algorithms

MDPI

Article

VLSD—An Efficient Subgroup Discovery Algorithm Based on Equivalence Classes and Optimistic Estimate

Antonio Lopez-Martinez-Carrasco [1,2,*,†], Jose M. Juarez [1,2,†], Manuel Campos [1,2,3,†] and Bernardo Canovas-Segura [1,2,*,†]

[1] MedAI-Lab, University of Murcia, 30100 Murcia, Spain; jmjuarez@um.es (J.M.J.); manuelcampos@um.es (M.C.)
[2] Facultad de Informatica, Campus de Espinardo, Universidad de Murcia, 30100 Murcia, Spain
[3] Murcian Bio-Health Institute (IMIB-Arrixaca), 30120 Murcia, Spain
* Correspondence: antoniolopezmc@um.es (A.L.-M.-C.); bernardocs@um.es (B.C.-S.)
† These authors contributed equally to this work.

Abstract: Subgroup Discovery (SD) is a supervised data mining technique for identifying a set of relations (subgroups) among attributes from a dataset with respect to a target attribute. Two key components of this technique are (i) the metric used to quantify a subgroup extracted, called quality measure, and (ii) the search strategy used, which determines how the search space is explored and how the subgroups are obtained. The proposal made in this work consists of two parts, (1) a new and efficient SD algorithm which is based on the equivalence class exploration strategy, and which uses a pruning based on optimistic estimate, and (2) a data structure used when implementing the algorithm in order to compute subgroup refinements easily and efficiently. One of the most important advantages of this algorithm is its easy parallelization. We have tested the performance of our SD algorithm with respect to some other well-known state-of-the-art SD algorithms in terms of runtime, max memory usage, subgroups selected, and nodes visited. This was completed using a collection of standard, well-known, and popular datasets obtained from the relevant literature. The results confirmed that our algorithm is more efficient than the other algorithms considered.

Keywords: data mining; subgroup discovery; algorithm; equivalence classes; optimistic estimate

Citation: Lopez-Martinez-Carrasco, A.; Juarez, J.M.; Campos, M.; Canovas-Segura, B. VLSD—An Efficient Subgroup Discovery Algorithm Based on Equivalence Classes and Optimistic Estimate. *Algorithms* **2023**, *16*, 274. https://doi.org/10.3390/a16060274

Academic Editor: Frank Werner

Received: 26 April 2023
Revised: 18 May 2023
Accepted: 25 May 2023
Published: 29 May 2023

1. Introduction

Subgroup Discovery [1] (SD) is a supervised data mining technique that is widely used for descriptive and exploratory data analysis. Its purpose is to identify a set of relations between attributes from a dataset with respect to a target attribute of interest. SD is useful as regards automatically generating hypotheses, obtaining general relations in the data and carrying out data analysis and exploration. When executing an SD algorithm, the relations obtained are denominated as *subgroups*. The SD technique has made it possible to obtain remarkable results in both medical and technical fields [2–4].

Assessing the quality of a subgroup extracted by an SD algorithm is a key aspect of this technique. There is a wide variety of metrics for this purpose, and these are denominated as *quality measures*. A quality measure is, in general, a function that assigns one numeric value to a subgroup according to certain specific properties [5], and selecting a good quality measure for each specific problem is, therefore, essential. The quality measures are divided into two groups, (1) quality measures designed to be applied when the target attribute is of a nominal type, and (2) quality measures designed to be applied when the target attribute is of a numeric type. Some examples of quality measures are Sensitivity, Specificity, Weighted Relative Accuracy (WRAcc), or Information Gain, among others [6]. Some popular quality measures, such as those previously enumerated, could be adapted to be applied to both attributes of the nominal type or attributes of the numeric type.

In addition to the quality measure, the other important aspect in this technique is the search strategy used by the SD algorithm. This strategy determines how the search space of the problem is explored and how the subgroups are obtained from it.

A preliminary six-page version of this work appeared in [7], in which we briefly presented an initial and very simple version of our SD algorithm. This work continues and extends that preliminary version on the basis of valuable reviewers' feedback. This manuscript incorporates a more detailed theoretical framework, an extended and improved SD algorithm (to which several modifications have also been made), detailed explanations, and extended experiments using a variety of well-known datasets. The main contributions of this paper are, therefore (1) the new and efficient SD algorithm, which is based on the equivalence class exploration strategy and uses a pruning based on optimistic estimate, and (2) the extended and improved data structure used to implement that algorithm.

It is essential to remark that, although the ideas contained in this paper were already presented in previous works separately, this is the first time that they are used, implemented and validated together.

The remainder of this paper is structured as follows. Section 2 provides a background to the SD technique and to some existing SD algorithms, and introduces related work, while Section 3 describes the formal aspects of the SD technique. Section 4 shows and explains our proposal, the VLSD algorithm and vertical list data structure. Section 5 describes the configuration of the experiments carried out in order to compare our proposal with other existing SD algorithms, the results obtained after this comparison process and a discussion of those results. Finally, Section 6 provides the conclusions reached after carrying out this research.

2. Related Work

Before introducing the state-of-the-art of the Subgroup Discovery (SD) technique, it is important to highlight the differences between this technique and others, such as clustering, pattern mining, or classification. In the first place, clustering and pattern mining algorithms are unsupervised and do not use an output attribute or class, while SD algorithms are supervised and generate relations (called subgroups) with respect to a target attribute. In the second place, classification algorithms generate a global model for the whole population with the aim of predicting the outcome of a new observation, while SD algorithms create local descriptive models with subpopulations that are statistically significant with respect to the whole population in relation to the target attribute. Moreover, the populations covered by different subgroups may overlap, while this is not the case in a classification model.

SD algorithms have several characteristics, which make them different from each other and which need to be considered depending on the problem and on the input data to be analyzed. It is possible to highlight (1) the exploration strategy carried out by the SD algorithm in the search space of the problem (exhaustive versus heuristic); (2) the number of subgroups that the SD algorithm returns (all subgroups explored versus top-k subgroups); (3) whether the SD algorithm carries out additional pruning in order to avoid the need to explore regions of the search space that have less quality (e.g., pruning based on optimistic estimate); and (4) the data structure that the SD algorithm employs (e.g., FPTree, TID List or Bitset).

Exhaustive SD algorithms are those that explore the complete search space of the problem, while heuristic SD algorithms are those that use a heuristic function in order to guide the exploration of the search space of the problem. Exhaustive algorithms guarantee that best subgroups are found; however, if the search space of the problem is too large, the application of these algorithms is not feasible. The alternative is heuristic algorithms, which are more efficient and make it possible to reduce the potential number of subgroups that must be explored. However, these algorithms do not guarantee that the best subgroups will be found [1,8].

When executing an SD algorithm (either exhaustive or heuristic) with a quality measure and a certain quality threshold that is established a priori, it can return either all

the subgroups explored or only the best k subgroups (i.e., the top-k). The main advantage of the top-k strategy is that it reduces the memory consumption of the SD algorithm because it is not necessary to store all the subgroups explored [1].

Many SD algorithms also implement additional pruning that improve the efficiency and avoid the need to explore certain regions of the search space that have less quality. One of them is the pruning based on optimistic estimate. An optimistic estimate is a quality measure that, for a certain subgroup, provides a quality upper bound for all its refinements [9]. This upper bound is a value that cannot be reached by any subgroup refinement. Therefore, if this value is less than the established quality threshold, then it means that not suitable subgroup can be generated by refining the current one, hence it can be dropped. This pruning avoids the need to explore complete regions of the search space that have less quality than the quality threshold established, after analyzing only one subgroup.

One disadvantage of the SD technique is the huge number of subgroups that could be generated (i.e., pattern explosion), and it is especially relevant when using input datasets with too many attributes. For this reason, the utilization of an optimistic estimate provides a solution of this problem when the quality measure threshold established allows not to explore a large part of the search space.

It is essential to remark that standard quality measures for SD, such as Sensitivity, Specificity, WRAcc, or Information Gain, are neither optimistic estimates nor monotonic. This means that, when using these standard quality measures, the refinements of a subgroup could have a higher-quality measure than its father, so it is necessary to explore the complete search space. However, optimistic estimate quality measures are monotonic by definition and can, therefore, be used for pruning and to reduce the search space of a problem, because if a certain subgroup is not of sufficient quality to be considered when using this optimistic estimate, it is certain that none of its refinements will be of that quality either [9].

It is a common practice for SD algorithms to be based on other non-SD algorithms. Many existing SD algorithms are adaptations of, for instance, classification algorithms or frequent pattern mining algorithms, among others. In these cases, their data structures and algorithmic schemes are modified with the objective of obtaining subgroups.

The following are examples of SD algorithms based on existing classification algorithms: EXPLORA [10], MIDOS [11], PRIM [12], SubgroupMiner [13], RSD [14], CN2-SD [15] or SD [16], among others. The following are examples of SD algorithms based on existing frequent pattern mining algorithms, Apriori-SD [17], DpSubgroups [9], SD4TS [18], or SD-Map* [19], among others.

In addition to the above, SD-Map [20] and BSD [21] algorithms (based on existing frequent pattern mining algorithms) are explained, since they are two representative examples of exhaustive SD algorithms.

SD-Map is an exhaustive SD algorithm based on the well-known FP-Growth [22] algorithm for frequent pattern mining. This algorithm uses the FPTree data structure in order to represent the complete dataset and to mine subgroups in two steps; a complete FPTree is first built from the input dataset, after which successive conditional FPTrees are built recursively in order to mine the subgroups.

BSD is an exhaustive SD algorithm that uses the Bitset data structure and the depth-first search approach. Each subgroup has an associated Bitset data structure that stores the instances covered and not covered by that subgroup through the use of bits. This data structure has several advantages, including (1) reduced memory consumption, since it uses bitset-based representation for the coverage information; (2) subgroup refinements efficiently obtained by using logical AND operations; and (3) highly time and memory efficient implementation in most programming languages. This data structure is used to mine subgroups in two steps; the Bitset data structure for each single selector involved in the subgroup discovery process is first constructed, after which all possible refinements are constructed recursively.

It is sometimes possible that two subgroups generated by a specific SD algorithm are redundant, because they represent and explain the same portion of data from a specific dataset. If these subgroups are redundant, one of them is dominant and the other is dominated in terms of their coverage. The dominated subgroup can, therefore, be removed. In this respect, it is possible to highlight two dominance relations, closed [23] and closed-on-the-positives [21]. Two subgroups have a closed dominance relation if the instances covered by both subgroup descriptions (no matter what the target value is) are the same. In this case, the most specific subgroup is dominant and the most general subgroup is dominated. Two subgroups have a closed-on-the-positives dominance relation if the positive instances (i.e., the instances in which the target is positive) covered by both subgroup descriptions are the same. In this case, the most general subgroup is dominant and the most specific subgroup is dominated.

The algorithms mentioned above can also be modified and adapted in order to only obtain closed subgroups or to only obtain closed-on-the-positives subgroups.

Apart from the exploration strategies indicated above, the equivalence class strategy has also been used for frequent pattern mining. This strategy was proposed by Zaki et al. [24], and there is, to the best of our knowledge, no SD algorithm that uses it.

With regard to pattern mining, other approaches with similar objectives can be mentioned. Utility pattern mining is a technique that is widely used and discussed in the literature and which consists of discovering patterns that have a high relevance in terms of a numeric utility function. This function does not simply measure the quality or the importance of a pattern in relation to a specific dataset, but can also consider other additional criteria of that pattern beyond the database itself [25,26]. Note that, while these algorithms use other types of upper bound measures that may not be monotonic in order to reduce the search space that must be explored, we use optimistic estimate quality measures that are monotonic by definition. Furthermore, other approaches with which to mine patterns in those cases in which the amount of data is limited have recently been presented. For example, the authors of [27] show an algorithm that can be used to mine colossal patterns, i.e., patterns extracted from databases with many attributes and values, but with few instances.

Finally, for a general review of the SD technique, we refer the reader to [1,8].

3. Problem Definition

The fundamental concepts of the Subgroup Discovery (SD) technique are provided in this section. Additionally, these concepts are extended and detailed in Appendix A.

First, an attribute a is a unique characteristic of an object, which has an associated value. An example of an attribute is a = *age*:30. Therefore, the domain of an attribute a (denoted as $dom(a)$) can be defined as the set of all the unique values that said attribute can take. An attribute can be nominal or numeric, depending on its domain. On the other hand, an instance i is a tuple $i = (a_1, \ldots, a_M)$ of attributes. Given the attributes a_1 = *age*:25 and a_2 = *sex*:*woman*, an example of an instance is i = (*age*:25, *sex*:*woman*). Additionally, a dataset d is a tuple $d = (i_1, \ldots, i_N)$ of instances. Given the instances i_1 = (*age*:30, *sex*:*man*) and i_2 = (*age*:25, *sex*:*woman*), an example of a dataset is d = ((*age*:30, *sex*:*man*), (*age*:25, *sex*:*woman*)).

It is necessary to state that all values from a dataset d can be indexed with two integers, x and y. We use the notation $v_{x,y}$ to indicate the value of the x-th instance i_x and of the y-th attribute a_y from a dataset d.

Given an attribute a_y from a dataset d, a binary $operator \in \{=, \neq, <, >, \leq, \geq\}$ and a value $w \in dom(a_y)$, a selector e is a 3-tuple of the form $(a_y.characteristic, operator, w)$. Note that when an attribute a_y is nominal, only the $=$ and $\neq$ operators are permitted. Informally, a selector is a binary relation between an attribute from a dataset and a value in the domain of that attribute. This relation represents a property of a subset of instances from that dataset.

It is essential to bear in mind that the first element of a selector refers only to the attribute name, i.e., the characteristic, and not to the complete attribute itself.

Definition 1 (Selector covering)**.** *Given an instance i_x and an attribute a_y from a dataset d, and a selector $e = (a_y.characteristic, operator, w \in dom(a_y))$, then i_x is covered by e (denoted as $i_x \sqsubset e$) if the binary expression "$v_{x,y}$ operator w" holds true. Otherwise, we say that it is not covered by e (denoted as $i_x \not\sqsubset e$).*

For example, given the instance i_x = (*age*:25, *sex*:*woman*) and the selectors $e_1 = (age, <, 20)$ and $e_2 = (sex, =, woman)$, it will be noted that $i_x \not\sqsubset e_1$ and $i_x \sqsubset e_2$.

Subsequently, a pattern p is a list of selectors $< e_1, \ldots, e_j >$ in which all attributes of the selectors are different. Moreover, its size (denoted as $|p|$) is defined as the number of selectors that it contains. In general, a pattern is interpreted as a list of selectors (i.e., as a conjunction) that represents a list of properties of a subset of instances from a dataset.

Definition 2 (Pattern covering)**.** *Given an instance i_x from a dataset d and a pattern p, then i_x is covered by p (denoted as $i_x \sqsubset p$) if $\forall e \in p, i_x \sqsubset e$. Otherwise, we say that it is not covered by p (denoted as $i_x \not\sqsubset p$).*

Following these definitions, a subgroup s is a pair (pattern, selector) in which the pattern is denoted as *s.description* and the selector is denoted as *s.target*. Given the dataset d = ((*fever*:*yes*, *sex*:*man*, *flu*:*yes*), (*fever*:*yes*, *sex*:*woman*, *flu*:*no*)), an example of a subgroup is $s = (< (fever, =, yes), (sex, =, woman) >, (flu, =, yes))$.

Definition 3 (Subgroup refinement s')**.** *Given a subgroup s, each of its refinements s' (denoted as $s \prec s'$) is a subgroup with the same target, $s'.target = s.target$, and with an extended description, $s'.description = concat(s.description, < e_1, \ldots, e_j >)$.*

Definition 4 (Refine operator)**.** *Given two subgroups, s_x and s_y, the refine operator generates a refinement $s_{x,y}$ of s_x, extending its description with the non-common suffix of s_y. For example, if $s_x.description$ = <e_1> and $s_y.description$ = <e_2>, then $s_{x,y}.description$ = <e_1, e_2>; and if $s_x.description$ = <e_1, e_2, e_3> and $s_y.description$ = <e_1, e_2, e_4>, then $s_{x,y}.description$ = <e_1, e_2, e_3, e_4>.*

Given a subgroup s and a dataset d, a quality measure q is a function that computes one numeric value according to that subgroup s and to certain characteristics from that dataset d [5]. Moreover, given a quality measure q and a dataset d, an optimistic estimate oe of q is a quality measure that, for a certain subgroup, provides a quality upper bound for all its refinements [9].

Focusing on a specific subgroup s and on a specific dataset d, different functions with which to compute quality measures can be defined.

The function *tp* (true positives) is defined as the number of instances i_x from the dataset d that are covered by the subgroup description *s.description* and by the subgroup target *s.target*. Formally:

$$tp(s, d) = |\{i_x \in d : i_x \sqsubset s.description \wedge i_x \sqsubset s.target\}| \quad (1)$$

The function fp (false positives) is defined as the number of instances i_x from the dataset d that are covered by the subgroup description *s.description*, but not by the subgroup target *s.target*. Formally:

$$fp(s, d) = |\{i_x \in d : i_x \sqsubset s.description \wedge i_x \not\sqsubset s.target\}| \quad (2)$$

The function TP (true population) is defined as the number of instances i_x from the dataset d that are covered by the subgroup target *s.target*. Formally:

$$TP(s, d) = |\{i_x \in d : i_x \sqsubset s.target\}| \quad (3)$$

The function FP (false population) is defined as the number of instances i_x from the dataset d that are not covered by the subgroup target $s.target$. Formally:

$$FP(s,d) = |\{i_x \in d : i_x \not\sqsubset s.target\}| \tag{4}$$

A quality measure q can, therefore, be redefined using the previous four functions, given a subgroup s and a dataset d, a quality measure q is a function that computes one numeric value according to the functions tp, fp, TP, and FP.

They are sufficiently expressive to compute any quality measure. However, the following are also used in the literature.

The function n is defined as the number of instances i_x from a dataset d that are covered by the subgroup description $s.description$. Formally:

$$n(s,d) = |\{i_x \in d : i_x \sqsubset s.description\}| \tag{5}$$

The function N is defined as the number of instances i_x from the dataset d. Formally:

$$N(s,d) = |\{i_x \in d : i_x\}| \tag{6}$$

The function p is defined as the distribution of the subgroup target $s.target$ with respect to the instances i_x from a dataset d covered by the subgroup description $s.description$. Formally:

$$p(s,d) = \frac{|\{i_x \in d : i_x \sqsubset s.description \wedge i_x \sqsubset s.target\}|}{|\{i_x \in d : i_x \sqsubset s.description\}|} \tag{7}$$

The function p_0 is defined as the distribution of the subgroup target $s.target$ with respect to all instances i_x from a dataset d. Formally:

$$p_0(s,d) = \frac{|\{i_x \in d : i_x \sqsubset s.target\}|}{|\{i_x \in d : i_x\}|} \tag{8}$$

The function tn (true negatives) is defined as the number of instances i_x from the dataset d that are covered by neither the subgroup description $s.description$ nor the subgroup target $s.target$. Formally:

$$tn(s,d) = |\{i_x \in d : i_x \not\sqsubset s.description \wedge i_x \not\sqsubset s.target\}| \tag{9}$$

The function fn (false negatives) is defined as the number of instances i_x from the dataset d that are not covered by the subgroup description $s.description$, but are covered by the subgroup target $s.target$. Formally:

$$fn(s,d) = |\{i_x \in d : i_x \not\sqsubset s.description \wedge i_x \sqsubset s.target\}| \tag{10}$$

Table 1 shows the confusion matrix of a subgroup s with respect to a dataset d. This matrix summarizes the functions describe above.

Table 1. Confusion matrix of a subgroup s with respect to a dataset d.

		s.target		
		True	**False**	
s.description	**True**	tp	fp	n = tp + fp
	False	fn = TP − tp	tn = FP − fp	TP + FP − tp − fp
		TP = tp + fn	FP = fp + tn	N = TP + FP

With regard to the functions defined previously, the following equivalences can be highlighted:

$$p(s,d) = \frac{tp(s,d)}{tp(s,d) + fp(s,d)} = \frac{tp(s,d)}{n(s,d)} \tag{11}$$

$$p_0(s,d) = \frac{TP(s,d)}{TP(s,d) + FP(s,d)} = \frac{TP(s,d)}{N(s,d)} \tag{12}$$

Having described the four functions with which to compute quality measures, some popular quality measures for SD presented in the literature can be rewritten as follows:

$$Sensitivity = \frac{tp}{TP} \tag{13}$$

$$Specificity = \frac{FP - fp}{FP} \tag{14}$$

$$Piatetsky\ Shapiro = (tp + fp) \cdot (\frac{tp}{tp + fp} - \frac{TP}{TP + FP}) \tag{15}$$

$$WRAcc = \frac{tp + fp}{TP + FP} \cdot (\frac{tp}{tp + fp} - \frac{TP}{TP + FP}) \tag{16}$$

The WRAcc quality measure is defined between −1 and 1 (both included). Moreover, an optimistic estimate of this quality measure [9] can be rewritten as follows:

$$WRAcc\ optimistic\ estimate = \frac{tp^2}{tp + fp} \cdot (1 - \frac{TP}{TP + FP}) \tag{17}$$

Note that, in this case, the parameters of the functions have not been shown for the sake of brevity and for reasons of space.

It is essential to keep in mind from the beginning that, although only WRAcc quality measure and its optimistic estimate are used in this research, they are only an example and, therefore, all quality measures which have an optimistic estimate could be used.

Finally, given a dataset d, a quality measure q and a numeric value $quality\_threshold$, the subgroup discovery problem consists of exploring the search space of d in order to enumerate the subgroups that have a quality measure value above the selected threshold. Formally

$$\mathcal{R} = \{(s, q(s,d)) | q(s,d) \geq quality\_threshold\} \tag{18}$$

The search space of a problem (i.e., of a dataset d) can be visually illustrated as a lattice [24] (see Figure 1). According to this comparison, the first level of the search space of a problem contains all those subgroups s whose descriptions have a size of 1 (i.e., $|s.description|$ is equal to one), the second level of the search space of a problem contains all those subgroups s whose descriptions have a size of 2 (i.e., $|s.description|$ is equal to two) and, in general, the level n of the search space of a problem contains all those subgroups s whose descriptions have the size n (i.e., $|s.description|$ is equal to n).

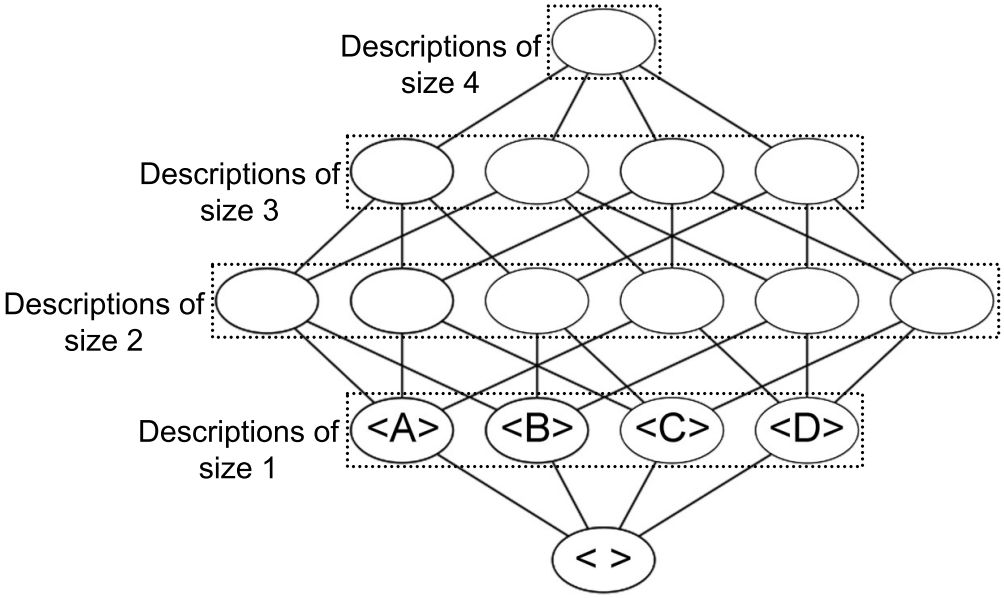

Figure 1. Search space of a problem visually illustrated as a lattice.

4. Algorithm

We propose a new and efficient Subgroup Discovery (SD) algorithm called VLSD (Vertical List Subgroup Discovery) that combines an equivalence class exploration strategy [24] and a pruning strategy based on optimistic estimate [9]. The implementation of this proposal is based on vertical list data structure, making it easily parallelizable [24].

The pruning based on optimistic estimate implies that, for all the nodes generated (i.e., subgroups), their optimistic estimate values are computed and compared with the threshold in order to discover whether they must be pruned (and it is, therefore, not necessary to explore their refinements), or whether their refinements (i.e., the next depth level) must also be explored.

Our proposal is described in the VLSD function (Algorithm 1) that is accompanied by a function that generates all subgroups whose descriptions have size one (GENERATE_SUBGROUPS_S1 function, described in Algorithm 2) and by a function that explores the search space and computes pruning (SEARCH function, described in Algorithm 3).

Algorithm 1 VLSD function.

Input: d { dataset }, *target* { selector }, q { quality measure }, *q_threshold* { $\mathbb{R}$ }, *oe* { optimistic estimate of q }, *oe_threshold* { $\mathbb{R}$ }, *sort_criterion_in_S*1 { criterion }, *sort_criterion_in_other_sizes* { criterion }

Output: $\mathcal{F}$: list of subgroups.

1: $\mathcal{F}$:=<>
2: $TP := TP((<>, target), d)$; $FP := FP((<>, target), d)$
3: $\mathcal{S}1$:= GENERATE_SUBGROUPS_S1(d, *target*, *oe*, *oe_threshold*, *sort_criterion_in_S*1, TP, FP)
4: **for each** subgroup $s \in \mathcal{S}1$ **do**
5: $q\_value := q(tp(s,d), fp(s,d), TP, FP)$
6: **if** $q\_value \geq q\_threshold$ **then**
7: $\mathcal{F}.add(s)$
8: **end if**
9: **end for**
10: $\mathcal{M}$:= 2-dimensional $|\mathcal{S}1| \times |\mathcal{S}1|$ triangular matrix, initialized $\mathcal{M}[i, j] = NULL$, in which $\mathcal{M}[i, j]$ is a subgroup (i and j selectors acting as indices).

Algorithm 1 *Cont.*

for each s_x, s_y in $\mathcal{S}1 =< s_1, s_2, \ldots, s_n >$, being $x < y$ **do**
  $s_{xy} := refine(s_x, s_y)$
  $oe\_quality := oe(tp(s_{xy}, d), fp(s_{xy}, d), TP, FP)$
  if $(tp(s_{xy}, d) + fp(s_{xy}, d) > 0)$ AND $(oe\_quality \geq oe\_threshold)$ **then**
    $\mathcal{M}[last(s_x.description)][last(s_y.description)] := s_{xy}$
  end if
end for
if $|\mathcal{S}1| \geq 2$ **then**
  for $i := 0$ **to** $(|\mathcal{S}1| - 2)$ **do**
    $selector\_i := last(\mathcal{S}1[i].description)$
    $\mathcal{P} := \mathcal{M}[selector\_i]$ { All subgroups whose descriptions have the size two and start with *selector_i* }
    $\mathcal{P} := \mathcal{P}.sort(sort\_criterion\_in\_other\_sizes)$
    for each subgroup $s \in \mathcal{P}$ **do**
      $q\_value := q(tp(s, d), fp(s, d), TP, FP)$
      if $q\_value \geq q\_threshold$ **then**
        $\mathcal{F}.add(s)$
      end if
    end for
    $\mathcal{F}.add\_all(SEARCH(d, \mathcal{P}, \mathcal{M}, q, q\_threshold, oe, oe\_threshold, sort\_criterion\_in\_other\_sizes, TP, FP))$
  end for
end if
return $\mathcal{F}$

Algorithm 2 `GENERATE_SUBGROUPS_S1` function.

Input: *d* { dataset }, *target* { selector }, *oe* { optimistic estimate }, *oe_threshold* { $\mathbb{R}$ }, *sort_criterion_in_S1* { criterion }, *TP* { $\mathbb{N}$ }, *FP* { $\mathbb{N}$ }
Output: $\mathcal{S}1$: list of subgroups whose descriptions have the size one.
$\mathcal{S}1 :=<>$
$\mathcal{E} :=$ scan *d* (except the target attribute) to generate the selector list.
for each selector $e \in \mathcal{E}$ **do**
  $s := (< e >, target)$
  $oe\_quality := oe(tp(s, d), fp(s, d), TP, FP)$
  if $oe\_quality \geq oe\_threshold$ **then**
    $\mathcal{S}1.add(s)$
  end if
end for
$\mathcal{S}1 := sort(\mathcal{S}1, sort\_criterion\_in\_S1)$
return $\mathcal{S}1$

Algorithm 3 `SEARCH` function.

Input: *d* { dataset }, $\mathcal{P}$ { subgroup list }, $\mathcal{M}$ { matrix }, *q* { quality measure }, *q_threshold* { $\mathbb{R}$ }, *oe* { optimistic estimate of *q* }, *oe_threshold* { $\mathbb{R}$ }, *sort_criterion_in_other_sizes* { criterion }, *TP* { $\mathbb{N}$ }, *FP* { $\mathbb{N}$ }
Output: $\mathcal{F}$: list of subgroups.
$\mathcal{F} :=<>$
while $|\mathcal{P}| > 1$ **do**
  $s_x := pop\_first(\mathcal{P})$
  $\mathcal{L} :=<>$ { List of subgroups }
  for each subgroup $s_y \in \mathcal{P}$ **do**
    $s_{\mathcal{M}} := get(\mathcal{M}, last(s_x.description), last(s_y.description))$
    $oe\_quality := oe(tp(s_{\mathcal{M}}, d), fp(s_{\mathcal{M}}, d), TP, FP)$

Algorithm 3 *Cont.*

```
8:          if (s_M ≠ NULL) AND (oe_quality ≥ oe_threshold) then
9:              s_xy := refine(s_x, s_y)
10:             oe_quality := oe(tp(s_xy, d), fp(s_xy, d), TP, FP)
11:             if (tp(s_xy, d) + fp(s_xy, d) > 0) AND (oe_quality ≥ oe_threshold) then
12:                 L.add(s_xy)
13:                 q_value := q(tp(s_xy, d), fp(s_xy, d), TP, FP)
14:                 if q_value ≥ q_threshold then
15:                     F.add(s_xy)
16:                 end if
17:             end if
18:         end if
19:     end for
20:     if L ≠<> then
21:         L := sort(L, sort_criterion_in_other_sizes)
22:         F.add_all(SEARCH(d, L, M, q, q_threshold, oe, oe_threshold,
        sort_criterion_in_other_sizes, TP, FP))
23:     end if
24: end while
25: return F
```

The VLSD function (Algorithm 1) requires the following parameters: a dataset d, a target attribute (a selector) *target*, a quality measure q, a threshold *q_threshold* for that quality measure, an optimistic estimate *oe* of q, a threshold *oe_threshold* for that optimistic estimate, a sorting criterion used to sort those subgroups whose descriptions have a size of 1, and a sorting criterion used to sort those subgroups whose descriptions have other sizes greater than 1. These criteria could be, for instance, by ascending quality measure value, by descending quality measure value, by description size ascending, no reorder, etc. Finally, this function returns a list $\mathcal{F}$ of subgroups.

The VLSD function is a constructive function that starts with the creation of the empty list $\mathcal{F}$ (in which the subgroups will be stored) and with the computation of the true population *TP* and the false population *FP* (lines 1–2). All those subgroups whose descriptions have a size of 1 (see Figure 1) are then generated, evaluated and added to the list $\mathcal{F}$ (lines 3–9). Note that these subgroups have already been sorted by a given criterion. A triangular matrix $\mathcal{M}$ is subsequently created and initialized (lines 10–17). This triangular matrix contains only those subgroups whose descriptions have a size of 2 (see Figure 1). The indices of this matrix are selectors, and for two indices, i and j, $\mathcal{M}[i][j]$ contains the subgroup whose descriptions have two such selectors (or NULL if that subgroup has been pruned). Moreover, the notation $\mathcal{M}[i]$ can be used to refer to all those subgroups whose descriptions have a size of 2 and start with the selector i. Finally, for each selector *selector_i* (lines 19–20), those subgroups from $\mathcal{M}$ whose descriptions have a size of 2 and start with that selector are obtained, evaluated, added to the list $\mathcal{F}$ and explored recursively (lines 18–31).

The utilization of matrix $\mathcal{M}$ makes the algorithm very efficient, because storing those subgroups whose descriptions have a size of 2, makes it possible to prune the rest of the search space with a higher cardinality quickly and easily (i.e., refinements of those subgroups whose descriptions have a size of 2) [28].

The `GENERATE_SUBGROUPS_S1` function (Algorithm 2) requires the following parameters, a dataset d, a target attribute (a selector) *target*, an optimistic estimate *oe*, a threshold *oe_threshold* for that optimistic estimate, a sort criterion used to sort those subgroups whose descriptions have a size of 1, and the *TP* and *FP* from the dataset d (these are passed by parameters in order to avoid the need to compute multiple times). Finally, this function returns a list $\mathcal{S}1$ of those subgroups whose descriptions have a size of one.

The `GENERATE_SUBGROUPS_S1` function starts with the creation of an empty list $\mathcal{S}1$ in which the subgroups will be stored (line 1). A selector list $\mathcal{E}$ is then generated from the

dataset d (line 2), and for each selector of that list, a subgroup is created, evaluated, and added to $\mathcal{S}1$ (lines 3–9). Finally, the subgroup list $\mathcal{S}1$ is sorted (line 10).

The SEARCH function (Algorithm 3) requires the following parameters, a dataset d, a subgroup list $\mathcal{P}$, a triangular matrix $\mathcal{M}$, a quality measure q, a threshold *q_threshold* for that quality measure, an optimistic estimate *oe* of q, a threshold *oe_threshold* for that optimistic estimate, a sort criterion used to sort those subgroups whose descriptions have other sizes greater than 1, and the TP and FP from the dataset d (these are passed by parameters in order to avoid the need to compute multiple times). Finally, this function returns a list $\mathcal{F}$ of subgroups.

The SEARCH function starts with the creation of an empty list $\mathcal{F}$ in which the subgroups will be stored (line 1). A double iteration through the subgroup list $\mathcal{P}$ is then carried out (loops of the lines 2 and 5). New subgroup refinements are subsequently generated and added to the subgroup list $\mathcal{L}$ (lines 9–18). Moreover, these subgroups are also evaluated and added to the list $\mathcal{F}$ (lines 13–16). It is important to highlight that matrix $\mathcal{M}$ is used in order to avoid the unnecessary generation of subgroup refinements, i.e., pruning the search space (lines 6–8). This is one of the key points of the efficiency of this algorithm [28]. Finally, the subgroup list $\mathcal{L}$ is sorted and the function is called recursively (lines 20–23).

Since the matrix $\mathcal{M}$ is a triangular matrix, indexing must be performed properly. This is taken into account in line 6.

The second part of our proposal is the vertical list data structure. This data structure is used in the algorithm implementation in order to compute the subgroup refinements easily and efficiently by making list concatenations and set intersections. Moreover, it stores all the elements required, with the objective of avoiding multiple and unnecessary recalculations.

Given a dataset d and a subgroup s, a vertical list vl is formed of the following elements:

1. The subgroup description (denoted as $vl.description$).
2. The set of IDs of the instances counted in $fp(s,d)$ (denoted as $vl.set\_fp$).
3. The set of IDs of the instances counted in $tp(s,d)$ (denoted as $vl.set\_tp$).

Note that $|vl.set\_fp|$ is equal to $fp(s,d)$ and $|vl.set\_tp|$ is equal to $tp(s,d)$.

An example of a vertical list data structure and an adapted $refine$ operator for it is depicted in Figure 2. In this case, the $refine$ operator is not applied over subgroups, but over vertical lists. First, this operator is applied over $vl1$ and $vl2$ in order to generate $vl3$ and, next, this operator is applied over $vl3$ and $vl4$ in order to obtain $vl5$.

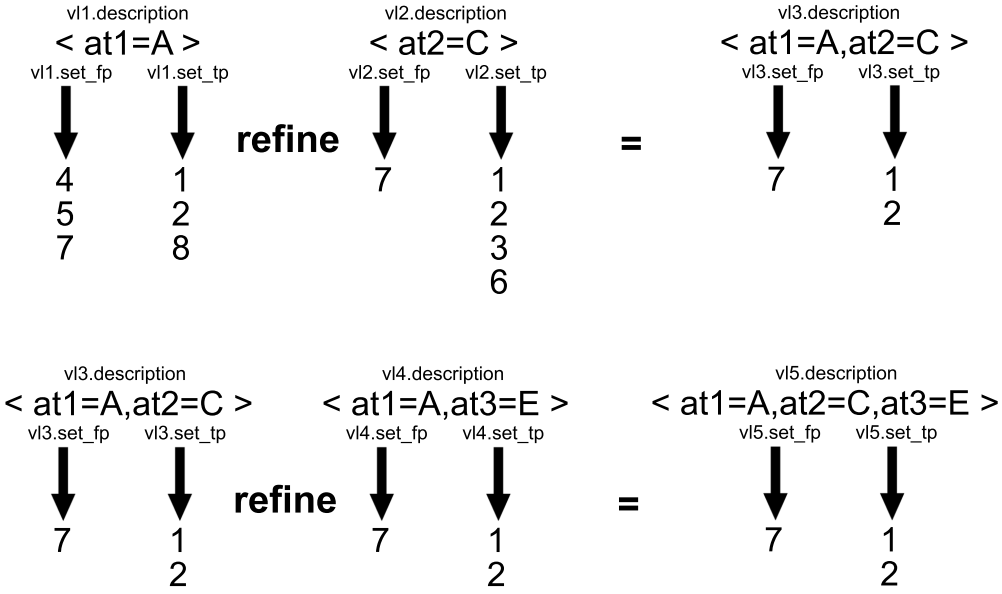

Figure 2. Examples of vertical list data structure and adapted refine operator.

It is important to state that both sets of IDs are actually implemented using bitsets in order to improve the efficiency to an even greater extent.

5. Experiments, Results and Discussion

The objective of these experiments was to test the performance of the VLSD algorithm with respect to some other well known state-of-the-art Subgroup Discovery (SD) algorithms. All experiments were implemented using a computer with an Intel Core i7-8700 3.20 GHz CPU, 32 GB of RAM memory, Windows 10, Anaconda3-2021.11 (x86_64), Python 3.9.7 (64 bits) and the following python libraries, pandas v1.3.4, numpy v1.20.3, and matplotlib v3.4.3. We used these python libraries because they are a reference in the Machine Learning field and they have been very used and tested by the community. Moreover, our proposal was implemented in `subgroups` python library (Source code available on: https://github.com/antoniolopezmc/subgroups, accessed on 24 May 2023).

We used a collection of standard, well-known, and popular datasets from the literature for performance evaluation. The following preprocessing pipeline was also applied to these datasets: (1) attribute type transformation (i.e., attributes that are actually nominal, but are represented with numerical values), (2) the treatment of missing values (imputing with the most frequent value in the nominal attributes and with the mean value in the numerical attributes), and (3) the discretization of numerical values using the Entropy based method [29]. Table 2 shows the datasets used in the experiments, along with their principal characteristics. Moreover, Table 3 shows the algorithms, along with their corresponding settings which were executed for this performance evaluation process. It is relevant to highlight that, although there are different heuristic SD algorithms, such as CN2-SD [15] or SDD++ [30], only exhaustive algorithms have been used in these experiments. Additionally, note that all these algorithms were implemented in the same programming language (Python 3) and strictly following the definitions from the original papers, and their results were also validated.

Table 2. Datasets used and their characteristics.

Name	Instances	Attributes	Selectors	Target
balloons	100	5	12	inflated = F
car-evaluation	1728	6	21	safety = acc
titanic	891	8	19	Survived = no
tic-tac-toe	958	10	29	class = positive
heart-disease	918	12	29	HeartDisease = yes
income	899	13	95	workclass = Private
vote	435	17	34	class = republican
lymph	148	19	54	class = malign_lymph
credit-g	1000	21	70	class = good
mushroom	8124	22	118	class = p

After all the aforementioned executions had been carried out, the following metrics were measured: runtime, max memory usage, subgroups selected, and nodes visited. The results obtained are depicted and explained in this section.

It is important to keep in mind that the search space of a problem (i.e., of a dataset) can be visually illustrated as a lattice in which the depth levels generally correspond to the number of attributes from the dataset and the nodes in each depth level correspond to the unique selectors extracted from the dataset. This means that (1) the more attributes in the input dataset the deeper the lattice, and (2) the greater the difference among the selectors in the input dataset the wider the lattice. Moreover, there is a fundamental difference between those algorithms that implement a pruning based on optimistic estimate and those that do not; while the former may not explore the complete search space, the latter always do so. This difference is shown in Figure 3.

Table 3. Algorithms and settings.

Algorithm	Quality Measure	Optimistic Estimate	Parameters
VLSD	WRAcc	Expression (17)	q_threshold and oe_threshold = −1, −0.25, 0, 0.25 both sort criteria = no reorder
SD-Map	WRAcc	-	threshold = −1, −0.25, 0, 0.25 min_support = 0
BSD	WRAcc	Expression (17)	top-k = 25, 50, 100, 250 min_support = 0 max_depth = maximum
Closed-BSD	WRAcc	Expression (17)	top-k = 25, 50, 100, 250 min_support = 0 max_depth = maximum
Closed-on-the-positives-BSD	WRAcc	Expression (17)	top-k = 25, 50, 100, 250 min_support = 0 max_depth = maximum

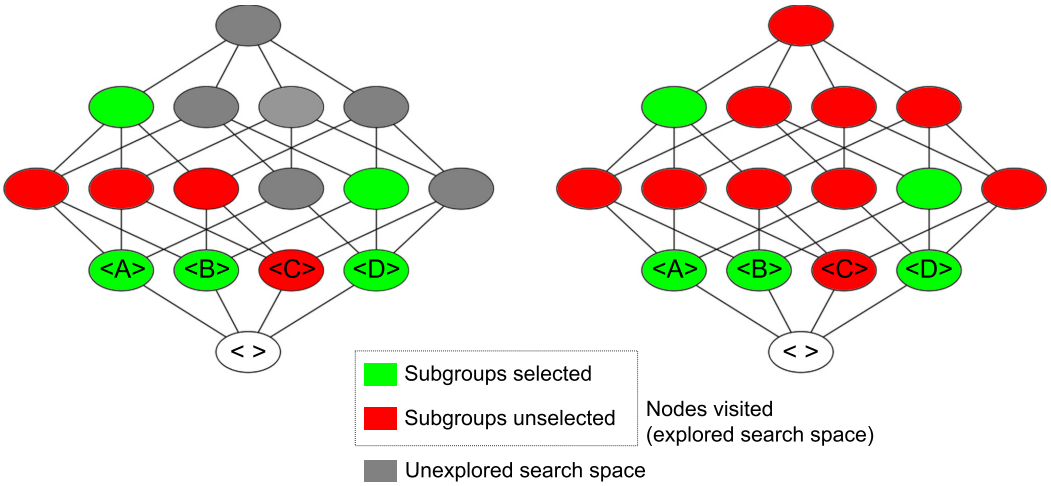

Figure 3. Examples of search spaces with (**left** side) and without (**right** side) optimistic estimate.

According to the above, it is to be expected that algorithms that do not implement a pruning based on optimistic estimate will have an exponential runtime and an exponential max memory usage with respect to the input size (i.e., dataset size). However, the utilization of optimistic estimates and other pruning strategies could, in practice, possibly produce other lower magnitude orders or at least make the exponential trend less steep. This is precisely one of the aspects that will be analyzed below.

In order to evaluate the scalability of the VLSD algorithm, 'mushroom' dataset is used to represent the runtime (see Figure 4) and the max memory usage (see Figure 5) when increasing the number of attributes (i.e., the depth of the lattice). This means that we start using only two attributes and we continue adding attributes up to 22 (i.e., all of them). Note that all instances are always used.

The scalability evaluation of the VLSD algorithm in terms of runtime (Figure 4) shows that there are significant differences between the datasets with less than 20 attributes and the datasets with more than 20 attributes. While the former spend less than 1 h, the latter spend significantly longer. Moreover, the runtime decreases considerably when using higher threshold values (i.e., when the search space is not completely explored). These results are owing to the exponential behavior of this algorithm (i.e., because it explores a data structure that grows exponentially in relation to the dataset size). Additionally, despite this exponential behavior, this figure also shows that the utilization of a pruning based on

optimistic estimate makes the exponential trend less steep. Here, it is possible to observe that, while the curves corresponding to the −1, −0.25 and 0 threshold values have the same trend, the curve corresponding to the 0.25 threshold value produces a less steep trend.

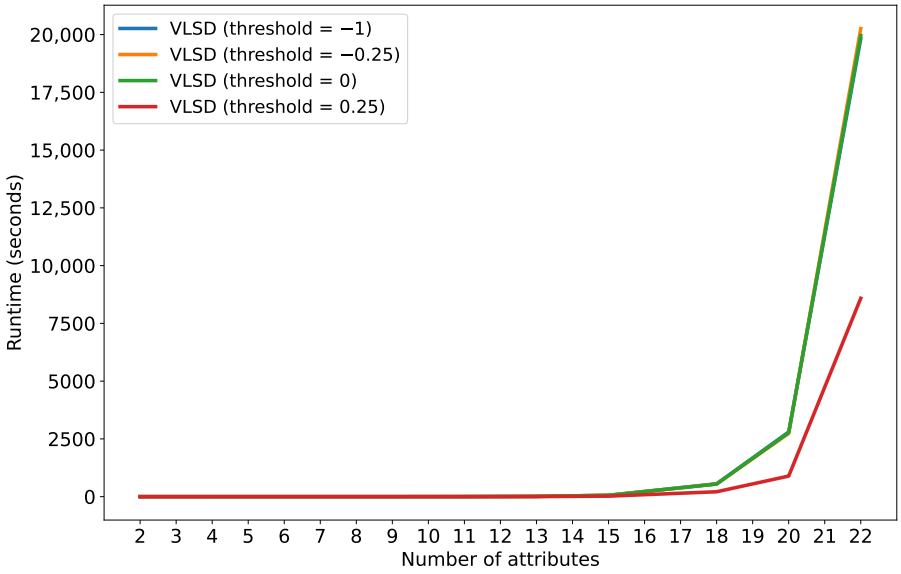

Figure 4. VLSD algorithm: runtime of mushroom dataset varying the number of attributes.

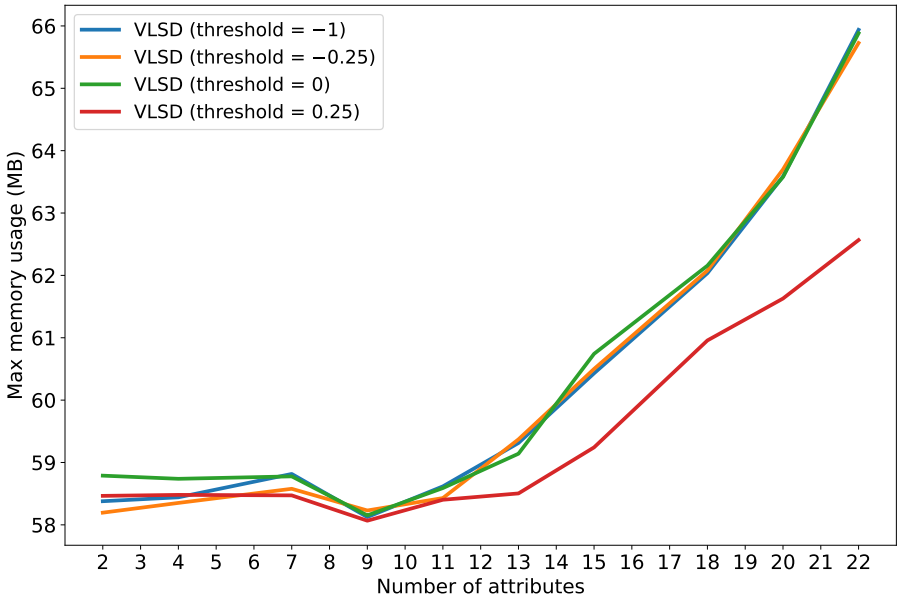

Figure 5. VLSD algorithm: max memory usage of mushroom dataset varying the number of attributes.

The scalability evaluation of the VLSD algorithm in terms of max memory usage (Figure 5) shows that the growth of the amount of memory in relation to the number of instances is not significant. Moreover, the max memory usage decreases when using higher threshold values (i.e., when the search space is not completely explored). These results are owing to the fact that, although the algorithm has exponential behavior, its design and the utilization of the equivalence class exploration strategy make it more efficient in relation to the max memory usage, because not all the search space is stored simultaneously in the memory (please recall that the regions already explored are being eliminated). This is a clear advantage when compared to the SD-Map algorithm, as will be shown below.

Additionally, Figures 6 and 7, which show the runtime and the max memory usage for each dataset and for each threshold value, also confirm the evidences about the VLSD algorithm described previously.

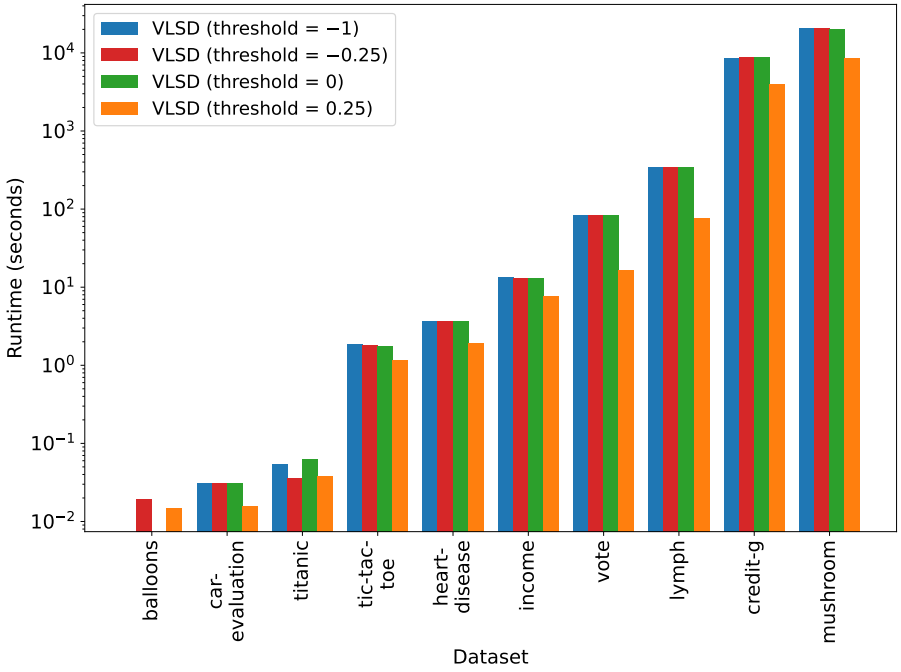

Figure 6. VLSD algorithm: runtime for each dataset (logarithmic scale).

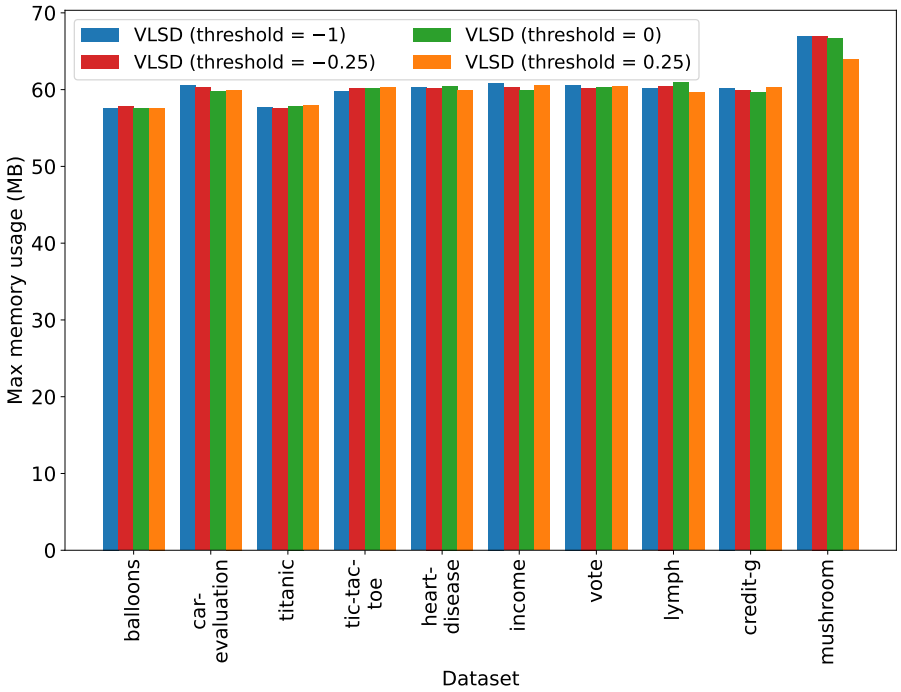

Figure 7. VLSD algorithm: max memory usage for each dataset.

Focusing on the runtime of the VLSD and SD-Map algorithms, Figure 8 shows that (1) there are significant differences among the executions of the VLSD algorithm (the higher the threshold, the less the time); (2) there are no significant difference between any of the execution of the SD-Map algorithm (i.e., using different threshold values), because that algorithm does not use a pruning based on optimistic estimate, and the complete search

space is, therefore, always explored; and (3) although both algorithms have an exponential trend, VLSD runtime is, in general, less than SD-Map runtime. Finally, Figure 9 also confirms these statements.

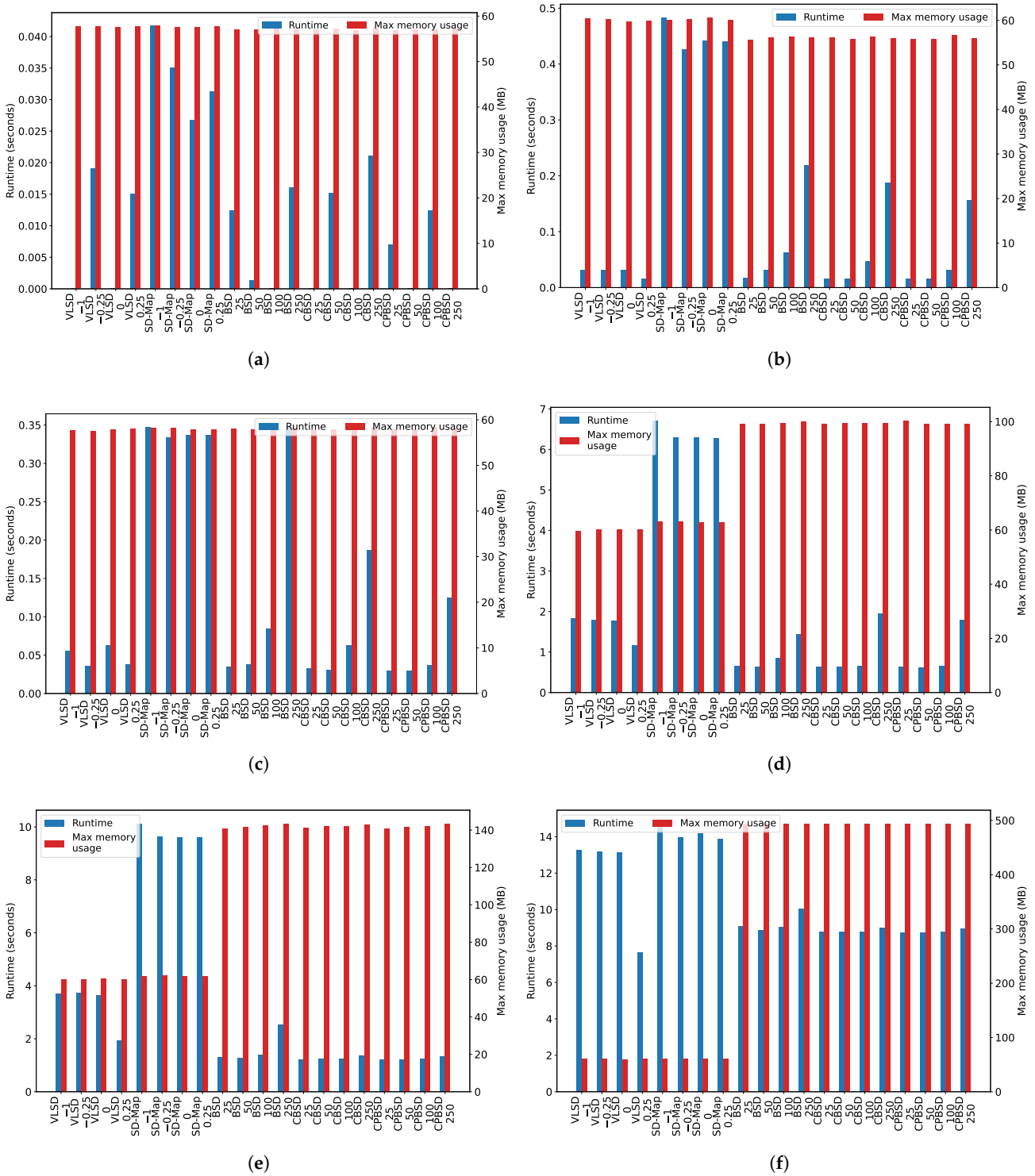

Figure 8. *Cont.*

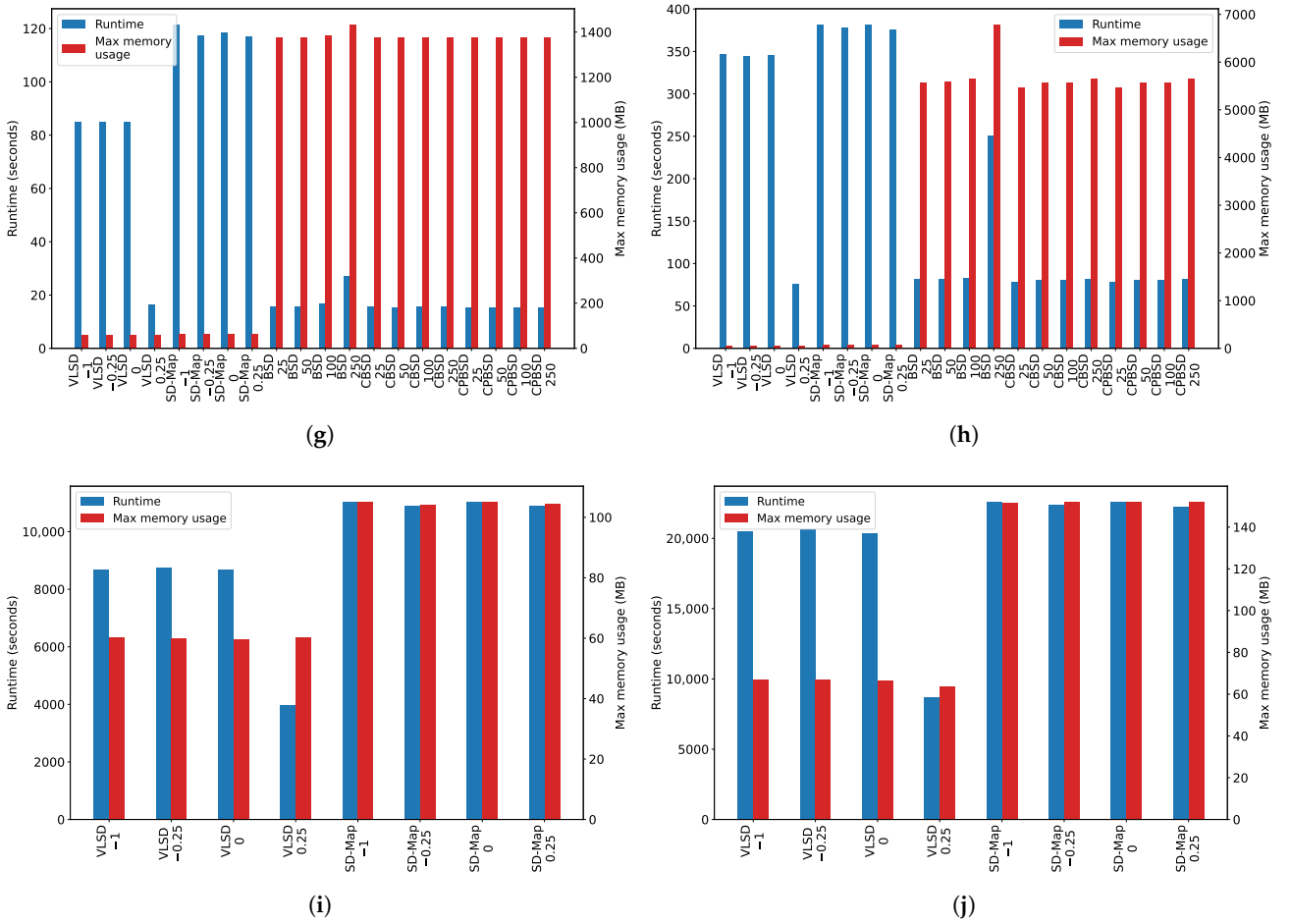

Figure 8. Runtime and max memory usage of all algorithms for each dataset. (**a**) balloons dataset. (**b**) car-evaluation dataset. (**c**) titanic dataset. (**d**) tic-tac-toe dataset. (**e**) heart-disease dataset. (**f**) income dataset. (**g**) vote dataset. (**h**) lymph dataset. (**i**) credit-g dataset. (**j**) mushroom dataset.

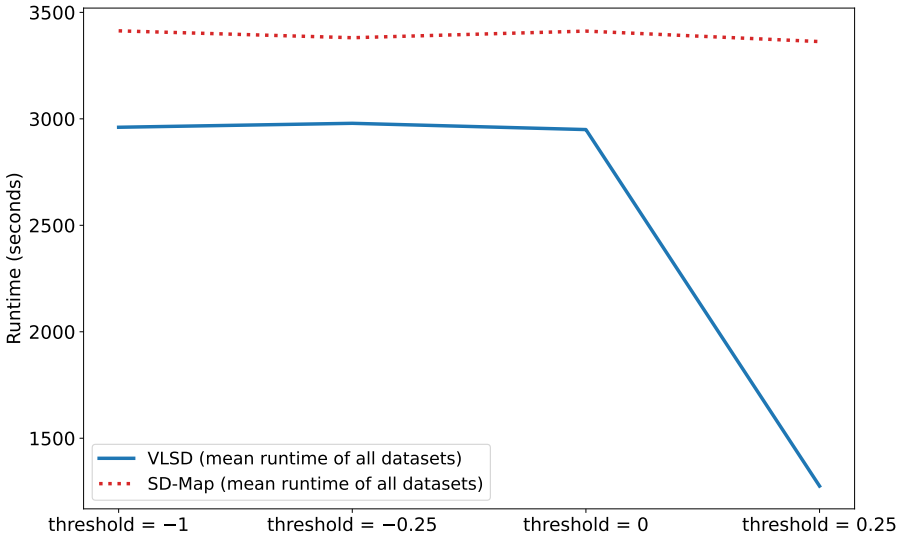

Figure 9. Mean runtime of all datasets for each quality threshold.

On the other hand, considering the runtime of the VLSD, BSD, CBSD, and CPBSD algorithms, Figure 8 shows that (1) there are significant differences when increasing the top-k parameter in the BSD algorithm, and (2) there are no significant differences when

increasing top-k parameter in the CBSD and CPBSD algorithms. The BSD algorithm explores a larger search space than the CBSD and CPBSD algorithms, which include an additional pruning for closed and closed-on-the-positives subgroups. The search space, therefore, increases in a more moderate manner in the CBSD and CPBSD algorithms when increasing the value of the top-k parameter. It is for this reason that the runtime increment of the BSD algorithm when increasing the top-k parameter is more significant than that of the CBSD and CPBSD algorithms. It will also be observed that (1) when the VLSD algorithm explores all the search space, its runtime is significantly higher than the runtime of the BSD, CBSD, and CPBSD algorithms; and (2) when the VLSD algorithm does not explore all the search space, there are no significant differences among its runtimes.

Concerning the max memory usage of the VLSD and SD-Map algorithms, Figure 8 shows that (1) there are no significant differences among the executions of the VLSD algorithm, because its design and the utilization of the equivalence class exploration strategy make it extremely efficient and, although the complete search space may not explored in all cases, the memory usage is, in general, always reduced; (2) there are no significant differences among any of the executions of the SD-Map algorithm (i.e., using different threshold values), because the complete search space is always stored in the FPTree data structure, and (3) there are significant differences between both algorithms (as will be clearly noted in Figure 8i,j). Finally, Figure 10 confirms these statements, because it shows that the mean of the max memory usage of all datasets for each quality threshold value is always more than 20% larger when using the SD-Map algorithm.

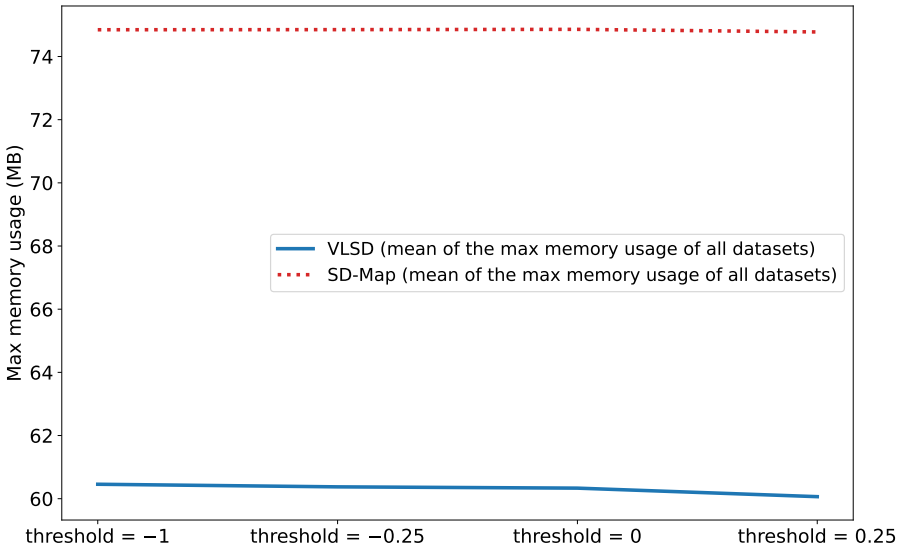

Figure 10. Mean of the max memory usage of all datasets for each quality threshold.

Comparing the max memory usage of the VLSD, BSD, CBSD, and CPBSD algorithms, Figure 8 shows the same behavior for BSD, CBSD, and CPBSD algorithms and for the same reasons as in the previous case. Additionally, note that, in general, these algorithms consume significantly more memory than the VLSD and SD-Map algorithms, and it is for this reason that it was impossible to execute them with the last two datasets.

When focusing on the search space nodes of the VLSD algorithm (Figure 11), it is important to state that, although it may not explore certain regions in the search space that have less quality owing to the utilization of the pruning based on optimistic estimate, this algorithm guarantees that the best subgroups will be found, because it is exhaustive.

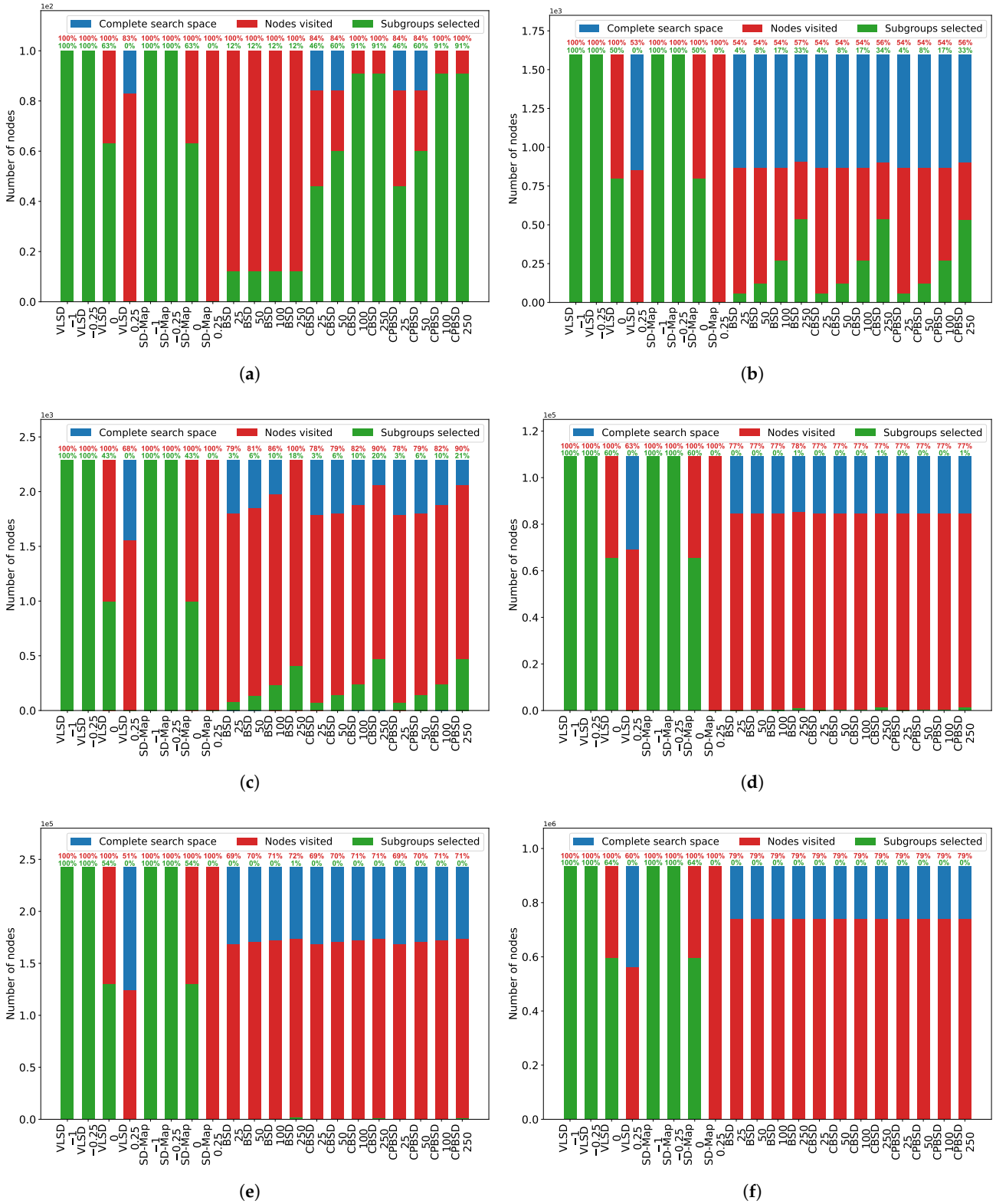

Figure 11. *Cont.*

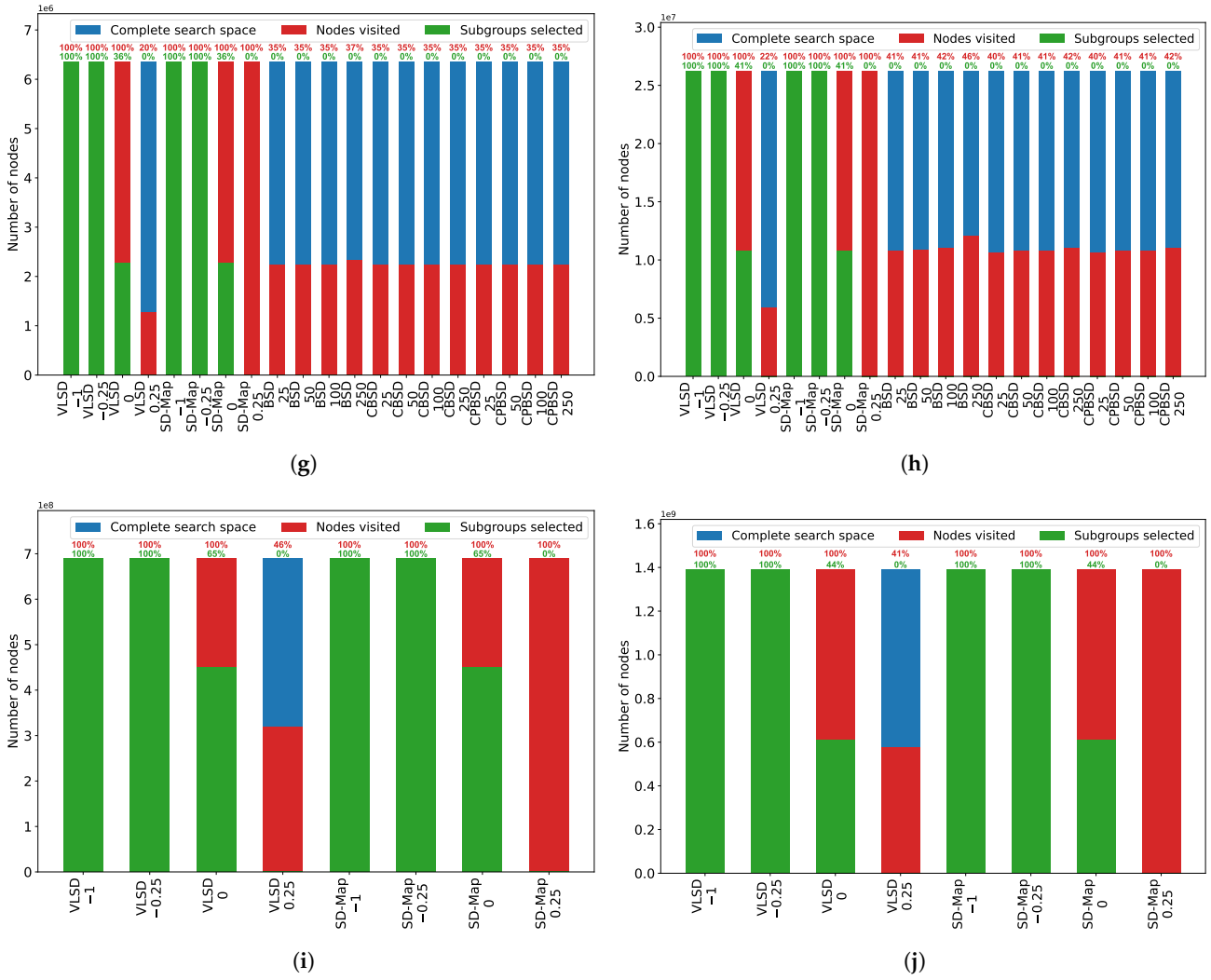

Figure 11. Search space nodes of all algorithms for each dataset. (**a**) balloons dataset. (**b**) car-evaluation dataset. (**c**) titanic dataset. (**d**) tic-tac-toe dataset. (**e**) heart-disease dataset. (**f**) income dataset. (**g**) vote dataset. (**h**) lymph dataset. (**i**) credit-g dataset. (**j**) mushroom dataset.

Regarding the search space nodes of the VLSD and SD-Map algorithms, Figure 11 shows that, first, the same subgroups are always generated for each dataset and for each threshold value. This proves that VLSD has been correctly designed and implemented, because it generates the same subgroups as the SD-Map, which is an exhaustive algorithm without optimistic estimate. Moreover, this figure also demonstrates the utilization of a pruning based on optimistic estimate, because, while the VLSD algorithm does not always explore the complete search space, the SD-Map algorithm always does so. Note that the VLSD algorithm explores fewer nodes when the threshold value is higher.

On the other hand, when comparing the search space nodes of the VLSD, BSD, CBSD, and CPBSD algorithms, Figure 11 shows that the BSD, CBSD, and CPBSD algorithms (1) do not explore the complete search space, because they use a pruning based on optimistic estimate; and (2) select significantly fewer subgroups than the VLSD and SD-Map algorithms, because they implement an additional pruning based on relevant subgroups (and, moreover, CBSD and CPBSD also implement another pruning based on closed subgroups and closed-on-the-positives subgroups, respectively).

It is necessary to state that the bitsets used by the VLSD algorithm are different from those employed by the BSD, CBSD and CPBSD algorithms. While our algorithm considers all the dataset instances in both bitsets, the others use bitsets of different sizes.

In summary, when comparing the VLSD and SD-Map algorithms, it will be noted that the utilization of a pruning based on optimistic estimate by the VLSD algorithm has an evident impact. It will also be noted that, overall, this pruning strategy allows the VLSD algorithm to spend less time, consume less memory, and visit fewer nodes; all of this while remaining exhaustive and generating the same subgroups. Additionally, when comparing the VLSD, BSD, CBSD, and CPBSD algorithms, it will be noted that the last three algorithms are at a clear disadvantage with respect to the VLSD algorithm as regards the max memory usage. However, it will also be noted that, overall, the BSD, CBSD, and CPBSD algorithms spend less time and select less nodes owing to the pruning based on optimistic estimate, relevant subgroups, closed subgroups, and closed-on-the-positives subgroups.

6. Conclusions

This research was carried out in order to design and implement a new exhaustive Subgroup Discovery (SD) algorithm that would be more efficient than the state-of-the-art algorithms. We have proposed the VLSD algorithm, along with a new data structure denominated as a vertical list. This algorithm is based on the equivalence class exploration strategy and uses a pruning based on optimistic estimate.

Note that, although all these concepts already appear in the literature separately, this is the first time that they are used, implemented, and validated together.

Some existing SD algorithms, such as SD-Map or BSD, have adapted and used classical data structures, such as FPTree or Bitsets. Our algorithm uses a vertical list data structure, which represents both a subgroup and the dataset instances in which it appears. Moreover, it provides an easy and efficient computation of the subgroup refinements. The VLSD algorithm is also easily parallelizable owing to the utilization of the equivalence class exploration strategy, along with the aforementioned data structure.

Our experiments were carried out using a collection of standard, well-known, and popular datasets from the literature, and analyzed certain metrics, such as runtime, max memory usage, subgroups selected, and nodes visited. They confirmed that, overall, our approach is more efficient than the other algorithms considered.

Additionally, as an example of practical implications, this algorithm could be applied to certain specific domains, e.g., medical research or patient phenotyping.

Future research could continue and extend the algorithm in different ways. First, certain modifications could be made in order to avoid the need to extract all the subgroups explored (e.g., extracting only the top-k subgroups). Finally, some other pruning strategies could be added in order to make the VLSD algorithm even more efficient (e.g., closed subgroups or closed-on-the-positives subgroups).

Author Contributions: All authors contributed equally to this work: conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing—original draft preparation, writing—review and editing, visualization, supervision, A.L.-M.-C., J.M.J., M.C. and B.C.-S.; project administration, J.M.J. and M.C.; funding acquisition, A.L.-M.-C., J.M.J. and M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially funded by the CONFAINCE project (Ref: PID2021-122194OB-I00) by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", by the "European Union" or by the "European Union NextGenerationEU/PRTR", and by the GRALENIA project (Ref: 2021/C005/00150055) supported by the Spanish Ministry of Economic Affairs and Digital Transformation, the Spanish Secretariat of State for Digitization and Artificial Intelligence, Red.es and by the NextGenerationEU funding. Moreover, this research was also partially funded by a national grant (Ref:FPU18/02220), financed by the Spanish Ministry of Science, Innovation and Universities (MCIU).

Data Availability Statement: We use a collection of public, well-known and popular datasets from the literature. For easing the reproducibility of our research, our proposal is implemented in subgroups python library, which is available on PyPI and on https://github.com/antoniolopezmc/subgroups, accessed on 24 May 2023.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Extended Problem Definition

The fundamental concepts of the Subgroup Discovery (SD) technique are extended and detailed as follows:

Definition A1 (Attribute a)**.** *An attribute a is a unique characteristic of an object, which has an associated value. An example of an attribute is a = age:30.*

Definition A2 (Domain of an attribute a)**.** *The domain of an attribute a (denoted as $dom(a)$) is the set of all the unique values that said attribute can take. An attribute can be nominal or numeric, depending on its domain.*

Definition A3 (Instance i)**.** *An instance i is a tuple $i = (a_1, \ldots, a_M)$ of attributes. Given the attributes a_1 = age:25 and a_2 = sex:woman, an example of an instance is i = (age:25, sex:woman).*

Definition A4 (Dataset d)**.** *A dataset d is a tuple $d = (i_1, \ldots, i_N)$ of instances. Given the instances i_1 = (age:30, sex:man) and i_2 = (age:25, sex:woman), an example of a dataset is d = ((age:30, sex:man), (age:25, sex:woman)).*

The dataset space is denoted as $\mathcal{D}$.

All values from a dataset d can be indexed with two integers, x and y. We use the notation $v_{x,y}$ to indicate the value of the x-th instance i_x and of the y-th attribute a_y from a dataset d.

Definition A5 (Selector e)**.** *Given an attribute a_y from a dataset d, a binary operator $\in \{=, \neq, <, >, \leq, \geq\}$ and a value $w \in dom(a_y)$, a selector e is a 3-tuple of the form $(a_y.characteristic, operator, w)$. Note that when an attribute a_y is nominal, only the $=$ and $\neq$ operators are permitted.*

Informally, a selector is a binary relation between an attribute from a dataset and a value in the domain of that attribute. This relation represents a property of a subset of instances from that dataset.

It is essential to bear in mind that the first element of a selector refers only to the attribute name, i.e., the characteristic, and not to the complete attribute itself.

Definition A6 (Selector covering)**.** *Given an instance i_x and an attribute a_y from a dataset d, and a selector $e = (a_y.characteristic, operator, w \in dom(a_y))$, then i_x is covered by e (denoted as $i_x \sqsubset e$) if the binary expression "$v_{x,y}$ operator w" holds true. Otherwise, we say that it is not covered by e (denoted as $i_x \not\sqsubset e$).*

For example, given the instance i_x = (*age*:25, *sex*:*woman*) and the selectors e_1 = (*age*, <, 20) and e_2 = (*sex*, =, *woman*), it will be noted that $i_x \not\sqsubset e_1$ and $i_x \sqsubset e_2$.

Definition A7 (Pattern p)**.** *A pattern p is a list of selectors $< e_1, \ldots, e_j >$ in which all attributes of the selectors are different. Moreover, its size (denoted as $|p|$) is defined as the number of selectors that it contains.*

In general, a pattern is interpreted as a list of selectors (i.e., as a conjunction) that represents a list of properties of a subset of instances from a dataset.

Definition A8 (Pattern covering)**.** *Given an instance i_x from a dataset d and a pattern p, then i_x is covered by p (denoted as $i_x \sqsubset p$) if $\forall e \in p, i_x \sqsubset e$. Otherwise, we say that it is not covered by p (denoted as $i_x \not\sqsubset p$).*

Definition A9 (Subgroup s)**.** *A subgroup s is a pair (pattern, selector) in which the pattern is denoted as s.description and the selector is denoted as s.target. Given the dataset d = ((fever:yes, sex:man, flu:yes), (fever:yes, sex:woman, flu:no)), an example of a subgroup is* $s = (< (fever, =, yes), (sex, =, woman) >, (flu, =, yes))$.

The subgroup space is denoted as $\mathcal{S}$.

Definition A10 (Subgroup refinement s')**.** *Given a subgroup s, each of its refinements s' (denoted as $s \prec s'$) is a subgroup with the same target, $s'.target = s.target$, and with an extended description, $s'.description = concat(s.description, < e_1, \ldots, e_j >)$.*

Definition A11 (Refine operator)**.** *Given two subgroups, s_x and s_y, the $refine$ operator generates a refinement $s_{x,y}$ of s_x, extending its description with the non-common suffix of s_y. For example, if s_x.description = <e_1> and s_y.description = <e_2>, then $s_{x,y}$.description = <e_1, e_2>; and if s_x.description = <e_1, e_2, e_3> and s_y.description = <e_1, e_2, e_4>, then $s_{x,y}$.description = <e_1, e_2, e_3, e_4>. Formally:*

$$refine : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S} \tag{A1}$$

This means that the $refine$ operator takes as an input two subgroups and generates as an output one subgroup.

Definition A12 (Quality Measure q)**.** *Given a subgroup s and a dataset d, a quality measure q is a function that computes one numeric value according to that subgroup s and to certain characteristics from that dataset d. Formally:*

$$q : \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{R} \tag{A2}$$

$$q(s, d) \in \mathbb{R} \tag{A3}$$

Definition A13 (Optimistic Estimate oe)**.** *Given a quality measure q and a dataset d, an optimistic estimate oe of q is a quality measure that satisfies the following condition:*

$$\forall s, s' \ , \ s \prec s' \Rightarrow oe(s, d) \geq q(s', d) \tag{A4}$$

Informally, an optimistic estimate is a quality measure which, for a certain subgroup, provides a quality upper bound for all its refinements [9].

Focusing on a specific subgroup s and on a specific dataset d, the following functions can be defined:

Definition A14 (Function tp (true positives))**.** *The function tp is defined as the number of instances i_x from the dataset d that are covered by the subgroup description s.description and by the subgroup target s.target. Formally:*

$$tp : \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{N} \tag{A5}$$

$$tp(s, d) = |\{i_x \in d : i_x \sqsubset s.description \wedge i_x \sqsubset s.target\}| \tag{A6}$$

Definition A15 (Function fp (false positives))**.** *The function fp is defined as the number of instances i_x from the dataset d that are covered by the subgroup description s.description, but not by the subgroup target s.target. Formally:*

$$fp : \mathcal{S} \times \mathcal{D} \rightarrow \mathbb{N} \tag{A7}$$

$$fp(s, d) = |\{i_x \in d : i_x \sqsubset s.description \wedge i_x \not\sqsubset s.target\}| \tag{A8}$$

Definition A16 (Function *TP* (true population))**.** *The function TP is defined as the number of instances i_x from the dataset d that are covered by the subgroup target s.target. Formally:*

$$TP : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A9}$$

$$TP(s, d) = |\{i_x \in d : i_x \sqsubset s.target\}| \tag{A10}$$

Definition A17 (Function *FP* (false population))**.** *The function FP is defined as the number of instances i_x from the dataset d that are not covered by the subgroup target s.target. Formally:*

$$FP : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A11}$$

$$FP(s, d) = |\{i_x \in d : i_x \not\sqsubset s.target\}| \tag{A12}$$

A quality measure q can, therefore, be formally redefined using the previous four functions as follows:

Definition A18 (Quality Measure q)**.** *Given a subgroup s and a dataset d, a quality measure q is a function that computes one numeric value according to the functions tp, fp, TP, and FP. Formally:*

$$q : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{R} \tag{A13}$$

$$q(tp(s, d), fp(s, d), TP(s, d), FP(s, d)) \in \mathbb{R} \tag{A14}$$

The four functions described above are sufficiently expressive to compute any quality measure. However, the following are also used in the literature:

Definition A19 (Function n)**.** *The function n is defined as the number of instances i_x from a dataset d that are covered by the subgroup description s.description. Formally:*

$$n : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A15}$$

$$n(s, d) = |\{i_x \in d : i_x \sqsubset s.description\}| \tag{A16}$$

Definition A20 (Function N)**.** *The function N is defined as the number of instances i_x from the dataset d. Formally:*

$$N : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A17}$$

$$N(s, d) = |\{i_x \in d : i_x\}| \tag{A18}$$

Definition A21 (Function p)**.** *The function p is defined as the distribution of the subgroup target s.target with respect to the instances i_x from a dataset d covered by the subgroup description s.description. Formally:*

$$p : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A19}$$

$$p(s, d) = \frac{|\{i_x \in d : i_x \sqsubset s.description \wedge i_x \sqsubset s.target\}|}{|\{i_x \in d : i_x \sqsubset s.description\}|} \tag{A20}$$

Definition A22 (Function p_0)**.** *The function p_0 is defined as the distribution of the subgroup target s.target with respect to all instances i_x from a dataset d. Formally:*

$$p_0 : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A21}$$

$$p_0(s, d) = \frac{|\{i_x \in d : i_x \sqsubset s.target\}|}{|\{i_x \in d : i_x\}|} \tag{A22}$$

Definition A23 (Function tn (true negatives)). *The function tn is defined as the number of instances i_x from the dataset d that are covered by neither the subgroup description $s.description$ nor the subgroup target $s.target$. Formally:*

$$tn : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A23}$$

$$tn(s,d) = |\{i_x \in d : i_x \not\sqsubset s.description \wedge i_x \not\sqsubset s.target\}| \tag{A24}$$

Definition A24 (Function fn (false negatives)). *The function fn is defined as the number of instances i_x from the dataset d that are not covered by the subgroup description $s.description$, but are covered by the subgroup target $s.target$. Formally:*

$$fn : \mathcal{S} \times \mathcal{D} \to \mathbb{N} \tag{A25}$$

$$fn(s,d) = |\{i_x \in d : i_x \not\sqsubset s.description \wedge i_x \sqsubset s.target\}| \tag{A26}$$

Table 1 shows the confusion matrix of a subgroup s with respect to a dataset d. This matrix summarizes the functions describe above.

With regard to the functions defined previously, some equivalences are defined in Section 3. Moreover, some popular quality measures for SD presented in the literature are described in that section.

Definition A25 (Subgroup Discovery problem). *Given a dataset d, a quality measure q and a numeric value $quality\_threshold$, the subgroup discovery problem consists of exploring the search space of d in order to enumerate the subgroups that have a quality measure value above the selected threshold. Formally:*

$$\mathcal{R} = \{(s, q(s,d)) | q(s,d) \geq quality\_threshold\} \tag{A27}$$

The search space of a problem (i.e., of a dataset d) can be visually illustrated as a lattice [24] (see Figure 1). According to this comparison, the first level of the search space of a problem contains all those subgroups s whose descriptions have a size of 1 (i.e., $|s.description|$ is equal to 1), the second level of the search space of a problem contains all those subgroups s whose descriptions have a size of 2 (i.e., $|s.description|$ is equal to 2) and, in general, the level n of the search space of a problem contains all those subgroups s whose descriptions have the size n (i.e., $|s.description|$ is equal to n).

References

1. Atzmueller, M. Subgroup Discovery—Advanced Review. *WIREs: Data Min. Knowl. Discov.* **2015**, *5*, 35–49.
2. Atzmüller, M.; Puppe, F.; Buscher, H.P. Exploiting Background Knowledge for Knowledge-Intensive Subgroup Discovery. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, Edinburgh, UK, 30 July–5 August 2005; pp. 647–652.
3. Gamberger, D.; Lavrac, N. Expert-Guided Subgroup Discovery: Methodology and Application. *J. Artif. Intell. Res.* **2002**, *17*, 501–527. [CrossRef]
4. Jorge, A.M.; Pereira, F.; Azevedo, P.J. Visual Interactive Subgroup Discovery with Numerical Properties of Interest. In Proceedings of the Discovery Science, Barcelona, Spain, 7–10 October 2006; pp. 301–305.
5. Duivesteijn, W.; Knobbe, A. Exploiting False Discoveries—Statistical Validation of Patterns and Quality Measures in Subgroup Discovery. In Proceedings of the 2011 IEEE 11th International Conference on Data Mining, Vancouver, BC, Canada, 11–14 December 2011; pp. 151–160.
6. Ventura, S.; Luna, J.M. *Supervised Descriptive Pattern Mining*; Springer: Berlin/Heidelberg, Germany, 2018.
7. Lopez-Martinez-Carrasco, A.; Juarez, J.M.; Campos, M.; Canovas-Segura, B. Phenotypes for Resistant Bacteria Infections Using an Efficient Subgroup Discovery Algorithm. In Proceedings of the Artificial Intelligence in Medicine, Virtual Event, 15–18 June 2021; pp. 246–251.
8. Herrera, F.; Carmona, C.J.; González, P.; Del Jesus, M.J. An overview on subgroup discovery: Foundations and applications. *Knowl. Inf. Syst.* **2011**, *29*, 495–525. [CrossRef]

9. Grosskreutz, H.; Rüping, S.; Wrobel, S. Tight Optimistic Estimates for Fast Subgroup Discovery. In Proceedings of the Proc. of Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Antwerp, Belgium, 15–19 September 2008; pp. 440–456.
10. Klösgen, W. Explora: A Multipattern and Multistrategy Discovery Assistant. In *Advances in Knowledge Discovery and Data Mining*; American Association for Artificial Intelligence: Washington, DC, USA, 1996; pp. 249–271.
11. Wrobel, S. An algorithm for multi-relational discovery of subgroups. In Proceedings of the Principles of Data Mining and Knowledge Discovery, Trondheim, Norway, 24–27 June 1997; pp. 78–87.
12. Friedman, J.; Fisher, N. Bump hunting in high-dimensional data. *Stat. Comput.* **1999**, *9*, 123–143. [CrossRef]
13. Klösgen, W.; May, M. Census Data Mining—An Application. In Proceedings of the 6th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2002), Helsinki, Finland, 19–23 August 2002; p. 733–739.
14. Lavrac, N.; Železný, F.; Flach, P. RSD: Relational Subgroup Discovery through First-Order Feature Construction. In Proceedings of the Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), Sydney, Australia, 9–11 July 2002; Volume 2583, pp. 149–165.
15. Lavrac, N.; Kavsek, B.; Flach, P.A.; Todorovski, L. Subgroup Discovery with CN2-SD. *J. Mach. Learn. Res.* **2004**, *5*, 153–188.
16. Lavrac, N.; Gamberger, D. Relevancy in Constraint-Based Subgroup Discovery. In Proceedings of the European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, 11–13 March 2004; pp. 243–266.
17. Kavšek, B.; Lavrac, N.; Jovanoski, V. APRIORI-SD: Adapting association rule learning to subgroup discovery. In Proceedings of the International Symposium on Intelligent Data Analysis, Berlin, Germany, 28–30 August 2003; Volume 20, pp. 230–241.
18. Mueller, M.; Rosales, R.; Steck, H.; Krishnan, S.; Rao, B.; Kramer, S. Subgroup Discovery for Test Selection: A Novel Approach and Its Application to Breast Cancer Diagnosis. In Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII, Lyon, France, 31 August–2 September 2009; p. 119–130.
19. Lemmerich, F.; Atzmüller, M.; Puppe, F. Fast exhaustive subgroup discovery with numerical target concepts. *Data Min. Knowl. Discov.* **2015**, *30*, 711–762. [CrossRef]
20. Atzmueller, M.; Puppe, F. SD-Map—A Fast Algorithm for Exhaustive Subgroup Discovery. In Proceedings of the Knowledge Discovery in Databases (PKDD 2006), Berlin, Germany, 18–22 September 2006; pp. 6–17.
21. Lemmerich, F.; Rohlfs, M.; Atzmüller, M. Fast Discovery of Relevant Subgroup Patterns. In Proceedings of the 23rd International Florida Artificial Intelligence Research Society Conference (FLAIRS-23), Daytona Beach, FL, USA, 19–21 May 2010.
22. Han, J.; Pei, J.; Yin, Y. Mining Frequent Patterns without Candidate Generation. *SIGMOD Rec.* **2000**, *29*, 1–12. [CrossRef]
23. Garriga, G.; Kralj Novak, P.; Lavrac, N. Closed Sets for Labeled Data. *J. Mach. Learn. Res.* **2006**, *9*, 163–174.
24. Zaki, M.J.; Parthasarathy, S.; Ogihara, M.; Li, W. Parallel Algorithms for Discovery of Association Rules. *Data Min. Knowl. Discov.* **1997**, *1*, 343–373. [CrossRef]
25. Nouioua, M.; Fournier Viger, P.; Wu, C.W.; Lin, C.W.; Gan, W. FHUQI-Miner: Fast high utility quantitative itemset mining. *Appl. Intell.* **2021**, *51*, 6785–6809. [CrossRef]
26. Qu, J.F.; Fournier-Viger, P.; Liu, M.; Hang, B.; Wang, F. Mining high utility itemsets using extended chain structure and utility machine. *Knowl.-Based Syst.* **2020**, *208*, 106457. [CrossRef]
27. Le, T.; Nguyen, T.L.; Huynh, B.; Nguyen, H.; Hong, T.P.; Snasel, V. Mining colossal patterns with length constraints. *Appl. Intell.* **2021**, *51*, 8629–8640. [CrossRef]
28. Fournier-Viger, P.; Gomariz, A.; Campos, M.; Thomas, R. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In Proceedings of the Advances in Knowledge Discovery and Data Mining—18th Pacific-Asia Conference (PAKDD), Tainan, Taiwan, 13–16 May 2014; Volume 8443, pp. 40–52.
29. Fayyad, U.M.; Irani, K.B. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93), Chambéry, France, 28 August–3 September 1993.
30. Proença, H.M.; Grünwald, P.; Bäck, T.; van Leeuwen, M. Robust subgroup discovery. *Data Min. Knowl. Discov.* **2022**, *36*, 1885–1970. [CrossRef]