Fast Redescription Mining Using Locality-Sensitive Hashing

Maiju Karjalainen[0000−0001−5885−3143] (✉), Esther Galbrun[0000−0003−2585−9252], and Pauli Miettinen[0000−0003−2271−316X]

University of Eastern Finland firstname.lastname@uef.fi

Abstract. Redescription mining is a data analysis technique that has found applications in diverse fields. The most used redescription mining approaches involve two phases: finding matching pairs among data attributes and extending the pairs. This process is relatively efficient when the number of attributes remains limited and when the attributes are Boolean, but becomes almost intractable when the data consist of many numerical attributes. In this paper, we present new algorithms that perform the matching and extension orders of magnitude faster than the existing approaches. Our algorithms are based on locality-sensitive hashing with a tailored approach to handle the discretisation of numerical attributes as used in redescription mining.

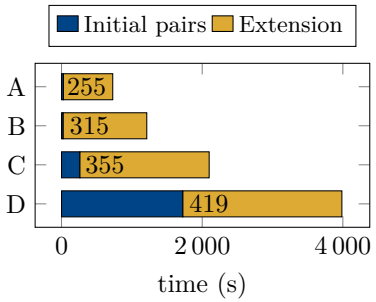
Keywords: Redescription mining · Locality-Sensitive hashing

1 Introduction

A redescription is a pattern that characterises roughly the same entities in two different ways, and redescription mining is the task of automatically extracting redescriptions from the input dataset, given user-defined constraints. Redescription mining has found applications in various fields of science, such as ecometrics. Ecometrics aims to identify and model the functional relationships between traits of organisms and their environments [5, 7]. For instance, the teeth of large plant-eating mammals are adapted to the food that is available in their environment, which in turn depends on the climatic conditions, potentially allowing one to reason about the climate in the past based on the fossil record.

To apply redescription mining in this context, the entities in the dataset represent localities, with two sets of attributes recording respectively the distribution of dental traits among species and the climatic conditions at each locality [11, 20]. Galbrun et al. [11] mined redescriptions from this dataset using the `ReReMi` [8] algorithm in about 50 minutes on a commodity laptop. Replicating the experiment, we obtained a comparable time (bar D of Fig. 1 (left)).

In contrast, the two top bars represent our proposed method, `Fier` (Fast Initialisation and Extension of Redescriptions), based on locality-sensitive hashing (LSH). Our approach uses the same two-phase procedure as `ReReMi`: it finds initial pairs in the first phase, and extends them in the second phase (see Sect. 2 for details). The top bar, `Fier_full`, uses a LSH-based approach for both phases,



$$q_L = (([\text{Hyp3} \leq 0.458] \wedge [0.061 \leq \text{AL} \leq 0.188]) \vee [0.032 \leq \text{Hod3} \leq 0.059]) \wedge [\text{OL} \leq 0.643]$$
$$q_R = [68.0 \leq \text{TIso}] \wedge [640 \leq \text{PTotY}]$$
$$J = 0.62 \qquad |\mathcal{E}_{11}| = 6185$$

Fig. 1. Left: Running times on the DentalW dataset for finding initial pairs (blue) and extending pairs (yellow) using (A) the proposed algorithm (`Fier_full`), (B) `Fier_init` for initial pairs and `ReReMi` for extensions, (C) `ReReMi` with pre-bucketing (`ReReMiBkt`) and (D) standard `ReReMi`. The number within each bar indicates how many initial pairs were found. Right: Example redescription.

while the second bar, `Fier_init`, uses LSH approach only for the first part. The third bar represents a straightforward speed-up of initial pair generation of `ReReMi` that we use as a baseline. It is obtained by replacing on-the-fly discretisation of numerical attributes by discretisation as a pre-processing step.

Figure 1 (left) shows that using `Fier` the whole mining process is completed before the standard `ReReMi` has even finished mining the initial pairs. The process is reduced from 66 min to mere 12 min; mining the initial pairs takes only 25 s.

`Fier`, being significantly faster than the existing methods, allows the use of redescription mining on even larger datasets. Alternatively, the speedup affords more responsive interactive redescription mining [10] and quickly testing parameter and constraint combinations to 'get a feel' for what kind of setup to use with the more exhaustive `ReReMi` algorithm. The price `Fier` has to pay for its speed is its probabilistic nature: unlike `ReReMi`, it does not guarantee to return the best initial pairs nor the (locally) best extensions.

Redescription mining. The input of redescription mining is a pair of *data tables* which we refer to as the left-hand side and right-hand side, denoted respectively $\mathbf{D}_L$ and $\mathbf{D}_R$, over the same *entities*, denoted $\mathcal{E}$. Then, a redescription is a pair of *queries*, denoted q_L and q_R, consisting of *literals* over the attributes of the corresponding table, combined with logical conjunction and disjunction operators, possibly involving negations. The set of entities that satisfy both queries, only the left-hand side query, only the right-hand side query, and neither of them are denoted respectively $\mathcal{E}_{11}$, $\mathcal{E}_{10}$, $\mathcal{E}_{01}$ and $\mathcal{E}_{00}$. The set $\mathcal{E}_{11}$ is also called the support of the redescription, and more in general, we call *support* (supp) of a literal or query the set of entities that satisfy it. The Jaccard index between the sets of entities that satisfy either query is called the *accuracy* of the redescription (we use the terms Jaccard and accuracy interchangeably) and is defined

$$J(q_L, q_R) = \frac{|\mathcal{E}_{11}|}{|\mathcal{E}_{10}| + |\mathcal{E}_{11}| + |\mathcal{E}_{01}|} .$$

Since it is meant to provide two ways to characterise roughly the same entities, accuracy is the main measure of the quality of a redescription. Further constraints that can be applied to redescriptions include a maximum threshold on the number of attributes they involve, since long queries might be difficult to interpret, and minimum thresholds on the number of entities that satisfy both queries and that satisfy neither, since redescriptions that characterise too few or too many of the entities are typically not considered interesting.

Going back to the application in ecometrics, the right-hand side query of the example redescription shown in Fig. 1 (right) characterises the climatic conditions encountered at the locality, in this case requiring high isothermality (TIso) and high annual precipitation (PTotY). The left-hand side query (q_L) is more complex and involves four literals over dental traits, requiring a limited prevalence of hypsodont species (Hyp3) and fairly low fraction of species with acute lophs (AL), or a very low fraction of hypsohorizodont species (Hod3), but allowing up to a rather large fraction of species with obtuse lophs (OL). Hypsodont and hypsohorizodont species refer to species with elongated teeth respectively along the vertical and the horizontal dimension, while acute and obtuse lophs refer respectively to the presence of sharp and blunt edges on the tooth across the chewing direction. In the considered dataset, $|\mathcal{E}_{11}| = 6\,185$ of the 28 886 localities satisfy both queries, representing about $J = 62\,\%$ of the 10 011 localities that satisfy at least one of the queries.

Related work. In the two decades since redescription mining was introduced [26], various algorithms have been proposed for this task, including based on decision tree induction [26, 27, 22], itemset mining [12], as well as iterative greedy heuristics [12, 8]. The `ReReMi` algorithm [8] belongs to the latter family.

Other lines of work have focused for instance on selecting a good set of redescriptions [16], designing an interactive tool [10] and providing differentially private methods for redescription mining [23, 17]. Meanwhile, redescription mining has been used in applications from domains as diverse as medicine [24], political sciences [9] and palaeontology [11, 20].

Also relevant here is the work of Meeng and Knobbe [21] studying the impact of different discretisation strategies on subgroup discovery, a task very similar to redescription mining.

Locality-sensitive hashing was introduced for efficient nearest neighbour search [15]. It has proven to be very useful for various tasks, such as collaborative filtering [4], clustering [2] and privacy preservation [6]. The work most related to the present work is the early application to faster association rule mining [3].

2 The Algorithm

We divide our full algorithm into two parts for the two phases: `Fier_init` computes the initial pairs and `Fier_ext` the extensions. The full algorithm is called `Fier_full`. Before presenting the algorithms, we provide short primers on the `ReReMi` algorithm and on locality-sensitive hashing.

2.1 The ReReMi algorithm

The algorithm on which we build, ReReMi [8], mines redescriptions in two main phases. In the first phase, the InitialPairs method returns candidate redescriptions with a single literal on either side. In the second phase, up to a chosen number of the best initial pairs are considered. Each is extended in turn through a greedy process that iteratively appends a literal to either of the queries to produce more refined and accurate redescriptions. The ReReMi algorithm can handle Boolean, categorical and numerical attributes. In particular, when forming literals for a numerical attribute, it performs on-the-fly discretisation, trying to find the interval that yields the best accuracy for the considered candidate, rather than using pre-determined buckets.

Going a bit further into technical details, the way ReReMi finds the initial pairs is simple but time consuming: Iterate over all pairs of attributes, one from either side; For each such pair, consider the possible pairs of literals over the two attributes; Compute the support and accuracy of the corresponding redescription and keep the best matches that satisfy support constraints.

Given a redescription to extend, ReReMi considers every available literal and the candidate extensions obtained by appending it (or its negation if allowed) to the current query on the corresponding side, with either the disjunction or the conjunction operator. The support and accuracy of each candidate extension are computed, the best one-step extensions are selected and extended in turn, for each one considering again all the possible ways to extend it with the remaining available literals. Starting with an initial pair, this greedy extension process is applied until no further improvement (in terms of accuracy) can be found or the maximum length of the queries (in terms of the number of involved literals) has been reached. The algorithm then proceeds with the next initial pair.

Computing the support and accuracy for a pair of literals involves computing the cardinality of the intersection and union of the sets of entities satisfying either literal. Similarly for computing the support and accuracy for an extension, although it has been shown [8] that not all entities can change status in the support of an extension. For instance, when extending a redescription by adding a literal to the left-hand side with a conjunction, entities that do not satisfy the current left-hand side query will not satisfy the extended one either. This fact is used to quickly identify the entities that need to be considered when performing the set operations to determine the quality of an extension.

2.2 Primer on LSH

The idea of Locality-Sensitive Hashing (LSH) is to calculate hash values for input items so that similar items get the same hash value with a high probability. The type of hash function depends on the similarity measure used. In this work we use two measures of similarity between binary vectors representing support sets, namely the Jaccard similarity and the Hamming distance.

When using the Jaccard similarity, the corresponding hashing technique is called minhashing [1]. To compute minhash values, we use k random hash

functions $h_1, \ldots, h_k$ that map the row indices to values between 0 and the maximum number of rows. To calculate the minhash signature of a vector, we consider the indices of all the rows containing a 1. For every hash function we obtain the hash values for all of these rows and take the smallest value. These concatenated smallest hash values make up the signature of the vector [19].

When using the Hamming distance, we calculate the length-k signature of the vector as the values of the binary vector in k random indices.

To match two binary vectors based on their signatures, the signatures are divided into b smaller sections called bands, each containing r hash values (total $k = r \cdot b$). Two vectors are paired if all r hash values match in at least one band.

The parameters b and r determine the matching probability, i.e. the probability with which two vectors will be paired. If the similarity of the vectors is p, then the matching probability is $1 - (1 - p^r)^b$. The relation between the similarity and the matching probability follows an S-curve; its threshold point can be approximated as $p_T = (1/b)^{1/r}$ [19]. With this approximation, only two parameters need to be set and the third one can be calculated accordingly. To increase the chances of finding pairs with an accuracy above a desired threshold, one can use a slightly lower value, whereas a slightly higher value yields a faster algorithm finding fewer pairs.

2.3 Finding Initial Pairs

For simplicity we first focus on Boolean attributes. The pseudocode for the algorithm is given in Algorithm 1. Starting with the left-hand side dataset $\mathbf{D}_L$, we go through the literals and obtain r_J minhash signatures for each of the corresponding binary support vectors. Literals with the same signature get hashed into the same bin, and we keep track of which side each literal belongs to.

We go through the right-hand side dataset $\mathbf{D}_R$ similarly, except that we discard signatures that point to an empty bin. After calculating signatures for all literals, we go through each signature bin and form candidate pairs between the literals from $\mathbf{D}_L$ and $\mathbf{D}_R$ hashed into that bin.

We repeat this process b_J times, calculating a different minhash signature for the literals for each band. If the same two literals already formed a pair in a previous band, we do not consider it again. After the candidate pairs have been formed, we calculate their actual accuracy and support, and we filter out any candidate that does not satisfy user-defined support and accuracy thresholds.

Building initial pairs from categorical attributes is slightly more complicated, as we can create several different literals by considering the different categories, separately or combined. We first consider the literals obtained by considering each category of an attribute a separately and calculate their minhash signatures the same way as for Boolean literals. Next, we create combinations of the categories by joining them with the disjunction operator, e.g. $[a = c_1 \vee a = c_2]$. The number of categories that are combined can be limited with a user-defined parameter. Since we already calculated the signatures for the separate categories, we can obtain the signature for a combination by simply taking the smallest minhash value among the categories included in the combination.

Algorithm 1 Generating initial pairs from Boolean attributes

Input: Data $\mathcal{D} = (\mathbf{D}_L, \mathbf{D}_R)$, number of bands b_J, width of bands r_J
Output: A set of initial redescriptions $\mathcal{R} = \{(q_L, q_R)\}$

```
1: function Fier_init(D, b, r)
2:     for band = 1, ..., b do
3:         B ← ∅
4:         generate r hash functions h_1, ..., h_r that permute the row indexes
5:         for side in D do
6:             for attribute a in side do
7:                 sig ← (min(h_1(supp(a))), ..., min(h_r(supp(a))))
8:                 B[sig] ← B[sig] ∪ {a}
9:             for all sig ∈ B do
10:                R ← R ∪ {pairs of literals (a_L, a_R) in B[sig], a_L ∈ D_L, a_R ∈ D_R}
11:    return R
```

Creating literals out of numerical attributes is similar to categorical attributes, but by specifying intervals such as $[a \leq x]$ or $[x \leq a \leq y]$, instead of categories. We determine the intervals in two steps. First, we discretise the attribute values into n_b small buckets $[buk_l \leq y \leq buk_u]$ (line 5 in Algorithm 2) and calculate signatures for the literals corresponding to the separate buckets. Second, we create extended intervals by combining consecutive buckets, and calculate the signature for an extended interval by taking the smallest minhash values of the buckets included in the interval (line 13). We use support thresholds supp_{min} and supp_{out} to limit the size of intervals so that they do not cover too few or too many entities. The supp_{out} threshold defines the minimum number of entities not covered by the literal.

The discretisation of attribute values into buckets can be done in different ways. We found that the bucketing method did not have a significant impact on the running time nor on the quality of the results. In our experiments we used equal-height binning, where each bucket covers a similar number of entities.

To avoid creating multiple pairs that are very similar to each other, we filter out any subinterval that has the same signature as the interval it is contained in. We do this by keeping track of the signatures we have seen for an attribute (within one band) as we iterate over the intervals.

We iterate over the intervals by progressively narrowing them down. Considering the ordered bucket edges $buk_0, buk_1, \ldots, buk_{n_b}$, we start by setting buk_0 as the lower and buk_{n_b} as the upper bound of the interval. We lower the upper bound by removing buckets from the top $buk_{n_b-1}, buk_{n_b-2}, \ldots$ until the support criteria is met. We calculate the signature as the smallest minhash values of the buckets contained in the interval. We lower the upper bound until the support of the interval is below the minimum support criterion. Then, we set buk_1 as the lower bound and repeat the process until we have gone through all lower bounds. We filter out subintervals of already seen intervals during this process, by discarding those intervals whose signature we have already seen, and whose

Algorithm 2 Generating initial pairs from numerical attributes

Input: Data $\mathcal{D} = (\mathbf{D}_L, \mathbf{D}_R)$, number of bands b_J, width of bands r_J, number of buckets n_b, min. nb. of entities not in the support supp_{out}, minimum support $\text{supp}_{\min}$
Output: A set of initial redescriptions $\mathcal{R} = \{(q_L, q_R)\}$

1: **function** `Fier_init`$(\mathcal{D}, b, r, n_b, \text{supp}_{\text{out}}, \text{supp}_{\min})$
2: **for** $band = 1, \ldots, b$ **do**
3: **for** data table in $\mathcal{D}$ **do**
4: **for** attribute a in data table **do**
5: $[buk_0, buk_2, \ldots, buk_{n_b}] \leftarrow$ DISCRETISE(a, n_b)
6: **for** $k = 1, \ldots, n_b$ **do**
7: Signatures$[k, :] \leftarrow r_J$ minhash values for $[buk_{k-1} \leq a \leq buk_k]$
8: $S \leftarrow \emptyset$
9: **for** $l = 0, \ldots, n_b - 1$ **do**
10: $u \leftarrow n_b$
11: **while** $\text{supp}([buk_l \leq a \leq buk_u]) > |\mathcal{E}| - \text{supp}_{\text{out}}$ **do** $u \leftarrow u - 1$
12: **while** $\text{supp}([buk_l \leq a \leq buk_u]) \geq \text{supp}_{\min}$ **do**
13: $sig \leftarrow \min_{i=l+1}^{u}(\text{Signatures}[i, :])$
14: **if** $u > S[sig]$ **then**
15: $B[sig] \leftarrow B[sig] \cup \{[buk_l \leq a \leq buk_u]\}$
16: $S[sig] \leftarrow u$
17: $u \leftarrow u - 1$
18: **for all** $sig \in B$ **do**
19: $\mathcal{R} \leftarrow \mathcal{R} \cup \{\text{pairs of literals } ([buk_l \leq a_L \leq buk_u], [buk'_l \leq a_R \leq buk'_u]) \text{ in } B[sig],\ a_L \in \mathbf{D}_L, a_R \in \mathbf{D}_R\}$
20: **return** $\mathcal{R}$

upper bound is smaller than the largest seen upper bound for that signature (line 14).

2.4 Extending Initial Pairs

Computing signatures for literals. We start the process by computing the Hamming signatures for all data columns. For this we first randomly select indices over the number of rows in the data, so that we have b_H sets of length r_H random indices to create all signatures from. Note that these are different parameters than b_J and r_J used with the initial pair mining. For each literal to extend with, we store b_H sets of length r_H signatures.

The discretisation of the numerical attributes is done differently from the initial pair search. We start with a small number of buckets to discretise the values into and compute the Hamming signature corresponding to each bucket. Next, we double the number of buckets, perform the discretisation again and compute new signatures. We repeat this process a fixed number of times. We store the signatures for each bucket of each attribute, but we do not store the actual intervals, as they will be re-calculated later. This approach has only a modest effect on the running time, as we only need to compute the signatures

for literals once. On the other hand, it sidesteps the issue of choosing the correct bucketing by using many different ones.

The target vector. As mentioned earlier, only some entities in the data matter when extending a redescription. Let us consider a redescription (q_L, q_R) which we want to extend on the right side with a conjunction. To improve $J(q_L, q_R)$, we want to find a literal that is satisfied by entities in $\mathcal{E}_{11}$ (so that the numerator does not shrink) and not by entities in $\mathcal{E}_{01}$ (so that the denominator shrinks). Entities in $\mathcal{E}_{10}$ and $\mathcal{E}_{00}$ do not matter, and we call these rows the 'don't-care' rows. In terms of LSH, we can see this as having a target vector that has 1s in rows corresponding to $\mathcal{E}_{11}$ and 0s in rows corresponding to $\mathcal{E}_{01}$. We can easily find data columns that match this target vector using the Hamming distance restricted to these rows (we use Hamming instead of Jaccard to reward matches on 1s and on 0s equally). Similarly, when extending the left side with a disjunction, we want to find literals that have 1s in rows corresponding to $\mathcal{E}_{01}$ and 0s in rows corresponding to $\mathcal{E}_{00}$, with $\mathcal{E}_{11}$ and $\mathcal{E}_{10}$ being the 'don't-care' rows. When extending the right side either by conjunction or disjunction the target vectors are otherwise same as with the left side, except that $\mathcal{E}_{10}$ and $\mathcal{E}_{01}$ switch places. As these subsets of rows are different for each redescription to be extended, we would have to re-compute the signatures for all data columns considering every candidate redescription and both connectives used to extend it on each side. This would clearly cancel the speed benefit. Instead, we replace the 'don't-care' rows in the target vector with 0s when extending with conjunction and with 1 when extending with disjunction. This way we only need to compute the signatures for the data columns once and we can use the same signatures for all extensions.

Extending redescriptions. The algorithm for extending the redescriptions, denoted `Fier_ext`, is presented in Algorithm 3. We start by storing the initial pairs in a priority queue Q, using accuracy as the key. We always expand the redescription with the current highest accuracy. A redescription can be extended on either of its sides, with a conjunction or a disjunction operator. The corresponding target vectors are computed by ComputeTargetVector. ComputeCandidates uses LSH to find the columns that match the target vectors, recording also the associated side and operator. The same column can match multiple times (in different bands, or different buckets for a numerical attribute), only the first match is stored. The actual extensions are then computed using the same approach as with `ReReMi` (i.e. using the cut-point method [8] for numerical attributes) and the best extension is stored and pushed back to Q if it has not already been extended to maximum length. Compared to `ReReMi`, the speed of `Fier_ext` benefits from not trying every column as a candidate extension, but only those that match with LSH.

2.5 Time Complexity

The time complexity for mining initial pairs from Boolean attributes is $O(N \cdot b \cdot r \cdot |\mathcal{E}| + n_L \cdot n_R)$, where $|\mathcal{E}|$ is the number of entities, N is the total number of

Algorithm 3 `Fier_ext`: Extend redescriptions

Input: Initial pairs $\mathcal{P}$, buckets with signatures for each possible extension B, maximum length of a redescription t
Output: A set of extended redescriptions $\mathcal{R}$

```
1: function Fier_ext(P, B, t)
2:     Q ← heapify the list of initial pairs with accuracy as the key; R ← ∅
3:     while Q ≠ ∅ do
4:         (q_L, q_R) ← Q.pop()
5:         V ← ComputeTargetVectors((q_L, q_R))
6:         C ← ComputeCandidates(V, B)
7:         for (column, side, operator) ∈ C do
8:             E ← ComputeExtensions((q_L, q_R), column, side, operator)
9:         if E ≠ ∅ then
10:            best ← extension with the highest Jaccard
11:            R ← R ∪ {best}
12:            if len(best) < t then Q.push(best)
13:    return R
```

attributes, $b \cdot r$ is the total number of hash functions in LSH, n_L is the largest number of left-hand side literals hashed to the same bin, and n_R is the largest number of right-hand side literals hashed to the same bin. The first term is for computing the signatures. The second term is for building the pairs; in the worst case it is quadratic, but in practise much smaller.

The time complexity for numerical attributes is $O\big(N(|\mathcal{E}| \log |\mathcal{E}| + n_b \cdot b \cdot r \cdot |\mathcal{E}| + n_b^2) + n_L \cdot n_R\big)$; assuming n_b, b, and r are constants, this becomes $O(N\,|\mathcal{E}| \log |\mathcal{E}| + n_L \cdot n_R)$. Bucketing with equal-height binning takes $O\big(N(|\mathcal{E}| \log |\mathcal{E}|)\big)$. We can think of each bucket as creating a new literal, so creating the signatures takes $O(N \cdot n_b \cdot b \cdot r\,|\mathcal{E}|)$. Subinterval filtering (Algorithm 2 line 9) takes $O(n_b^2)$ for each of the N attributes. Similarly to Boolean attributes we assume at most n_L and n_R literals hashed to the same bin. The time complexity for the categorical attributes falls between the Boolean and numerical attributes.

The time complexity for mining the extensions is $\tau \cdot n_c \cdot T_R$, where τ is the number of extensions done (at most the number of initial pairs times the maximum length of a redescription), n_c is the maximum number of candidate extensions for a target vector (at most the number of literals in the data but typically much fewer), and T_R is the time `ReReMi` takes to compute an extension for a given redescription and a literal.

3 Experimental Evaluation

The experimental evaluation asserts that the results obtained by `Fier` are comparable to `ReReMi` in quality and that `Fier` is significantly faster than `ReReMi`. It is split into three parts: generating the initial pairs (Sect. 3.2), extending initial pairs (Sect. 3.3) and the full system (Sect. 3.4). We also test the sensitivity of `Fier` to the parameters of locality-sensitive hashing. We did not test the effects

Table 1. dataset properties.

dataset	$\lvert\mathcal{E}\rvert$, entities	$\mathbf{D}_L$ attributes	$\mathbf{D}_R$ attributes
EuroClim	2 575	12 numerical	12 numerical
NoisyClim	2 575	36 numerical	36 numerical
MammalsW	54 013	48 numerical	4 754 Boolean
Ethno	1 267	23 numerical and categorical	90 numerical
DentalW	28 886	11 numerical	19 numerical
DentalA	6 404	7 numerical	19 numerical
VAA	1 656	9 categorical	107 categorical
CMS_d	d	152 numerical	452 numerical

of standard redescription mining parameters, such as $\text{supp}_{\min}$, that are common to all algorithms of this family and not a property of our proposed approach.

3.1 Experimental Setup

We used twelve different datasets. Their properties are listed in Table 1.

The MammalsW dataset contains information about which mammal species inhabit which areas of the world on one side, and climate information on the other side [14]. That same climate data restricted to Europe, with monthly average temperatures on one side and monthly total precipitations on the other, constitutes the EuroClim dataset. NoisyClim contains three copies of the columns of EuroClim. The first copy is as is, the second and third copies are perturbed by adding noise distributed following $\mathcal{N}(0,1)$ and $\mathcal{N}(0,3)$ respectively.

DentalW is the dataset from the ecometrics study [11] presented in the introductory example. One side contains dental traits aggregated over the resident large herbivorous species whereas the other side contains climate information, at localities across the world. DentalA is a similar dataset, restricted to Southeast Asia and China and considering a slightly different set of dental traits [20].

Ethno is based on Murdock's Ethnographic Atlas [25]. It contains ecological attributes on the left-hand side, and nominal attributes recording cultural features of various tribes on the right hand side. VAA [13] contains the background information and answers to questions regarding political opinions of candidates to the Finnish parliamentary elections of 2011, as collected by an online voting advice application (VAA).

Finally, the CMS data are based on the synthetic data released by the Centers for Medicare and Medicaid Services.[1] We used five versions of the data, all having the same attributes, but increasing number of rows (7199, 14 414, 29 136, 58 204 and 116 395). The differently sized versions of CMS are indicated with the numbers of rows in subscript where relevant.

All algorithms are implemented in Python and the experiments (except when mentioned otherwise) are run with Python 3.6.8 on a machine with 2 AMD EPYC

[1] `https://www.cms.gov/Research-Statistics-Data-and-Systems/Downloadable-Public-Use-Files/SynPUFs/DE_Syn_PUF`

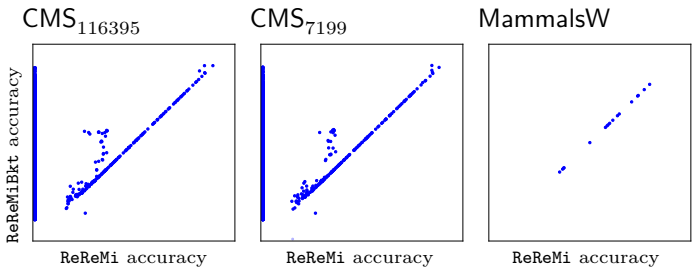


Fig. 2. Comparing the accuracy of pairs found by `ReReMiBkt` and `ReReMi`. Each dot represents a pair of columns, and its location indicates the highest-accuracy initial pair `ReReMiBkt` and `ReReMi`.

7702 processors with 64 cores each and 1 TB of main memory. All experiments were run single-threaded. For `ReReMi`, we used the publicly available version.[2] Our code is publicly available.[3]

Unless otherwise noted, all comparative experiments were run with parameters $b_J = 40$, $r_J = 10$, $\text{supp}_{\min} = 0.1\,|\mathcal{E}|$, $\text{supp}_{\text{out}} = 0.3\,|\mathcal{E}|$ and with $n_b = 40$ buckets for numerical attributes. An exception to this are the DentalW and DentalA datasets, for which we used the parameters $b_J = 40$, $r_J = 5$, $n_b = 50$ and $\text{supp}_{\min} = 0.01\,|\mathcal{E}|$. For DentalW we set supp_{out} to $0.6\,|\mathcal{E}|$ and for DentalA to $0.3\,|\mathcal{E}|$. When using `Fier`, supp_{out} was further multiplied by 1.15 for DentalW and 1.2 for DentalA. Further experimental results are presented in the technical report [18].

3.2 Finding Initial Pairs

Quality of the pairs. We compare the quality of the pairs found by `Fier_init` and `ReReMiBkt`, the `ReReMi` algorithm with pre-bucketing. `ReReMiBkt` gives more comparable running times and the quality of the initial pairs was essentially the same as with `ReReMi`, as shown in Fig. 2 for the CMS_{116395}, CMS_{7199} and MammalsW datasets. They show that `ReReMiBkt` and `ReReMi` get the same results with the MammalsW dataset and there are no dots in the x-axis, which would indicate pairs found only by `ReReMi`. With the CMS datasets, `ReReMi` and `ReReMiBkt` find somewhat different initial pairs. `ReReMiBkt` finds almost always equally good initial pairs as `ReReMi`, as well as many intial pairs that `ReReMi` does not find. This indicates that comparison to `ReReMiBkt` for the quality of the initial pairs is fair.

Results on the CMS, EuroClim, NoisyClim, Ethno, MammalsW, VAA, DentalW and DentalA datasets are shown in Fig. 3.

The optimal outcome for these experiments is a diagonal line, indicating that `Fier_init` finds pairs equivalent to those found by `ReReMiBkt`. This is mostly

[2] `https://pypi.org/project/python-clired/`
[3] `https://doi.org/10.5281/zenodo.11545892`

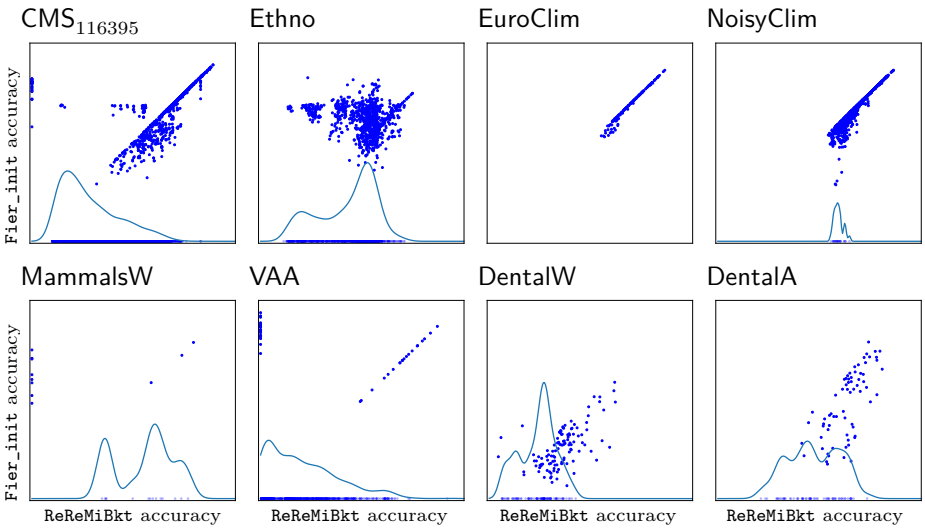


Fig. 3. Comparing the accuracy of pairs found by `Fier_init` and `ReReMiBkt`. Each dot represents a pair of columns, and its location indicates the highest-accuracy initial pair `Fier_init` and `ReReMiBkt` found. The light blue line at the bottom shows the density of the dots that lie along the x-axis. All axes range across the unit interval.

the case. With CMS, MammalsW and VAA, there are also some dots along the x-axis, meaning that there are pairs of columns that `Fier` did not find, but most of those dots are at the left end of the x-axis, as indicated by the density plots (light blue). That is, they have low accuracy. DentalW and DentalA show weaker correlation, but still a clear diagonal pattern. Ethno is an outlier, where `Fier_init` does not find some pairs of literals although they have reasonably high accuracy (indicated by the peak around the middle of the x-axis), but also finds pairs that have significantly higher accuracy than what `ReReMiBkt` returns.

Speed of execution. The running times on different datasets are presented in Table 2. The running times for DentalW and DentalA are not comparable to other datasets as they were carried out using a different setup (to reflect [11, 20]).

Overall, we see that `Fier_init` is consistently an order of magnitude faster than `ReReMiBkt`, and often two orders of magnitude faster than `ReReMi`. The only exception to this is VAA; the small size of the data and categorical attributes mean that the overhead of the hashing dominates the running time. Another case where the difference is smaller is MammalsW; here, the Boolean attributes mean less benefit for `Fier_init`, and the on-the-fly discretisation of numerical attributes by `ReReMi` is actually faster than pre-bucketing of `ReReMiBkt`.

Looking at the different-sized CMS datasets, we also see that the running time of `Fier_init` grows linearly with the number of rows, matching the asymptotic runtime. Even though `Fier_init` uses more complex data structures for hashing, its memory consumption was only slightly larger than that of `ReReMi` ($\approx$30 %).

Table 2. Comparing the running times (in seconds) of the algorithms on all datasets. Running times with DentalA and DentalW are not comparable with others, as they were run using a different setup.

Algorithm	EuroClim	NoisyClim	Ethno	VAA	DentalW	DentalA
`ReReMi`	1 184.89	12 922.03	9 069.96	0.56	1 569.01	1 960.38
`ReReMiBkt`	341.90	3 163.01	102.70	0.68	320.09	233.02
`Fier_init`	10.50	68.52	13.65	1.87	20.66	18.24
	MammalsW	CMS_{7199}	CMS_{14414}	CMS_{29136}	CMS_{58204}	CMS_{116395}
`ReReMi`	138.52	336.61	436.53	626.02	953.50	1 961.12
`ReReMiBkt`	143.35	308.39	418.40	610.99	943.21	1 899.83
`Fier_init`	47.66	21.22	40.22	71.02	145.20	290.54

Table 3. The effect of LSH parameters to running time (left) and average Jaccard of the fifth best result $\bar{J}$@5 (right) depending on the number (b) and the width (r) of bands in LSH, for five repetitions and using EuroClim data.

	time (s) r_J						$\bar{J}$@5 r_J					
b_J	3	5	7	10	12	15	3	5	7	10	12	15
20	33.89	21.34	15.00	6.94	5.68	5.14	0.69	0.70	0.63	0.69	0.50	0.63
40	45.71	33.22	23.09	13.46	11.30	9.54	0.71	0.73	0.65	0.72	0.57	0.67
60	56.91	40.77	29.79	18.59	15.66	13.88	0.73	0.73	0.65	0.71	0.62	0.70
80	68.86	48.69	35.61	23.76	20.35	17.73	0.72	0.71	0.66	0.73	0.63	0.70

Sensitivity to parameters. We evaluated the sensitivity of `Fier_init` to the parameters affecting locality-sensitive hashing. Overall, we found it to be very robust.

We tested all combinations of values 20, 40, 60 and 80 for b_J and values 3, 5, 7, 10, 12 and 15 for r_J and evaluated the results both w.r.t. running time and quality of answers. For these experiments we used the EuroClim dataset.

Table 3 shows how the parameters impact the running time and accuracy. We can see that r_J has the largest impact, the running time increasing with smaller values of r_J. This is because r_J determines the length of the minhash signature, and shorter signatures imply a higher chance of matching and forming a candidate pair. On the other hand, the parameters do not have much impact on the average accuracy of the fifth-best result ($\bar{J}$@5).

As the average quality does not change by much, we can conclude that the user can set rows and bands primarily based on how many initial pairs they want to find, but that the settings we have used throughout these experiments ($r_J = 10$, $b_J = 40$) seem to give a good balance.

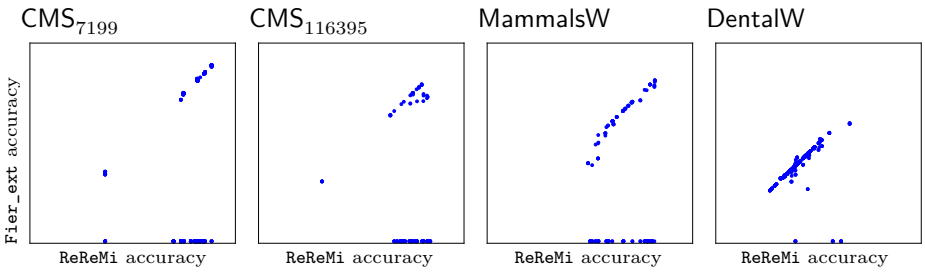


Fig. 4. Comparing the accuracy of once extended initial pairs by `Fier_ext` and `ReReMi`. Each dot represents a pair of columns, and its location indicates the highest-accuracy initial pair `Fier` and `ReReMi` found. All axes range across the unit interval.

3.3 Extending Initial Pairs

The next phase of the `Fier` algorithm is the extension of the initial pairs. To test this phase, we use the same pre-mined initial pairs for all algorithms and compare extensions obtained with `Fier_ext` versus with the exhaustive attempts of `ReReMi`. For the extensions, pre-bucketing as done by `ReReMiBkt` is not expected to bring any benefits for `ReReMi`, as its cut-point algorithm is very efficient (cf. Table 2). `Fier_ext` might extend some initial pairs with literals that are somewhat inferior to the ones found by `ReReMi`, or it might fail to find any extensions at all. On the other hand, it should be much faster than `ReReMi`. For these tests we only consider MammalsW, DentalW and two CMS datasets due to space reasons.

Extending once. In the first test, we check how similar the extensions are after one round of extensions. The results are presented in Fig. 4. This plot is similar to Fig. 3. As we can see, there are some initial pairs for which `Fier_ext` does not find any valid extension, but for those that it does find, the results are generally as good as those of `ReReMi`.

Extending multiple times. Genuine redescription mining involves extending the pairs more than just once. At this point it is no longer sensible to compare the accuracies of individual initial pairs, as small differences in each extension step can yield vastly different redescriptions, and the greedy extension done by `ReReMi` can sometimes yield worse results than `Fier_ext`. Instead, we consider the average Jaccard of the 10th best result ($\bar{J}$@10). We also measure the number of extension steps done by the algorithms *in total*. That is, if there are 100 initial pairs, and each pair is extended 3 times on both sides, the total number of extension steps is 600. Notice that `ReReMi` stops extensions when it cannot find any literal to extend with. The results of this experiment are presented in Table 4. Further results are presented in the technical report [18].

As can be seen in Table 4, the average $\bar{J}$@10 for `Fier_ext` is somewhat less than for `ReReMi`. With appropriate parameters, however, the difference is small. On the other hand, the running times can differ significantly. With DentalW for

Table 4. Accuracy, total number of extension steps, and time in seconds when extending the `ReReMi` initial pairs multiple times. All results are averages over 5 runs.

			MammalsW			DentalW		
algorithm	r_H	b_H	J@10	# ext.	time (s)	J@10	# ext.	time (s)
`Fier_ext`	10	10	0.77	296.4	1471.74	0.60	535.8	643.16
		20	0.79	329.0	1664.60	0.61	547.8	712.50
		40	0.79	354.8	1764.90	0.62	543.8	776.40
	20	10	0.72	112.6	768.05	0.57	500.2	390.47
		20	0.72	130.6	973.91	0.58	523.0	472.29
		40	0.75	237.4	1383.91	0.59	530.6	578.33
	30	10	0.68	60.0	438.41	0.54	402.8	245.93
		20	0.70	65.4	641.48	0.57	462.2	348.41
		40	0.71	101.0	990.65	0.57	512.8	483.04
	40	10	0.63	38.4	285.89	0.51	229.0	125.20
		20	0.68	54.6	473.24	0.53	329.0	221.54
		40	0.70	62.0	727.29	0.54	426.2	395.37
`ReReMi`			0.83	410.0	3263.28	0.61	517.0	1494.44
			CMS_{7199}			CMS_{116395}		
algorithm	r_H	b_H	J@10	# ext.	time (s)	J@10	# ext.	time (s)
`Fier_ext`	10	10	0.82	28.8	90.08	0.77	90.0	1072.10
		20	0.82	40.0	181.25	0.78	106.8	1906.59
		40	0.82	47.0	299.37	0.80	126.4	2646.13
	20	10	0.80	4.0	7.56	0.72	11.4	202.65
		20	0.81	6.4	17.68	0.73	20.4	303.56
		40	0.81	8.6	31.13	0.74	41.8	532.56
	30	10	0.80	0.6	6.82	0.71	5.0	148.69
		20	0.80	1.4	12.58	0.72	5.6	228.74
		40	0.81	3.0	23.88	0.72	8.4	421.66
	40	10	0.79	0.0	7.35	0.71	1.4	117.36
		20	0.79	0.6	13.86	0.71	2.8	223.92
		40	0.80	0.8	25.98	0.71	2.8	396.22
`ReReMi`			0.83	55.0	695.85	0.81	154.0	7626.95

Table 5. Accuracy, number of extension steps, number of resulting redescriptions and time in seconds for full redescription mining. All results are averages over 5 runs.

			MammalsW				DentalW			
alg.	r_H	b_H	J@10	# ext.	# results	time	J@10	# ext.	# results	time
`Fier_full`	20	20	0.74	94.6	32.4	765	0.57	486.8	216.8	445
		40	0.72	75.0	24.0	901	0.59	505.8	222.6	549
	30	20	0.63	39.0	14.2	446	0.52	423.8	189.0	315
		40	0.69	66.8	20.8	705	0.55	454.6	202.6	447
	40	20	0.63	28.0	10.6	333	0.51	258.8	139.6	199
		40	0.58	37.0	13.2	500	0.53	362.8	178.2	358
`ReReMi`			0.83	410.0	71.0	3401	0.61	517.0	93.0	3063
`ReReMiBkt`			0.83	410.0	71.0	3381	0.63	512.0	90.0	1866
			CMS_{7199}				CMS_{116395}			
alg.	r_H	b_H	J@10	# ext.	# results	time	J@10	# ext.	# results	time
`Fier_full`	20	20	0.75	12.2	18.8	34	0.76	19.6	21.4	463
		40	0.77	23.2	23.0	53	0.76	22.8	23.8	592
	30	20	0.74	3.6	17.0	30	0.75	4.2	17.2	381
		40	0.75	8.8	19.8	41	0.75	6.4	18.4	517
	40	20	0.73	3.0	17.6	31	0.73	0.6	16.0	367
		40	0.72	2.4	17.2	41	0.74	1.4	16.4	510
`ReReMi`			0.83	55.0	22.0	1032	0.81	154.0	45.0	96
`ReReMiBkt`			0.83	55.0	22.0	957	0.83	79.0	29.0	5796

instance, `ReReMi` takes almost 1500 seconds while `Fier_ext` achieves the same accuracy in half the time (using $r_H = 10, b_H = 20$), or comparable accuracy in less than a quarter of the time (using $r_H = 30, b_H = 20$). We can also see that for the extensions, smaller values of r_H yield higher running times. This is because with smaller r_H, LSH is more random and tends to generate much more potential extensions. On the other hand, higher values of b_H tend to increase the accuracy, as LSH then simply does more repetitions.

3.4 Building Full Redescriptions

The final test is for the full algorithm `Fier_full`, where we use LSH for both initial pairs and extensions. The results are presented in Table 5 with further results in the technical report [18].

For the CMS datasets, `Fier_full` cannot find many extensions. This situation is the same as in Table 4, indicating that it is probably a feature of the extension algorithm rather than of the initial pairs. On the other hand, the average accuracies are still quite high, indicating that the lack of extensions might be a consequence of having high-accuracy initial pairs that cannot be extended with higher accuracy. The running times of the algorithm are very low, showing a significant improvement over `ReReMi` and `ReReMiBkt`.

4 Conclusions

Locality-sensitive hashing in `Fier` significantly speeds up finding the initial pairs and extending them, without sacrificing the quality. Handling intervals from numerical variables requires a more complex approach, especially for the initial pairs, but it pays off with notably improved running times. Based on our experiments, we can recommend using `Fier` for the initialisation of greedy redescription mining without reservation. The extension phase is somewhat more sensitive to hyperparameters and is usually not as significant a bottleneck. For larger data sets or for quick or interactive testing, we can recommend `Fier_full`.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. In: STOC'98. pp. 327–336 (1998). https://doi.org/10.1145/276698.276781
2. Cochez, M., Mou, H.: Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time. In: SIGMOD'15. pp. 505–517 (2015). https://doi.org/10.1145/2723372.2751521
3. Cohen, E., Datar, M., Fujiwara, S., Gionis, A., Indyk, P., Motwani, R., Ullman, J.D., Yang, C.: Finding interesting associations without support pruning. IEEE Trans. Knowl. Data Eng. **13**(1), 64–78 (2001). https://doi.org/10.1109/69.908981
4. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: Scalable online collaborative filtering. In: WWW'07. pp. 271–280 (2007). https://doi.org/10.1145/1242572.1242610
5. Eronen, J.T., Polly, P.D., Fred, M., Damuth, J., Frank, D.C., Mosbrugger, V., Scheidegger, C., Stenseth, N.C., Fortelius, M.: Ecometrics: The traits that bind the past and present together. Integrative Zoology **5**(2), 88–101 (2010). https://doi.org/10.1111/j.1749-4877.2010.00192.x
6. Fernandes, N., Kawamoto, Y., Murakami, T.: Locality sensitive hashing with extended differential privacy. In: ESORICS'21. pp. 563–583 (2021). https://doi.org/10.1007/978-3-030-88428-4_28
7. Fortelius, M., Eronen, J., Jernvall, J., Liu, L., Pushkina, D., Rinne, J., Tesakov, A., Vislobokova, I., Zhang, Z., Zhou, L.: Fossil mammals resolve regional patterns of Eurasian climate change over 20 million years. Evol. Ecol. Res. **4**(7), 1005–1016 (2002)
8. Galbrun, E., Miettinen, P.: From black and white to full color: Extending redescription mining outside the Boolean world. Stat. Anal. Data Min. **5**(4), 284–303 (2012). https://doi.org/10.1002/sam.11145
9. Galbrun, E., Miettinen, P.: Analysing political opinions using redescription mining. In: ICDM'16 Workshops. pp. 422–427 (2016). https://doi.org/10.1109/ICDMW.2016.0066
10. Galbrun, E., Miettinen, P.: Mining Redescriptions with Siren. ACM Trans. Knowl. Discov. Data **12**(1), 6 (2018). https://doi.org/10.1145/3007212

11. Galbrun, E., Tang, H., Fortelius, M., Žliobaitė, I.: Computational biomes: The ecometrics of large mammal teeth. Palaeontol. Electron. (2018). https://doi.org/10.26879/786
12. Gallo, A., Miettinen, P., Mannila, H.: Finding subgroups having several descriptions: Algorithms for redescription mining. In: SDM'08. pp. 334–345 (2008). https://doi.org/10.1137/1.9781611972788.30
13. Helsingin Sanomat: Parliamentary elections 2011: Candidate responses to helsingin sanomat candidate selector (2016), `http://urn.fi/urn:nbn:fi:fsd:T-FSD2701`, version 2.1
14. Hijmans, R.J., Cameron, S.E., Parra, L.J., Jones, P.G., Jarvis, A.: Very High Resolution Interpolated Climate Surfaces for Global Land Areas. Int. J. Climatol. **25**, 1965–1978 (2005), `www.worldclim.org`
15. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC'98. pp. 604–613 (1998). https://doi.org/10.1145/276698.276876
16. Kalofolias, J., Galbrun, E., Miettinen, P.: From sets of good redescriptions to good sets of redescriptions. In: ICDM'16. pp. 211–220 (2016). https://doi.org/10.1109/ICDM.2016.0032
17. Karjalainen, M., Galbrun, E., Miettinen, P.: Serenade: An approach for differentially private greedy redescription mining. In: Proc. of the 20th anniv. Workshop on KDID at ECML-PKDD'22. pp. 31–46. CEUR Workshop Proceedings (2022)
18. Karjalainen, M., Galbrun, E., Miettinen, P.: Fast redescription mining using locality-sensitive hashing. arXiv 2406.04148 (2024). https://doi.org/10.48550/arXiv.2406.04148
19. Leskovec, J., Rajaraman, A., Ullman, J.: Mining of massive data sets. Cambridge university press (2020)
20. Liu, L., Galbrun, E., Tang, H., Kaakinen, A., Zhang, Z., Zhang, Z., Žliobaitė, I.: The emergence of modern zoogeographic regions in Asia examined through climate–dental trait association patterns. Nat. Commun. **14**(1) (2023). https://doi.org/10.1038/s41467-023-43807-w
21. Meeng, M., Knobbe, A.: For real: A thorough look at numeric attributes in subgroup discovery. Dat Min. Knowl. Discov. **35**(1), 158–212 (2021). https://doi.org/10.1007/s10618-020-00703-x
22. Mihelčić, M., Džeroski, S., Lavrač, N., Šmuc, T.: A framework for redescription set construction. Expert Syst. Appl. **68**, 196–215 (2017). https://doi.org/10.1016/j.eswa.2016.10.012
23. Mihelčić, M., Miettinen, P.: Differentially private tree-based redescription mining. Data Min. Knowl. Discov. **37**(4), 1548–1590 (2023). https://doi.org/10.1007/S10618-023-00934-8
24. Mihelčić, M., Šimić, G., Babić-Leko, M., Lavrač, N., Džeroski, S., Šmuc, T.: Using redescription mining to relate clinical and biological characteristics of cognitively impaired and alzheimer's disease patients. PLOS ONE **12**(10) (2017). https://doi.org/10.1371/journal.pone.0187364
25. Murdock, G.P.: Ethnographc atlas: A summary. Ethnology **6**(2), 109–236 (1967)
26. Ramakrishnan, N., Kumar, D., Mishra, B., Potts, M., Helm, R.F.: Turning CARTwheels: An alternating algorithm for mining redescriptions. In: KDD'04. pp. 266–275 (2004). https://doi.org/10.1145/1014052.1014083
27. Zinchenko, T., Galbrun, E., Miettinen, P.: Mining Predictive Redescriptions with Trees. In: ICDM'15 Workshops. pp. 1672–1675 (2015). https://doi.org/10.1109/ICDMW.2015.123