. This approach relaxes skyline queries in such a way that the result obtained is not a polyline anymore but a stripe composed of the regular skyline elements completed with possible “substitutes”. However, the main drawbacks of this definition are: (i) the potentially large number of points returned, (ii) the impossibility to distinguish, among the skyline points, those that are not at all dominated from those that *are* dominated (by more or less typical points).

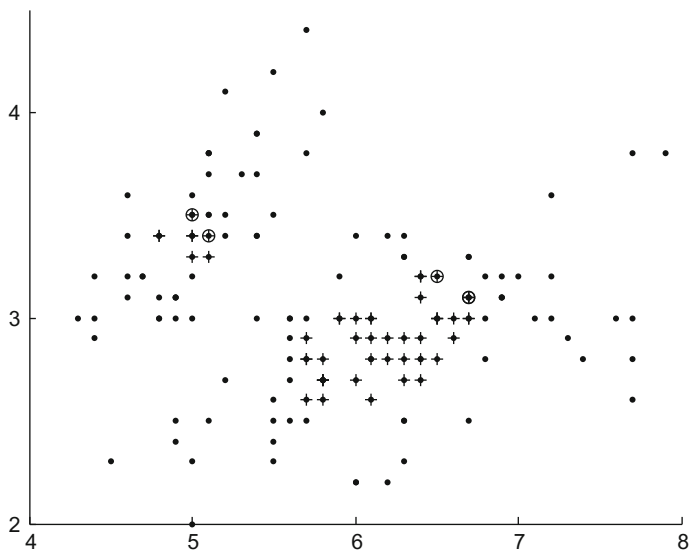


Fig. 2 Skyline of the Iris points whose typicality degree is ≥ 0.7

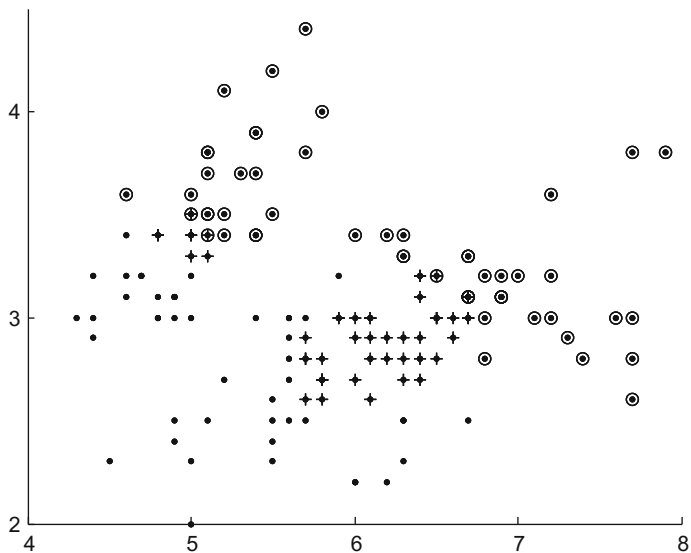


Fig. 3 Points that are not dominated by any other whose typicality degree is ≥ 0.7

3.2 Gradual View

A third version makes it possible to compute a *graded* skyline, seen as a fuzzy set, that preserves the gradual nature of the concept of typicality. By doing so, no threshold γ is applied to typicality degrees anymore. This variant considers that a point totally belongs to the skyline (membership degree equal to 1) if it is dominated by no other point. A point does not belong at all to the skyline (membership degree equal to 0) if it is dominated by at least one totally typical point. In the case where p is dominated by somewhat (but not totally) typical points, its degree of membership to the skyline is strictly positive and depends on the typicality of these points. The corresponding formula is:

$$\text{SKY}_{\mathcal{D}}^3(\mathcal{S}) = \{\mu/p \mid p \in \mathcal{S}\} \quad (5)$$

with μ defined as:

$$\mu = \begin{cases} 1 & \text{if } p \in \text{SKY}_{\mathcal{D}}(\mathcal{S}) \\ 1 - t & \text{otherwise, where } t = \max_{q \in \mathcal{S} \mid q \succ_{\mathcal{D}} p} \text{typ}(q) \end{cases} \quad (6)$$

This latter definition is a fuzzy interpretation of the statement

$$p \text{ is in the skyline} \Leftrightarrow \forall q, \text{ if } q \text{ is typical, then } q \text{ does not dominate } p.$$

Indeed, using Kleene-Dienes implication ($x \rightarrow_{KD} y = \max(1 - x, y)$) and translating $\forall$ by the triangular norm minimum (which is the usual interpretation of the conjunction in fuzzy logic), we obtain the formula above.

With the *Iris* dataset, one gets the result presented in Fig. 4, where the degree of membership to the skyline corresponds to the z axis.

This approach appears interesting in terms of visualization. Indeed, the score associated with each point of the result makes it possible to focus on different α -cuts of the skyline. In Fig. 4, one may notice a slope from the optimal points towards the

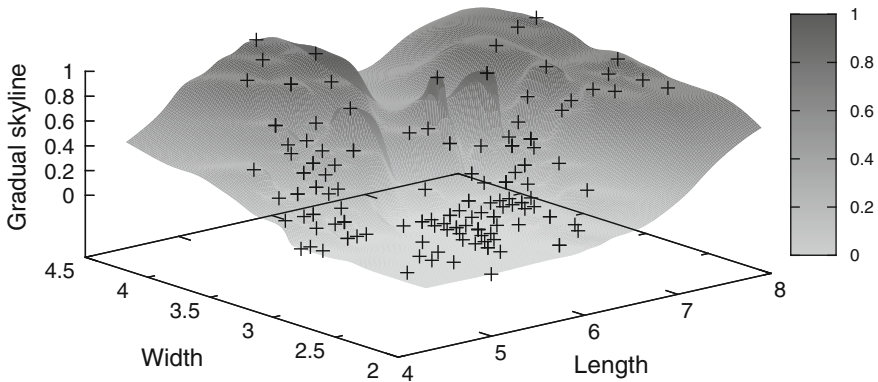


Fig. 4 Graded skyline obtained with the *Iris* dataset

less typical or completely dominated ones. The user may select points that are not necessarily optimal but that represent good alternatives to the regular skyline answers (in the case, for instance, where the latter look “too good to be true”). Finally, an element of the graded skyline is associated with two scores: a degree of membership to the skyline, and a typicality degree (that expresses the extent to which it is *not* exceptional). One may imagine different ways of navigating inside areas in order to explore the set of answers: a simple scan for displaying the characteristics of the points, the use of different filters aimed, for instance, at optimizing diversity on certain attributes, etc.

Remark 2 In Eq. (6), t denotes the typicality of the most typical point that dominates p . A possible refinement would be to take into account, not only the typicality of the most typical point that dominates p , but also the *number* of such points. Indeed, let p and p' be two points which have the same typicality r . Assume that p is dominated by a large number N of points, all of which have typicality r' . Assume further that p' is dominated by only one point, whose typicality is also r' . The approach described above cannot discriminate p and p' . A way to overcome this limitation would be to associate two values with each point p of the fuzzy skyline: a membership degree $\mu(p)$ as defined in Eqs. (5) and (6), and a number $n(p)$ defined as

$$n(p) = |\{q \in \mathcal{S} \mid q \succ_{\mathcal{D}} p \text{ and } typ(q) = \max_{u \in \mathcal{S} \mid u \succ_{\mathcal{D}} p} typ(u)\}| \quad (7)$$

Finally, each element p of the result would be associated with three degrees: $typ(p)$, $\mu(p)$ and $n(p)$ and the selection of the most interesting points of this fuzzy skyline could be performed by means of a second (classical) skyline query aimed at maximizing both $typ(p)$ and $\mu(p)$, and involving a nested condition (*cascade clause*) minimizing $n(p)$ in order to break ties if any. The detailed study and implementation of such an extension is left for future work.

4 Implementation Aspects

Two steps are necessary for obtaining the graded result: (i) computation of the typicality degrees, and (ii) computation of the graded skyline. The typicality degrees are used to compute the degree of membership of all tuples to the skyline.

Let us first investigate the methods to process Boolean skylines. Lee et al. (2007) highlights two key points: (i) dominance tests are expensive, (ii) the organization of skyline candidates and their examination strategies are critical for the efficiency of the dominance test. This second point relates specifically to Boolean skyline queries. With regards to these points, three categories of algorithms have been proposed in the literature. The first category is based on Divide and Conquer (Börzsönyi et al. 2001). Processing is done separately on subsets of data, then the results are merged. The second category relies on exploiting indices such as B-trees or R-trees to organize the data in order to avoid many dominance tests in Papadias et al. (2005), or like bitmap indices in Tan et al. (2001). The third category gathers various methods based

on sorting: Block-Nested-Loops (BNL) (Börzsönyi et al. 2001) and its improvement named Sort-Filter-Skyline (Chomicki et al. 2003, 2005), and a strategy proposed in Bartolini et al. (2008) that relies on a preordering of the tuples aimed to limit the number of elements to be accessed and compared.

We propose to implement the computation of both the typicality degrees and the skyline by parallelizing most of the calculations, so that they can be run on a CUDA processor. CUDA (Compute Unified Device Architecture) is a parallel application programming model defined by *nVidia* for their graphic cards. CUDA allows for developing programs that contain some functions to be executed on the GPU (Graphic Processing Unit) by taking advantage of the large number of computing cores. These functions, called *kernels*, are run simultaneously on different data. The number of parallel executions depends on the number of physical cores available on the GPU.

We can expect a major gain in computation speed with a large amount of cores, but only if the algorithm can be parallelized in the manner that CUDA expects. Indeed, CUDA has some drawbacks which can slow down the whole process, if they are not addressed properly. Generally, CUDA threads are designed to process an array in parallel, with one thread per cell. There are some recommendations for a thread to avoid random accesses to another cell, due to the very high latency delays of the global memory.

Only few research works deal with parallel implementation of skyline queries. Choi et al. (2012) proposes an adaptation of the Block-Nested-Loops algorithm to the CUDA architecture, taking the number of really parallel threads into consideration, in order to balance the workload. Park et al. (2009) compares two different approaches. The first approach parallelizes the BBS Branch-and-Bound algorithm proposed by Papadias et al. (2005). The second approach called *pskyline* for *parallel skyline*, aims at being much simpler, and is based on a *map-reduce* paradigm that we will describe later. Both approaches are implemented on OpenMP, and appear to have similar performances. However, in the second approach, the computation of the skyline itself is not fully parallelized. It uses a sequential function which computes a partial skyline by iterations on a subset of the tuples. Moreover, this function makes use of the memory in a way that could slow down CUDA a lot.

Bøgh et al. (2013) proposes a fast implementation of skyline queries on CUDA, taking finely into account all the particularities of the architecture. The first step is to sort the tuples according to the Manhattan distance, as the tuples at the top of the list may more likely belong to the skyline. This sorting also guarantees that a data point cannot be dominated by a successor. Then, the algorithm processes the tuples by subsets of fixed size at each major iteration. The authors have also carefully designed the computations to avoid branches in the CUDA kernels and minimize data transfers between GPU and CPU.

Unfortunately, in our problem, there is no motivation in efficiently selecting candidate tuples, by presorting the set and removing dominated tuples from the candidates. Indeed all tuples may belong to the graded skyline, but with different degrees, from 0^+ (very low membership to the skyline) to 1 (total membership to the skyline). We have to compare each tuple with every other tuples to determine its degree. So the main issue remains the efficient computation of typicality degrees and domination, as stated by Eqs. (2) and (5).

This leads to Algorithm 3, composed of two loops over the dataset $\mathcal{S}$. The outer loop computes the membership degree of all the tuples p . The inner loop compares p with all the other tuples and those who dominate p are used to get the highest dominant typicality.

Algorithm 3: Sequential Algorithm for computing the graded skyline

Require: dataset $\mathcal{S}$, typicality degree $Typ(p)$ available for all tuple p in $\mathcal{S}$

Ensure: graded skyline: $\forall p \in \mathcal{S}, Sky_{grad}(p)$

```

for all  $p \in \mathcal{S}$  do
   $best \leftarrow 0$ 
  for all  $q \in \mathcal{S}$  do
    if  $q \succ_{\mathcal{D}} p$  and  $best < Typ(q)$  then
       $best \leftarrow Typ(q)$ 
    end if
  end for
   $Sky_{grad}(p) \leftarrow 1 - best$ 
end for

```

Typicality degrees $Typ(p)$ are computed by Algorithm 4. Note that this second algorithm could be improved to avoid the loop that computes the maximum value of the frequency F , but this would hinder the parallelization.

Algorithm 4: Sequential Algorithm for computing the typicality degrees

Require: dataset $\mathcal{S}$ of cardinality n

Ensure: typicality degrees: $\forall p \in \mathcal{S}, Typ(p)$

```

// compute the frequency
for all  $p \in \mathcal{S}$  do
   $neighbors \leftarrow 0$ 
  for all  $q \in \mathcal{S}$  do
    if  $d(p, q) \leq \tau$  then
       $neighbors \leftarrow neighbors + 1$ 
    end if
  end for
   $F(p) \leftarrow neighbors/n$ 
end for
// compute the maximum frequency
 $maxF \leftarrow 0$ 
for all  $p \in \mathcal{S}$  do
  if  $F(p) > maxF$  then
     $maxF \leftarrow F(p)$ 
  end if
end for
// compute the typicality degrees
for all  $p \in \mathcal{S}$  do
   $Typ(p) \leftarrow F(p)/maxF$ 
end for

```

The algorithm proposed in Tan et al. (2001) is well-suited to the calculation of the graded skyline, because it generates for each tuple p a Boolean vector D of the size of the database, and indicating for each cell $D[q]$ whether the tuple q dominates p . Then, one has just to replace each value equal to *true* in D by the typicality of the related tuple in order to find the most typical tuple which dominates p . However, the entire algorithm is not directly parallelizable on CUDA, both because of the data structure necessary (big bitmap index), and calculations such as Boolean operations on these bitmaps.

Therefore, we propose parallel algorithms based on general parallelization principles, that can be implemented efficiently with CUDA to evaluate Formulas (2) and (5) keeping the spirit of the approach of Tan and getting some inspiration from Bøgh et al. (2013) and Park et al. (2009).

4.1 Parallel Algorithm Principles

We presented the sequential version of the graded skyline computation in Algorithms 3 and 4. They contains several *for* loops that can easily be parallelized because they are kinds of *map* and *reduce* operations. In Skeletal Parallel Programming (Cole 1991), $map(f, C)$ is an operation that applies in parallel the same function f to every element of the collection C , and returns the resulting collection. If $C = \{c_1, c_2, \dots, c_n\}$, $map(f, C)$ returns $\{f(c_1), f(c_2), \dots, f(c_n)\}$. The other operation, $reduce(f, C)$ returns the aggregation of an associative binary operator f . $reduce(f, C)$ returns the value $c_1 f c_2 f \dots f c_n$. For instance, $reduce(\lambda x, y : x + y, map(\lambda x : x^2, \{2, -3, 4\})) = 29$. Note that the *lambda* notation is necessary when functions have no name. The parameters come from the collection to process. The free variables, if any, come from the context.

Both operations, $map(f, C)$ and $reduce(f, C)$ are parallelizable with high performances in CUDA. Function f must be written as a CUDA kernel—this is a function which will run on the GPU. The dataset C shall be put into an array in the memory of the graphic card, called *device global memory*. Another array of appropriate size must be allocated to receive the results. Then, for *map*, the kernel is launched with as many instances as there are data to process. So, in theory, $map(f, C)$ has the same order of complexity than function f (multiplied by the collection size, divided by the number of parallel threads). For *reduce*, the kernel is launched hierarchically following a binary tree scheme. If f has a time complexity of $\theta(1)$, then $reduce(f, C)$ has a complexity of $\theta(\log_2 n)$ where n is the cardinality of C .

Using these principles, we propose Algorithm 5 that computes the degree of membership to the skyline of every tuple in the dataset. We use a convention in C/C++ about Booleans: *true* is equivalent to the integer value 1, and *false* to 0. This allows to replace a condition by a multiplication in the kernel. We can combine the typicality degree with the Boolean test $q \succ p$, and then get the degree by reducing with the *maximum* function.

Algorithm 5: Parallel algorithm for computing the graded skyline**Require:** dataset $\mathcal{S}$, typicality degree $Typ(p)$ available for every tuple p in $\mathcal{S}$ **Ensure:** graded skyline: $\forall p \in \mathcal{S}, Sky_{grad}(p)$

```

for all  $p \in \mathcal{S}$  do
   $best \leftarrow reduce(max, map(\lambda q : (q \succ_{\mathcal{D}} p) * Typ(q), \mathcal{S}))$ 
   $Sky_{grad}(p) \leftarrow 1 - best$ 
end for

```

The computation of the test $q \succ_{\mathcal{D}} p$ is performed by Algorithm 6. It consists of a loop over the attributes of both tuples p and q . We use the technique proposed by Bøgh et al. (2013) to avoid then-else branches. Here, we handle two Boolean variables $p_does_not_dominate_q$ and $q_does_not_dominate_p$.

$p_does_not_dominate_q$ is changed to *true* when q is better than p on at least one attribute. Reciprocally, $q_does_not_dominate_p$ is changed to *true* when p is better than q . At the end of the loop, q dominates p ($q \succ_{\mathcal{D}} p$) iff $p_does_not_dominate_q$ is *true* and $q_does_not_dominate_p$ is *false*. The loop on the attributes in Algorithm 6 has not been parallelized but has been unrolled, following CUDA guidelines (Harris 2007).

Algorithm 6: Determination of the domination $q \succ_{\mathcal{D}} p$ (Bøgh et al. 2013)**Require:** tuples p and q , each one is a sub-array of attributes $attributes[1..number\ of\ attributes]$ **Ensure:** $q \succ_{\mathcal{D}} p$

```

 $p\_does\_not\_dominate\_q \leftarrow 0$ 
 $q\_does\_not\_dominate\_p \leftarrow 0$ 
for all  $i \in 1..number\ of\ attributes$  do
   $attribute\_p \leftarrow p.attributes[i]$ 
   $attribute\_q \leftarrow q.attributes[i]$ 
   $p\_does\_not\_dominate\_q \leftarrow p\_does\_not\_dominate\_q \text{ or } (attribute\_q \succ attribute\_p)$ 
   $q\_does\_not\_dominate\_p \leftarrow q\_does\_not\_dominate\_p \text{ or } (attribute\_p \succ attribute\_q)$ 
end for
return ( $p\_does\_not\_dominate\_q > q\_does\_not\_dominate\_p$ )

```

Algorithm 7 computes the typicality degree of every tuple with the same kind of operations. We also use the Boolean $d(p, q) \leq \tau$ as a number 0 or 1 that we accumulate along the tuples. Contrary to Algorithm 4, we compute the numbers of neighbors, instead of the frequencies, because $\frac{F(p)}{maxF} = \frac{nb_neighbors(p)}{max_nb_neighbors}$.

Algorithm 7: Parallel algorithm for computing the typicality

Require: dataset $\mathcal{S}$ of cardinality n
Ensure: typicality degrees: $\forall p \in \mathcal{S}, Typ(p)$
 // Compute the number of neighbors of all tuples
for all $p \in \mathcal{S}$ **do**
 $nb_neighbors(p) \leftarrow reduce(sum, map(\lambda q : d(p, q) \leq \tau, \mathcal{S}))$
end for
 // Compute the highest number of neighbors
 $max_nb_neighbors \leftarrow reduce(max, nb_neighbors)$
 // Compute the typicality of every tuple
 $Typ \leftarrow map(\lambda p : nb_neighbors(p)/max_nb_neighbors, \mathcal{S})$

4.2 Implementation in CUDA

These algorithms are easy to code in CUDA, using the Thrust library.¹ Thrust is a CUDA Toolkit resembling the C++ Standard Template Library (STL). For instance, Thrust makes it simple to allocate an array on the GPU, to exchange data with the CPU and to launch *map* and *reduce* operations on predefined or custom kernels.

All the algorithms can be implemented this way. However, to obtain maximum speed, some computations have to be implemented more efficiently. With Thrust, each *reduce* operation brings back the result into the CPU memory. In some cases, it would be better to keep it in the GPU memory. For instance, in Algorithm 7, the instruction $F(p) \leftarrow reduce(sum, \dots)$ causes many data transfers between CPU and GPU memories. For instance, the instruction $F(p) \leftarrow reduce(sum, map(\dots))$ in Algorithm 5 is not fast enough with Thrust because of the data transfers between the GPU and the CPU when implemented with Thrust. Such memory transfers are very slow. It is necessary to rewrite some parts of the program without Thrust.

Firstly, it appears very important to organize data in the global memory of the GPU in a way that memory accesses are coalesced between threads. In CUDA, threads are grouped by warps of 32, to work together. It is recommended that a thread number i shall access a memory cell i when the thread $i + 1$ accesses the cell $i + 1$. In other cases, memory accesses cannot be grouped and cause high latencies. The memory is accessed in two places: when computing the distance between two tuples $d(p, q) \leq \tau$ and when computing the domination between two tuples in Algorithm 4. Consecutive threads will deal with consecutive tuples, written as q in the algorithm, and each thread will try to compare the first attribute of p and q , then the second, and so on. So it is necessary to put all the values for the first attribute of all tuples in sequence, then all the values of the second attribute etc., instead of putting the first tuple, then the second tuple and so on.

Other optimizations can be thought of Harris (2007) shows how to efficiently design *reduce* functions on CUDA. It is better to group *map* and *reduce* operations when the latter is applied on the former. The *map* step shall be done inside the first step

¹<http://docs.nvidia.com/cuda/thrust/>.

of *reduce*. To avoid waiting cycles in alternatives, it is worth to replace all branches by simple computations, when possible.

5 Experimental Results

We have experimented our implementation on both synthetic and real-world datasets. The first subsection shows the results obtained with data coming from an ad sales web site. The second subsection is devoted to the performances of the CUDA implementation of the graded skyline.

5.1 Application to a Real-World Dataset

The approach has been tested using a subset of the database of 845,810 ads about second hand cars from the website *Le bon coin*² from 2012. The skyline query used hereafter as an example aims at minimizing both the price and the mileage. In the query considered, we focus on small urban cars with a regular (non-diesel) engine, which corresponds to 441 ads. Figure 5 shows the result obtained. In dark grey are the points that belong the most to the skyline (membership degree between 0.8 and 1). These points are detailed in Table 2. According to the definition used, points dominated by others that are not totally typical belong to the result. It is the case for instance of ad number 916264 that is dominated by ads number 1054357 and 1229833. The identifiers in bold correspond to the points that belong to the regular skyline. One may observe that the points from Table 2 (area [0.8, 1]) are not very (or even not at all) typical. Moreover, certain features may not satisfy the user (the mileage can be very high, the price can be very low) and may look suspicious. On the other hand, Table 3, which shows an excerpt of the 0.6-cut of the graded skyline, contains more typical—thus more credible—points whose overall satisfaction remains high.

Let us mention that from such a result, we can devise different kinds of querying services such as sorting the answers by descending skyline degrees and then by descending typicality degrees or gathering answers by α -cuts over the skyline degree. As mentioned in Remark 2, it is also possible to obtain the most interesting points of the fuzzy skyline by means of a second (classical) skyline query aimed at maximizing both the membership degree and the typicality degree.

²www.leboncoin.fr.

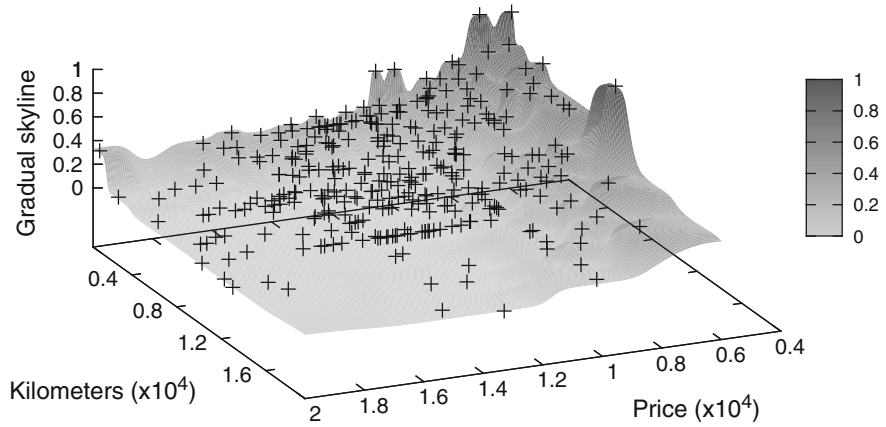


Fig. 5 3D representation of the graded skyline

Table 2 Excerpt of the database and associated degrees (skyline and typicality)

Id	Price	km	Skyline	Typicality
1211574	7000	500	1	0.352
1156771	6000	700	1	0.247
1229833	5990	10000	1	0.126
1596085	5800	162643	1	0.005
1054357	1800	118000	1	0
1333992	500	220000	1	0
1380340	800	190000	1	0
891125	1000	170000	1	0
1276388	1300	135000	1	0
916264	5990	2514000	0.874	0
1674045	6000	3500	0.753	0.315

Table 3 Excerpt of the area [0.6, 0.8]

Id	Price	km	Skyline	Typicality
1208620	6500	3300	0.716	0.363
870279	6900	1000	0.716	0.358
1334605	10500	500	0.647	0.642
1635437	9900	590	0.647	0.621
1529678	7980	650	0.647	0.458
1166077	7750	2214	0.642	0.532
1685854	7890	1000	0.642	0.458
1366336	7490	4250	0.637	0.516
981939	6500	4000	0.637	0.363
1022586	6500	7200	0.637	0.258
1267726	6500	100000	0.637	0

5.2 Application to Synthetic Data

Two kinds of measurements have been performed. We first compare the CUDA implementation to a sequential version on datasets of various cardinalities but a constant number of attributes. Then, we assess the efficiency of the CUDA version with a constant number of tuples, and an increasing number of attributes.

Our CUDA engine has four Tesla M2090, with 6 GB of memory on each. The data is very far from exhausting the memory. For instance, 1,000,000 tuples with 16 attributes represented as float numbers will occupy 64 Mbytes. To improve speed, we add padding bytes to align data, but the total size is many orders below the available memory.

Figure 6 shows a comparison between both programs, with an increasing number of tuples, from 1,000 to 1,000,000 and a constant number of 4 attributes, on logarithmic scales both in tuple number and in computation time, to show that time is a power of the number of tuples. The time is expressed in seconds. Broadly, we can say that the CUDA version is 40 times faster than the sequential version. The computation of the typicalities takes a bit less time, but has the same complexity.

We then studied the impact of the number of attributes with the same number of tuples: 100, 200 and 500 k. The result is shown in Fig. 7. The slope of the line is $\theta(n^2)$ where n is the number of tuples. These experimental results together confirm that the global computation time is $\theta(n^2 * a)$ where a the number of attributes for each tuple. Some long duration results are somewhat imprecise, due to other processes in the system.

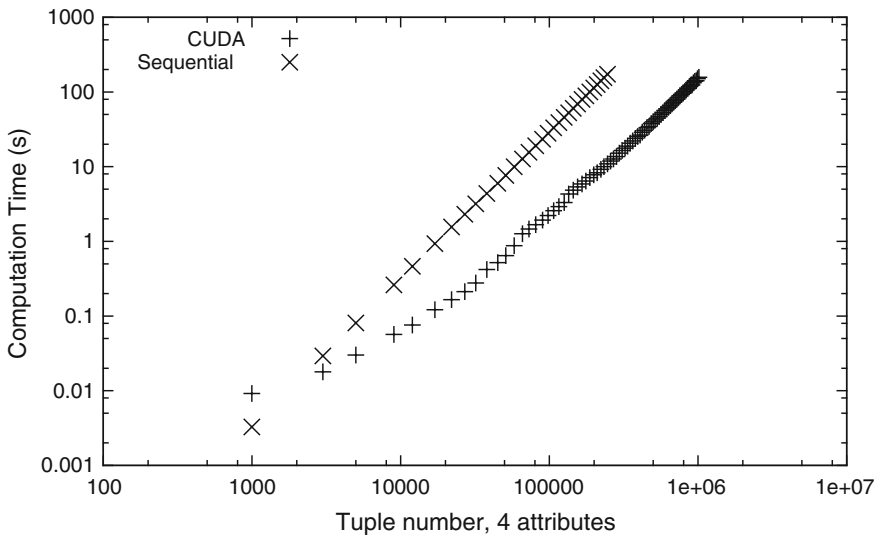


Fig. 6 CUDA compared to sequential algorithm

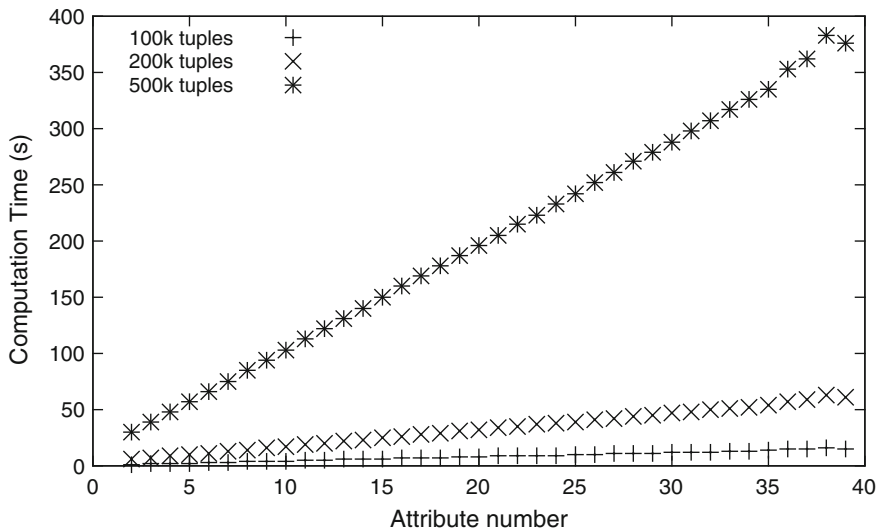


Fig. 7 Impact of the number of attributes

These results show that it is possible to calculate a graded Skyline within a reasonable time, even on a large number of tuples with many attributes.

6 Conclusion

In this paper, we have proposed a graded version of skyline queries aimed at controlling the impact of exceptions on the result (so as to prevent interesting points to be hidden because they are dominated by exceptional ones). We have also proposed a parallel implementation of the computation of the graded skyline. An improvement of our approach could consist in using more sophisticated techniques for characterizing the points, for instance, a typicality-based clustering approach (Lesot 2006) or statistical methods for detecting outliers (Hodge and Austin 2004). As a short-term perspective, we intend to devise a technique for efficiently computing the α -cut of a graded skyline, in the spirit of the derivation method described in Pivert and Bosc (2012) for fuzzy queries.

References

- Bartolini, I., Ciaccia, P., & Patella, M. (2008). Efficient sort-based skyline evaluation. *ACM Transactions on Database Systems*, 33(4), 1–49.
- Bøgh, K. S., Assent, I., & Magnani, M. (2013). Efficient GPU-based skyline computation. In *Proceedings of the Ninth International Workshop on Data Management on New Hardware, DaMoN 2013* (pp. 5:1–5:6). New York: ACM.

- Börzsönyi, S., Kossmann, D., & Stocker, K. (2001). The skyline operator. In *Proceedings of ICDE 2001* (pp. 421–430).
- Chan, C., Jagadish, H., Tan, K., Tung, A., & Zhang, Z. (2006a). Finding k-dominant skylines in high dimensional space. In *Proceedings of SIGMOD 2006* (pp. 503–514).
- Chan, C., Jagadish, H., Tan, K., Tung, A., & Zhang, Z. (2006b). On high dimensional skylines. In *Proceedings of EDBT 2006. LNCS* (Vol. 3896, pp. 478–495).
- Choi, W., Liu, L., & Yu, B. (2012). Multi-criteria decision making with skyline computation. In *2012 IEEE 13th International Conference on Information Reuse and Integration (IRI)* (pp. 316–323).
- Chomicki, J., Godfrey, P., Gryz, J., & Liang, D. (2003). Skyline with presorting. In *Proceedings of ICDE 2003* (pp. 717–719).
- Chomicki, J., Godfrey, P., Gryz, J., & Liang, D. (2005). Skyline with presorting: Theory and optimizations. In *Proceedings of IIS 2005* (pp. 595–604).
- Cole, M. (1991). *Algorithmic skeletons: Structured management of parallel computation*. Cambridge: MIT Press.
- Dubois, D. & Prade, H. (1984). On data summarization with fuzzy sets. In *Proceedings of IFSA 1993* (pp. 465–468).
- Ester, M., Kriegel, H., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han & U. M. Fayyad (Eds.), *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD 1996), Portland, Oregon, USA* (pp. 226–231). AAAI Press.
- Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179–188.
- Friedman, M., Ming, M., & Kandel, A. (1995). On the theory of typicality. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 3(2), 127–142.
- Gabel, M., Keren, D., & Schuster, A. (2013). Communication-efficient outlier detection for scale-out systems. In *Proceedings of the First International Workshop on Big Dynamic Distributed Data, Riva del Garda, Italy, 30 August 2013* (pp. 19–24).
- Goncalves, M., & Tineo, L. (2007). Fuzzy dominance skyline queries. In R. Wagner, N. Revell & G. Pernul (Eds.), *Proceedings 18th International Conference on Database and Expert Systems Applications, DEXA 2007, Regensburg, Germany, 3–7 September 2007* (Vol. 4653, pp. 469–478). Lecture notes in computer science. Heidelberg: Springer.
- Gupta, M., Gao, J., and Han, J. (2013). Community distribution outlier detection in heterogeneous information networks. In H. Blockeel, K. Kersting, S. Nijssen & F. Železný (eds.), *Machine learning and knowledge discovery in databases* (Vol. 8188, pp. 557–573). *Lecture notes in computer science*. Berlin: Springer.
- Hadjali, A., Pivert, O., & Prade, H. (2011). On different types of fuzzy skylines. In *Proceedings of ISMIS 2011* (pp. 581–591).
- Hampton, J. (1988). Overextension of conjunctive concepts: evidence for a unitary model of concept typicality and class inclusion. *Journal of Experimental Psychology: Learning Memory and Cognition*, 14(1), 12–32.
- Harris, M. (2007). Optimizing parallel reduction in cuda. Technical report, nVidia.
- Hodge, V., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2), 85–126.
- Ji, T., Yang, D., & Gao, J. (2013). Incremental local evolutionary outlier detection for dynamic social networks. In H. Blockeel, K. Kersting, S. Nijssen & F. Železný (Eds.), *Machine learning and knowledge discovery in databases* (Vol. 8189, pp. 1–15). *Lecture notes in computer science*. Berlin: Springer.
- Lee, K. C. K., Zheng, B., Li, H., & Lee, W.-C. (2007). Approaching the skyline in z order. In *VLDB* (pp. 279–290).
- Lesot, M. (2006). Typicality-based clustering. *International Journal of Information technology and Intelligent Computing*, 12, 279–292.
- Lin, X., Yuan, Y., Zhang, Q., & Zhang, Y. (2007). Selecting stars: The k most representative skyline operator. In *Proceedings of the ICDE 2007* (pp. 86–95).

- Niu, Z., Shi, S., Sun, J., & He, X. (2011). A survey of outlier detection methodologies and their applications. In *Proceedings of Third International Conference Artificial Intelligence and Computational Intelligence, Part I, AICI 2011, Taiyuan, China, September 24–25, 2011* (pp. 380–387).
- Osherson, D., & Smith, E. (1997). On typicality and vagueness. *Cognition*, 64, 189–206.
- Papadias, D., Tao, Y., Fu, G., & Seeger, B. (2005). Progressive skyline computation in database systems. *ACM Transactions on Database Systems*, 30(1), 2005.
- Park, S., Kim, T., Park, J., Kim, J., & Im, H. (2009). Parallel skyline computation on multicore architectures. In *ICDE* (pp. 760–771).
- Pivert, O., & Bosc, P. (2012). *Fuzzy preference queries to relational databases*. London: Imperial College Press.
- Rojas, W. U., Boizumault, P., Loudni, S., Crémilleux, B., and Lepailleur, A. (2014). Mining (soft-) skypatterns using dynamic CSP. In H. Simonis (ed.), *Proceedings of 11th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR 2014, Cork, Ireland, 19–23 May 2014* (Vol. 8451, pp. 71–87). *Lecture notes in computer science*. Cham: Springer.
- Rosch, E., & Mervis, C. (1975). Family resemblance: Studies in the internal structure of categories. *Cogn. Psychol.*, 7, 573–605.
- Tan, K.-L., Eng, P.-K., & Ooi, B. C. (2001). Efficient progressive skyline computation. In *Proceedings of VLDB 2001* (pp. 301–310).
- Tao, Y., Ding, L., Lin, X., & Pei, J. (2009). Distance-based representative skyline. In Y. E. Ioannidis, D. L. Lee & R. T. Ng (Eds.), *ICDE* (pp. 892–903). IEEE.
- Yager, R. (1997). A note on a fuzzy measure of typicality. *International Journal of Intelligent Systems*, 12(3), 233–249.
- Zadeh, L. A. (1982). A note on prototype theory and fuzzy sets. *Cognition*, 12, 291–298.
- Zadeh, L. A. (1984). A computational theory of dispositions. In Y. Wilks (Ed.), *COLING* (pp. 312–318). ACL.
- Zadeh, L. (1987). A computational theory of dispositions. *International Journal of Intelligent Systems*, 2, 39–63.
- Zhang, J. (2013). Advancements of outlier detection: A survey. *EAI Endorsed Transactions on Scalable Information Systems*, 1, e2.
- Zimek, A., Gaudet, M., Campello, R. J. G. B., & Sander, J. (2013). Subsampling for efficient and effective unsupervised outlier detection ensembles. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, 11–14 August 2013* (pp. 428–436).
- Zimek, A., Schubert, E., & Kriegel, H. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining*, 5(5), 363–387.

Advances in Knowledge Discovery and Management

Volume 6

Guillet, F.; Pinaud, B.; Venturini, G. (Eds.)

2017, XXI, 278 p. 81 illus., 61 illus. in color., Hardcover

ISBN: 978-3-319-45762-8