The Minimum Consistent Subset Cover Problem and its Applications in Data Mining

Byron J. Gao[1,2], Martin Ester[1], Jin-Yi Cai[2], Oliver Schulte[1], and Hui Xiong[3]

[1] School of Computing Science, Simon Fraser University, Canada

[2] Computer Sciences Department, University of Wisconsin - Madison, USA

[3] Department of Management Science & Information Systems, Rutgers University, USA

ABSTRACT

In this paper, we introduce and study the Minimum Consistent Subset Cover (MCSC) problem. Given a finite ground set X and a constraint t, find the minimum number of consistent subsets that cover X, where a subset of X is consistent if it satisfies t. The MCSC problem generalizes the traditional set covering problem and has Minimum Clique Partition, a dual problem of graph coloring, as an instance. Many practical data mining problems in the areas of rule learning, clustering, and frequent pattern mining can be formulated as MCSC instances. In particular, we discuss the Minimum Rule Set problem that minimizes model complexity of decision rules as well as some converse k-clustering problems that minimize the number of clusters satisfying certain distance constraints. We also show how the MCSC problem can find applications in frequent pattern summarization. For any of these MCSC formulations, our proposed novel graph-based generic algorithm *CAG* can be directly applicable. *CAG* starts by constructing a maximal optimal partial solution, then performs an example-driven specific-to-general search on a dynamically maintained bipartite assignment graph to simultaneously learn a set of consistent subsets with small cardinality covering the ground set. Our experiments on benchmark datasets show that *CAG* achieves good results compared to existing popular heuristics.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications – *Data Mining*

General Terms: algorithms, theory, performance

Keywords: minimum consistent subset cover, minimum rule set, converse k-clustering, pattern summarization

1. INTRODUCTION

Combinatorial optimization problems such as set covering and graph coloring have been extensively studied. By making connections to these classical problems, we can gain important insights into practical data mining applications.

In this paper, we introduce and study the Minimum Consistent Subset Cover (MCSC) problem. Given a finite ground set X and a constraint t, find the minimum number of consistent subsets that cover X, where a subset of X is consistent if it satisfies t. The MCSC problem provides one way of generalizing the traditional set covering problem [12], where a subset of X is consistent if it is a given subset. Different from set covering, in typical MCSC instances the consistent subsets are not explicitly given and they need to be generated. For example, Minimum Clique Partition (MCP), a dual problem of graph coloring, can be considered as an MCSC instance, where a subset is consistent if it forms a clique and the cliques are not given as input.

As a practical application of the MCSC problem in rule learning, the Minimum Rule Set (MRS) problem finds a complete and consistent set of rules with the minimum cardinality for a given set of labeled examples. The completeness and consistency constraints require correct classifications of all the given examples. With the goal of minimizing model complexity, the MRS problem can be motivated from both data classification and data description applications. The MRS problem is a typical MCSC instance, where a subset is consistent if it forms a consistent rule, i.e., the bounding box of the subset contains no examples of other classes.

As a prominent clustering model, k-clustering generates k clusters minimizing some objective, such as maximum radius as in the k-center problem [21] or maximum diameter as in the pairwise clustering problem [4]. The *radius* of a cluster is the maximum distance between a fixed point (center) and any point in the cluster, and the *diameter* is the maximum distance between any two points in the cluster. Since the number of clusters is often hard to determine in advance, converse k-clustering can be a more appropriate clustering model, where a maximum radius or diameter threshold is given and the number of clusters k is to be minimized. The converse k-center and converse pairwise clustering problems are both MCSC instances, where a subset is consistent if it forms a cluster satisfying a given distance constraint.

Frequent pattern mining has been a trademark of data mining. While the mining efficiency has been greatly improved, interpretability instead became a bottleneck to its successful application. As a known problem, the overwhelmingly large number of generated frequent patterns containing redundant information are in fact "inaccessible knowledge" that need to be further mined and explored. Thus, summarization of large collections of patterns in the pursuit of usability has emerged as an important research problem. The converse k-clustering models discussed above as well as some other MCSC formulations appear to be a very reasonable and promising approach towards this problem.

These formulated MCSC instances generally feature anti-monotonic constraints, under which any subset of a consistent subset is also consistent. Such instances allow the design of efficient and effective heuristic algorithms. Stimulated by a theoretical study, our graph-based generic algorithm *CAG* starts by constructing a maximal optimal partial solution, and then performs an example-driven specific-to-general search on a dynamically maintained bipartite assignment graph to simultaneously learn a small consistent subset cover. The construction of initial optimal partial solution, the use of assignment graph allowing good choices of both the element and the consistent subset in an assignment, and the example-driven simultaneous learning (in contrast to the most-seen separate-and-conquer) strategy are the three novel design ideas that help to well guide the search leading to good performance of *CAG*, as demonstrated by our experiments on rule learning and graph coloring benchmarks in comparison with existing popular heuristics.

Contributions. (1) We introduce and study the Minimum Consistent Subset Cover (MCSC) problem, which generalizes the set covering problem and has Minimum Clique Partition, a dual problem of graph coloring, as an instance.

(2) The MCSC problem has many practical applications in data mining. In particular, we study the Minimum Rule Set problem for rule learning. We also discuss applications in converse k-clustering and frequent pattern summarization.

(3) To solve MCSC instances, we present a graph-based generic algorithm *CAG* with several novel design ideas, whose performance is justified by our experimental evaluation.

2. PRELIMINARIES AND RELATED WORK

In this section we provide preliminaries on graph theory and a review of related work in a very concise manner.

Preliminaries. A graph is *complete* if all of its vertices are pairwise adjacent. A *clique* of a graph $G = (V, E)$ is a subset of V such that the induced subgraph is complete. An *independent set* of G is a subset of V such that no two vertices in the subset are connected in G. The *independence number* of G, usually denoted by $\alpha(G)$, is the cardinality of the largest independent set. Independent sets and cliques are opposite in the sense that every independent set in G corresponds to a clique in the complementary graph $\overline{G}$.

We say $V' \subseteq V$ is a *dominating set* if for all $u \in V - V'$, there is some $v \in V'$ such that $(u, v) \in E$. A maximal independent set, say V', is also a dominating set. If it is not, there must be some $u \in V - V'$ that is not adjacent to any $v \in V'$, then u can be added to V' to form a larger independent set, but then V' is not maximal.

The Minimum Clique Partition (MCP) problem is to find a partitioning of the vertices of a graph G into the minimum number of disjoint vertex sets, each of which must be a clique in G. That minimum number is called the *clique partition number* of G and usually denoted by $\overline{\chi}(G)$.

The well-known graph coloring problem is to use the minimum number of colors to color the vertices of a graph G such that no adjacent vertices receive the same color. That minimum number is called the *chromatic number* of G and usually denoted by $\chi(G)$. Since the vertices of an independent set can be safely colored with the same color, the graph coloring problem precisely minimizes the number of disjoint independent sets of G that form a partition of V.

Apparently, MCP is a dual problem of graph coloring. An instance of the former on G is an instance of the latter on $\overline{G}$. Thus, we have $\overline{\chi}(G) = \chi(\overline{G})$.

Observation 1. $\alpha(G) \leq \overline{\chi}(G)$.

The observation states that the clique partition number of a graph is lower-bounded by its independence number. It is rather straightforward since two vertices in an independent set cannot appear together in a clique.

Related work. The traditional set covering problem [12] finds the minimum number of subsets from a given collection of subsets that cover a given ground set. It is one of the most fundamental algorithmic problems that has many variants, settings, and applications. The problem is NP-hard and there is no constant factor approximation. It is approximable within $1 + \log n$ by a simple greedy algorithm [12], which iteratively selects the subset that covers the largest number of uncovered elements until the ground set is covered. This greedy algorithm essentially adopts a separate-and-conquer approach as seen in most rule learners.

In the MCSC problem we study, the subsets are not explicitly given. Instead, a constraint is given and used to qualify the subsets that can be used in a cover. The MCP problem and many practical data mining applications can be formulated as MCSC instances.

Graph coloring heuristics can be applied to complementary graphs to solve MCP instances. *DSATUR* [3] is one of the most popular construction heuristics. In *DSATUR*, a vertex with the largest number of different colors assigned to adjacent vertices is chosen and assigned with the first feasible color, where colors are pre-ordered. Ties are broken favoring the vertex with the largest number of uncolored adjacent vertices. *DSATUR* has a cubic runtime complexity.

In the past few decades, numerous rule learners have been developed, such as the famous *AQ* family, *CN2* and *RIPPER* [5]. Most of them follow a separate-and-conquer approach, which originated from *AQ* and still enjoys popularity. The approach searches for a rule that covers a part of the given (positive) examples, removes them, and recursively conquers the remaining examples by learning more rules until no examples remain. Most rule learners minimize model complexity assuming the validity of Occam's Razor.

Previous work related to converse k-clustering and frequent pattern summarization will be discussed in Section 4.

3. THE MINIMUM CONSISTENT SUBSET COVER PROBLEM

In this section, we introduce the Minimum Consistent Subset Cover (MCSC) problem and study its properties, which provide important insights for our algorithm design.

3.1 Definition

The MCSC problem finds the minimum number of consistent subsets that cover a given set of elements, where a subset is consistent if it satisfies a given constraint.

Definition 1. (Minimum Consistent Subset Cover) Given a finite ground set X and a constraint t, find a collection C of consistent subsets of X with $\bigcup_{S \in C} S = X$ such that $|C|$ is minimized, where a subset is consistent if it satisfies t.

We use a tuple (X, t) to denote an MCSC instance with ground set X and constraint t. The *consistent subset cover number* for (X, t), denoted by $\gamma(X, t)$, is the minimum number of consistent subsets with respect to t that cover X.

Given (X, t), we say the constraint t is *granular* if $\{x\}$ is consistent with respect to t for any $x \in X$. Apparently, with a granular constraint, (X, t) always has a non-empty feasible region. Most reasonable MCSC formulations feature granular constraints. For example, in the MCP problem, a single vertex also forms a clique.

The set covering problem can be considered as an MCSC instance where a subset is consistent if it is given. Consistent subsets are not explicitly given in typical MCSC instances. If we take a pre-processing step to generate all the consistent subsets, then an MCSC instance becomes a set covering instance. Unfortunately, as argued in [7], the generated collection of subsets would be prohibitively large and this is not a feasible approach to solve MCSC instances.

3.2 Properties

Covering problems have corresponding partitioning problems as special cases. Disallowing overlapping, partitioning problems are easier in the sense that they have smaller search spaces. Algorithms for partitioning problems usually work for the associated covering problems as well but typically generate solutions of larger sizes. However, finding partitions can be an advantageous way of finding covers for MCSC instances under certain conditions.

Definition 2. (anti-monotonic constraint) Given (X, t), we say t is anti-monotonic if for any subset $S \subseteq X$ that is consistent, any $S' \subseteq S$ is also consistent.

For example, the MCP problem has an anti-monotonic constraint since a subset of a clique is still a clique. As to be shown, many practical data mining problems are also MCSC instances with anti-monotonic constraints, such as the Minimum Rule Set and converse pairwise clustering problems.

Theorem 1. Given (X, t) where t is anti-monotonic, any solution to (X, t) can be transformed into a solution to the associated partitioning problem with the same cardinality.

PROOF. We give a constructive proof of the theorem. Suppose we have a solution to (X, t) at hand which is a set of overlapping consistent subsets. For each pair of consistent subsets that overlap, we can simply assign the overlap to any of the two and remove it from the other. Since t is anti-monotonic, the consistent subset with the overlap removed remains consistent. Then, we obtain a set of consistent subsets of the same cardinality that form a partition of X. $\square$

Theorem 1 implies that for (X, t) where t is anti-monotonic, optimal or good partitions are also optimal or good covers. By targeting the simpler partitioning problem, we may design well-guided yet efficient search heuristics for (X, t).

In the following, we define the so-called *consistency graph* for (X, t), based on which we connect the MCSC problem to the MCP problem and derive lower-bounds for $\gamma(X, t)$.

Definition 3. (consistency graph) Given (X, t), a consistency graph for (X, t), $G_c = (V_c, E_c)$, is a simple graph where $V_c = X$, and there is an edge $(u, v) \in E_c$ for a pair of vertices $u, v \in V_c$ if and only if $\{u, v\}$ is consistent.

Observation 2. Given (X, t) where t is anti-monotonic, a consistent subset forms a clique in G_c, where G_c is the consistency graph for (X, t).

The observation is rather straightforward. Since t is anti-monotonic, any pair of elements in a consistent subset must also constitute a consistent subset and the two corresponding vertices in G_c will share an edge connection.

The observation implies that a feasible solution to (X, t) is also a feasible solution to the MCP instance on G_c, thus the feasible region for (X, t) is a subset of the feasible region for the MCP instance, which implies that an optimal solution to the MCP instance must be an optimal solution to (X, t). Therefore, the consistent subset cover number $\gamma(X, t)$ is lower-bounded by the clique partition number $\overline{\chi}(G_c)$, which is further lower-bounded by the independence number $\alpha(G_c)$ as we have discussed in the preliminaries.

Theorem 2. $\alpha(G_c) \leq \overline{\chi}(G_c) \leq \gamma(X, t)$, where G_c is the consistency graph for (X, t) and t is anti-monotonic.

Based on Theorem 2, since $\alpha(G_c)$ is the size of a maximum independent set in G_c, the size of any independent set must be less than or equal to any feasible solution to the MCSC (or MCP) problem. If the two are equal, then the solution is optimal for both. This implication has been used to confirm the optimality of some of our experimental results.

3.3 Extensions

In the following, we relax the conditions specified in the properties established above so that they can be extended to more applications. For example, converse k-center does not come with an anti-monotonic constraint. The removal of cluster centers may corrupt the consistency of clusters.

Definition 4. (pivot-anti-monotonic constraint) Given (X, t), we say t is pivot-anti-monotonic if for any subset $S \subseteq X$ that is consistent, there exists a *pivot* element $p \in S$ such that $S' \cup \{p\}$ is consistent for any $S' \subseteq S$.

Example 1. Let us consider (X, t) for the converse k-center problem, where t requires each cluster to have a radius no larger than a given threshold. t is pivot-anti-monotonic with cluster centers as pivots. As long as the center remains, any sub-cluster of a consistent cluster remains consistent.

Obviously, if t is anti-monotonic, t must be pivot-anti-monotonic as well. A consistent subset could have multiple pivots. A pivot of a consistent subset may be present in another as a non-pivot element. The concept of pivot can be extended from a single element to a set of elements, however, we keep the case simple in this study.

Theorem 3. Given (X, t) where t is pivot-anti-monotonic, and given that $S_1 \cup S_2$ is consistent if S_1 and S_2 are consistent subsets sharing the same pivot, any solution to (X, t) can be transformed into a solution to the associated partitioning problem with the same or smaller cardinality.

PROOF. We give a constructive proof of the theorem. Suppose we have a solution to (X, t) at hand which is a set of overlapping consistent subsets. We first merge all the consistent subsets sharing the same pivot resulting in a set of consistent subsets such that no two of them share the same pivot. Now, even if a pivot may still appear in an overlap of two consistent subsets, it is a pivot for one of them but not both. We just need to make sure the pivot is assigned to the one that needs it. We assign all the non-pivot elements in an overlap to either of the two overlapping consistent subsets and remove them from the other. Since t is pivot-anti-monotonic, the consistent subset with the non-pivot elements removed remains consistent. Then, we get a set of consistent subsets of (X, t) of the same or smaller cardinality that form a partition of X. $\square$

We also define the so-called *pseudo consistency graph* for (X,t), based on which a similar observation as in Observation 2 and a similar conclusion as in Theorem 2 can be obtained by following similar arguments.

Definition 5. (pseudo consistency graph) Given (X,t), a pseudo consistency graph for (X,t), $G'_c = (V'_c, E'_c)$, is a simple graph where $V'_c = X$, and there is an edge $(u,v) \in E'_c$ for a pair of vertices $u, v \in V'_c$ if and only if there exists $p \in X$ such that $\{u, v, p\}$ is consistent.

Observation 3. Given (X,t) where t is pivot-anti-monotonic, a consistent subset forms a clique in G'_c, where G'_c is the pseudo consistency graph for (X,t).

Theorem 4. $\alpha(G'_c) \le \overline{\chi}(G'_c) \le \gamma(X,t)$, where G'_c is the pseudo consistency graph for (X,t) and t is pivot-anti-monotonic.

In this paper we focus on MCSC instances with anti-monotonic (or pivot-anti-monotonic) constraints. The theoretical results established above as well as our proposed algorithm *CAG* can be applied to such instances. In the following, we introduce several practical data mining problems that have MCSC formulations of this kind.

4. DATA MINING APPLICATIONS

In this section, we discuss several practical data mining applications that can be formulated as Minimum Consistent Subset Cover (MCSC) instances, in particular, the Minimum Rule Set (MRS) problem for rule learning, converse k-clustering, and frequent pattern summarization.

4.1 The Minimum Rule Set Problem

The MRS problem finds a disjunctive set of if-then rules with the minimum cardinality that cover a given set of labeled examples completely and consistently. A rule covers an example if the attribute values of the example satisfy the conditions specified in the antecedent (if-part) of the rule.

Definition 6. (Minimum Rule Set) Given a set X of labeled examples of multiple classes, find a complete and consistent set R of propositional rules for X, i.e., for each $e \in X$, there exists some $r \in RS$ that covers e and for each $r \in RS$, all examples covered by r must have the same class label, such that $|RS|$ is minimized.

As the most human-comprehensible classification tool, decision rules play a unique role in both research and application domains. In addition to data classification, rules can also be used for the purpose of data description as decision trees [17]. Data *description* focuses on existing data instead of unseen data as in classification, seeking to reduce the volume of data by transforming it into a more compact and interpretable form while preserving accuracy. For both applications, simpler models are preferred. By the widely applied principle of Occam's Razor, simpler models tend to generalize better to unseen data. From the understandability point of view, simpler models provide more compact and concise descriptions that are easier to comprehend.

Decision trees can be used to extract a set of mutually exclusive rules [20]. The optimal decision tree problem has been studied to induct a perfect tree that correctly classifies all the given examples with some objective optimized (e.g., [16]). While various objectives have been investigated, a common one is to minimize tree complexity, which can be measured by the number of leaf nodes. The problem we study, MRS, can be accordingly referred to as an optimal rule set problem with an objective of the same kind.

Another popular measure for tree complexity is the total number of tests (internal nodes), which corresponds to the total number of conditions in a rule set. The two measures tend to agree to each other. In the rule set case, fewer number of rules usually lead to fewer number of conditions, as demonstrated in our experimental study.

Most rule learners, e.g., the *AQ* family, *CN2* and *RIPPER* [5], also implicitly reduce the complexity of rule sets in order to achieve good generalization accuracy. However, as pointed out by [11], the minimality of rule sets has not been a "seriously enforced bias", which is also evident in our experimental comparison study.

The MCSC formulation. The MRS problem can be formulated as an MCSC instance (X,t), where X is the given example set and t requires a subset $S \subseteq X$ to form a consistent rule. In particular, S is consistent if its bounding box contains no examples of other classes. Since any single example forms a consistent rule, t is granular and (X,t) has a non-empty feasible region. In addition, any rule that covers a subset of the examples covered by a consistent rule is also consistent, thus t is also anti-monotonic, and the results in Theorem 1 and Theorem 2 are directly applicable. Recall that Theorem 2 gives $\alpha(G_c) \le \overline{\chi}(G_c) \le \gamma(X,t)$. In the following, we present an additional MRS-specific result.

Lemma 1. Given (X,t) for the MRS problem, where X is in d-dimensional space with $d \le 2$, a clique in G_c forms a consistent subset of (X,t).

To prove the lemma, it suffices to prove the combination of the bounding boxes of all pairs of vertices in a clique covers the bounding box for the clique, which can be proved by induction on the number of vertices. Due to the page limit, we omit the full proof, which can be found in [6]. Based on Lemma 1, the following theorem immediately follows.

Theorem 5. Given (X,t) for the MRS problem, where X is in d-dimensional space with $d \le 2$, $\overline{\chi}(G_c) = \gamma(X,t)$.

Lemma 1 and Theorem 5 do not hold for $d > 2$. We construct a counter example for $d = 3$. Let $X = \{a, b, c, e\}$ where $a = (2,4,5)$, $b = (4,3,2)$, $c = (7,9,4)$, and $e = (3,5,3)$. Let e be the only negative example. Since e is not covered by the bounding box of any pair of examples in $S = \{a, b, c\}$, S is a clique in G_c. However, e is covered by the bounding box of S, thus S is not consistent.

Approximate models are useful for both data classification (to avoid overfitting) and data description (to improve interpretability) applications of decision rules. For efficiency, it is desirable for rule learners to generate approximate models during the induction process, instead of post-processing, while keeping its ability to generate perfect models. The MCSC formulation of the MRS problem provides this flexibility, where we can simply adjust t and allow each rule to contain a maximum number of examples of other classes. This constraint is anti-monotonic.

The *RGB* rule learner. By solving (X,t), we obtain a solution to the MRS problem containing a set of bounding boxes. The bounding boxes, which we call *rectangles* in the following, are a special type of rules with all the attributional conditions specified. To learn a set of compact rules (of the same cardinality) for the MRS problem, we propose the

Algorithm 1 *RGB*

```
Input: $X$, $t$, and $b$: $X$ is the example set, a set of multi-class examples. $t$ is a constraint requiring rules to be consistent. $b$ is a user-specified beam search width.
Output: $R$: a compact rule set with redundancy removed.
 1: $R \leftarrow CAG(X, t)$; //$R$ stores a set of rectangle rules
 2: for each $r \in R$
 3:     initialize $b$ empty rules $r'_1, r'_2, ..., r'_b$;
 4:     initialize $b$ sets of examples $E_1, E_2, ..., E_b$;
 5:     while $(E_1 \neq \emptyset \wedge E_2 \neq \emptyset \ ... \wedge E_b \neq \emptyset)$
 6:         for each $r'_i$, choose the top $b$ conditions from $r$, each being added to $r'_i$, to form $b$ candidates that eliminate the most examples from $E_i$;
 7:         choose the top $b$ rules from the $b \times b$ candidates as $r'_1, r'_2, ..., r'_b$ and update $E_1, E_2, ..., E_b$ accordingly;
 8:     end while
 9:     $r \leftarrow r'_j$ suppose $E_j = \emptyset$ caused the loop to terminate;
10: end for
```

rectangle-based and graph-based algorithm *RGB* that takes two steps. First, we focus on rectangle rules only and solve the formulated MCSC instance. Then, we remove redundant conditions in the rectangle rules. A condition is considered *redundant* for a rule if its removal will not cause the inclusion of any new examples of other classes or the exclusion of any examples of the same class previously covered by the rule.

As shown in Algorithm 1, *RGB* first calls *CAG* (fully explained in Section 5) for the MCSC instance (X, t) and stores the set of learned rectangle rules in R. Then for each $r \in R$, a general-to-specific beam search is performed to remove redundant conditions. To explain the idea, we consider a beam search of width 1. We initialize and maintain a set of examples of other classes, the *elimination set* for r, that need to be eliminated to achieve consistency. The search starts with an empty rule, i.e., *true*, the most general rule, and the conditions to add are chosen from the original conditions in r. For the choice of condition to add, a greedy approach is adopted favoring the condition excluding the largest number of examples from the elimination set of r. The search stops when the elimination set is empty.

Since the conditions to add are chosen from the original conditions in the input rules, the extracted rules are more general than their original ones. Since the consistency of each rule is also guaranteed, the resulting compact rule set R is complete and consistent. The runtime of *RGB* is dominated by *CAG*, which we will discuss in Section 5.

As a rule learner, *RGB* has many unique and/or desirable properties. Unlike most existing methods, *RGB* does not follow a separate-and-conquer approach, instead, all rules are learned simultaneously. Unlike most existing bottom-up methods that start the initial rule set with the example set, *RGB* starts with a subset containing a maximal number of examples such that no pair of them can co-exist in a single consistent rule, which constitutes a maximal optimal partial solution. Unlike many existing methods that learn a set of rules for one class at a time, *RGB* naturally learns a set of rules for all classes simultaneously. Unlike many existing methods that can only learn either perfect models such as early members of the *AQ* family, or approximate models such as *CN2* and *RIPPER*, *RGB* has the flexibility to learn both without resorting to post-processing.

4.2 Converse *k*-clustering

k-clustering methods generate k clusters that optimize a certain compactness measure, typically distance-based, that varies among different clustering models. While some measures use the sum or average of (squared) distances as in k-means and k-medoid [14], some measures use a single distance value, radius or diameter, as in k-center [21] and pairwise clustering [4]. The *radius* of a cluster is the maximum distance between a fixed point (center) and any point in the cluster, and the *diameter* is the maximum distance between any two points in the cluster.

A known limitation of k-clustering is that the appropriate number of clusters k is often hard to specify in advance, and methods that try to automatically determine this number have been investigated [19]. As another approach to address this limitation, alternative clustering models have been explored. In *converse k-clustering*, a compactness measure is given as threshold, and the task is to minimize the number of clusters k. Converse k-clustering models have not received much attention in the data mining community. However, in many applications a distance-based constraint is easier to provide, based on domain knowledge, than the number of clusters. For example, molecular biologists have the knowledge that how similar a pair of sequences should be so that the two proteins can be assumed sharing the same functionality with high probability, or businessmen have the knowledge that how similar two customers should be so that they would have similar purchasing behaviors. The facility location problem [15], extensively studied in the operations research community, has the general goal of minimizing the total cost of serving all the customers by facilities. One of the problem formulations minimizes the number of located facilities based on the pairwise distance knowledge of customers, which is precisely a converse k-clustering model.

The converse k-center and converse pairwise clustering models can both be formulated as MCSC instances. In such an instance (X, t), X is the input data points for clustering and t requires a cluster to satisfy the maximum radius or diameter threshold constraint. As single distance measures, radius and diameter are more intuitive for domain experts to specify constraints than the more complex ones.

Both MCSC instances have granular constraints since singleton clusters satisfy any distance threshold. As explained in Example 1, converse k-center has a pivot-anti-monotonic constraint with cluster centers as pivots. For converse pairwise clustering, the constraint is anti-monotonic since any sub-cluster of a cluster can only have an equal or smaller diameter than the cluster. The results established in Section 3 apply to the two MCSC instances. Note that in converse k-center, a union of two consistent clusters sharing the same center is also consistent, thus the additional condition required by Theorem 3 is satisfied and the theorem applies. In addition, we have the following observation.

Observation 4. Given (X, t) for the converse pairwise clustering problem, we have $\overline{\chi}(G_c) = \gamma(X, t)$.

Based on Observation 2, a consistent subset of (X, t) forms a clique in the consistency graph G_c. On the other hand, a clique in G_c must form a consistent subset, thus converse pairwise clustering is equivalent to the MCP problem on G_c. The same observation does not hold for the converse k-center problem.

4.3 Pattern Summarization

Frequent pattern mining has been studied extensively for various kinds of patterns including itemsets, sequences, and graphs. While great progress has been made in terms of efficiency improvements, interpretability of the results has become a bottleneck to successful application of pattern mining due to the huge number of patterns typically generated. A closely related problem is that there is a lot of redundancy among the generated patterns. Interpretable and representative summarization of large collections of patterns has emerged as an important research direction.

As a first approach, maximal frequent patterns [9] and closed frequent patterns [18] have been introduced. These subsets of frequent patterns are more concise and allow to derive all the frequent patterns. However, the patterns thus generated are still too many to handle. As an alternative, [10] finds the top-k frequent closed patterns of length no less than a given threshold. While this approach effectively limits the output size, the returned patterns are often not representative of the entire collection of frequent patterns. Moreover, these approaches fail to address the redundancy problem.

Some recent work aims at finding a fixed number of patterns representing the whole set of frequent patterns as well as possible. In [1], the objective is to maximize the size of the part of the input collection covered by the selected k sets. [23] presents an approach that takes into account not only the similarity between frequent itemsets, but also between their supports. Using a similarity measure based on Kullback-Leibler divergence, they group highly correlated patterns together into k groups.

Similar to the scenario for k-clustering, the appropriate number k of patterns to summarize the set of frequent patterns is often hard to specify in advance. However, the users may have the domain knowledge that how similar a group of patterns should be so that they can be represented as a whole without losing too much information. In light of this, some converse k-clustering models that can be formulated as MCSC instances, converse k-center or converse pairwise clustering, appear to be very reasonable and promising to provide concise and informative pattern summarizations. Such clustering models generate clusters, i.e., groups of patterns, with certain quality guarantee. In such a formulation, the objective is to minimize the number of pattern groups necessary to cover the entire collection of frequent patterns, which is natural for the purpose of summarization since it maximizes interpretability of the result representing the entire collection and at the same time reduces redundancy. For the radius or diameter threshold, standard distance functions can be employed, such as the Jaccard's coefficient for itemsets or edit distance for sequential patterns.

5. THE GENERIC *CAG* ALGORITHM

In this section, we introduce a generic algorithm CAG that works with consistency graphs and assignment graphs to solve Minimum Consistent Subset Cover (MCSC) instances featuring anti-monotonic constraints.

5.1 Overview

Given an MCSC instance (X, t) where the constraint t is anti-monotonic, CAG starts by constructing a maximal optimal partial solution from the consistency graph G_c, then performs an example-driven specific-to-general search on a dynamically maintained bipartite assignment graph G_a to learn a set of consistent subsets with small cardinality.

The design of CAG has closely followed the insights provided in Section 3. From the definition of G_c and Theorem 2, we know that any pair of vertices in an independent set of G_c cannot appear in the same consistent subset. Thus a maximal independent set of G_c, denoted by IS, constitutes a maximal optimal partial solution to (X, t).

Each vertex in IS forms a singleton consistent subset, represented by a so-called *condensed vertex* in an assignment graph G_a. The rest of the vertices in G_a are called *element vertices.* G_a is defined such that there is an edge between an element vertex u and a condensed vertex v if and only if u can be assigned to v while maintaining the consistency of v. Since a maximal independent set is also a dominating set as discussed in the preliminaries of Section 2, there is an edge for each u connecting to some v in the initial G_a.

Element vertices are processed in a sequential manner and assigned to condensed vertices. With the growth of condensed vertices, some element vertices would get isolated and new condensed vertices have to be created for them. Upon completion, G_a becomes edge-free with all vertices assigned, and the set of condensed vertices, each representing a consistent subset, constitute a solution for (X, t).

The dynamically maintained G_a provides the necessary information based on which effective evaluation measures can be designed to guide the search by deciding which element vertex is the next to be assigned to which condensed vertex. For example, with the *least degree first* criterion, we can process first the element vertex with the least degree since it is the most likely one to get isolated.

Note that CAG actually returns a partition of X. As argued in Section 3, a solution to a partitioning problem is also a feasible solution to its corresponding covering problem. Also due to Theorem 1 and Theorem 3, partitioning does not cause the increase of solution size compared to covering under certain conditions. In addition, partitioning problems have much smaller search spaces and the search can be better guided.

Also note that under anti-monotonic constraints, if a subset $S \subseteq X$ is not consistent, $S' \supseteq S$ cannot be consistent. Thus such inconsistent S does not need to be considered and the search space can be significantly pruned. Our assignment graph G_a maintains consistency of all condensed vertices after each assignment transaction allowing CAG to work in a pruned search space.

In contrast to the most-seen separate-and-conquer approach, e.g., the greedy algorithm [12] for set covering and most existing rule learners, CAG adopts a less greedy example-driven strategy to learn consistent subsets simultaneously. In summary, the construction of initial optimal partial solution, the use of assignment graph, and the example-driven simultaneous learning strategy are the three novel design ideas that account for the good performance of CAG.

5.2 Assignment graph

In the following, we define and explain assignment graphs, a unique element of CAG that helps to guide the search.

Definition 7. (assignment graph) In a bipartite assignment graph $G_a = (U_a \cup V_a, E_a)$, U_a contains a set of element vertices and V_a contains a set of condensed vertices each representing a consistent subset. $(u, v) \in E_a$ if and only if $u \in U_a$, $v \in V_a$, and $v \cup \{u\}$ is consistent.

In *CAG*, G_a is dynamically maintained showing all the feasible choices for the next assignment. Each condensed vertex is essentially a set of vertices condensed together. In order to maintain the consistency of subsets, an element vertex u can be assigned to some condensed vertex v only via an edge connection between them. However, each assignment may cause some edges to disappear in G_a. For those isolated element vertices (with degree of 0), new condensed vertices have to be created. Each creation may introduce some new edges to G_a.

Figure 1 (c) shows an initial G_a and (d)-(f) show its dynamic evolvement along the assignment process, for which we will provide further explanations later in this section.

Observation 5. While a vertex assignment may cause multiple edge gains or losses in G_a, it can cause at most one edge gain or loss for any single element vertex.

The observation holds because all the gained or lost edges have to be incident to the condensed vertex involved in an assignment, and any element vertex shares at most one edge with a condensed vertex. The implication of the observation is that if an element vertex has a degree of more than two, it will not become isolated after the next assignment.

Evaluation measures. In the following, we show how the information embedded in G_a can help to guide the search by answering important questions in the assignment process: Which element vertex is the next to be assigned? Which condensed vertex should it be assigned to? In principle, the element vertex with the least degree should be considered first since it is most likely to get isolated. Also based on Observation 5, element vertices with degree more than two will stay "safe" after such an assignment. Thus, a *least degree first* criterion can be used to choose the next element vertex to be assigned.

In addition, we want to keep as many edges as possible in G_a since they are the relational resources for the assignment of element vertices. Edges have different degrees of importance. In general, a gained or lost edge is more important if it is incident to an element vertex with a smaller degree in G_a. We design a measure, *weighted edge gain*, that reflects the influence of an assignment transaction on G_a.

In the following, we define $weg(uu)$ and $weg(uv)$. $weg(uu)$ is the weighted number of edges that can be added to E_a if u is chosen to become a new condensed vertex. In such cases, u is assigned to itself, which happens only when $d^\circ(u) = 0$, where $d^\circ(u)$ denotes the degree of u in G_a. $weg(uv)$ is the weighted number of edges that would get lost (an edge loss is just a negative gain) if u is assigned to v via edge $(u, v) \in E_a$. For $weg(uu)$, the weight for each possibly added edge is $\frac{1}{d^\circ(u')+1}$ where we use $d^\circ(u') + 1$ because $d^\circ(u')$ can be 0 and for that case, the weight should be the largest, i.e., 1. For both measures, different weighting schemes can be used.

$$weg(uu) = \sum_{\forall\, u' \in U_a\ s.t.\ \{u',u\}\ is\ consistent} \frac{1}{d^\circ(u')+1}$$

$$weg(uv) = \sum_{\forall\, u'\ s.t.\ (u',v) \in E_a\ \wedge\ v \cup \{u',u\}\ is\ inconsistent} \frac{-1}{d^\circ(u')}$$

Once u, the next element vertex to be assigned, is chosen, a *largest* $weg(uv)$ *first* criterion can be used to decide which condensed vertex v that should take u so as to keep as many weighted edges as possible.

We also define the following measure $weg(u)$ that measures the influence on G_a if u is chosen as the next element vertex to be assigned. The associated *largest weg(u) first* criterion can be used alone for the choice of u. It can also be used as a tie-breaking mechanism for the least degree first criterion in choosing u.

$$weg(u) = \begin{cases} weg(uu) & \text{if } d^\circ(u) = 0 \\ weg(uv)\ where\ v = \\ argmax_{\forall\, v'\ s.t.\ (u,v') \in E_a}\{weg(uv')\} & \text{otherwise} \end{cases}$$

In the formula, for $d^\circ(u) = 0$, $weg(u) = weg(uu)$. For other cases, $weg(u)$ is the largest $weg(uv)$ value considering all the condensed vertices adjacent to u in G_a, indicating the best assignment choice for u.

5.3 Generic *CAG*

We have explained the working principles of *CAG* in the overview. In the following we present the algorithm and explain it in more details.

Algorithm 2 generic *CAG*

```
Input: $(X, t)$: an MCSC instance.
Output: $V_a$: a set of condensed vertices representing a set
    of consistent subsets that cover $X$.
 1: initialize $G_a = (U_a \cup V_a, E_a)$;
 2: while $(U_a \neq \emptyset)$
 3:   choose $u \in U_a$;
 4:   if $(d^\circ(u) = 0)$ then
 5:     $V_a \leftarrow V_a \cup \{u\}$;
 6:     $U_a \leftarrow U_a \backslash \{u\}$;
 7:     $E_a \leftarrow E_a \cup \{(u, u')\}$ for each $u'$ with $u' \in U_a$ and
        $\{u, u'\}$ is consistent;
 8:   else
 9:     choose $v \in V_a$;
10:     $v \leftarrow v \cup \{u\}$;
11:     $U_a \leftarrow U_a \backslash \{u\}$;
12:     $E_a \leftarrow E_a \backslash \{(u, u')\}$ for each $u'$ with $(u', v) \in E_a$
        and $v \cup \{u, u'\}$ is inconsistent;
13:   end if
14: end while
```

CAG starts by initializing the assignment graph G_a using a maximal independent set of G_c (line 1). Then, the element vertices are processed in a sequential manner (line 2). For each assignment, a decision is made on which $u \in U_a$ is the next element vertex to be assigned (line 3). After u is chosen, if it has a degree of 0 (line 4), a new condensed vertex has to be created for u (line 5) and u will be removed from U_a (line 6). At this moment, E_a needs to be updated (line 7). Some element vertices (that are connected to u in E_c) may be able to connect to the newly added condensed vertex u resulting in creation of some new edges. If u is not an isolated vertex (line 8), another decision is to be made on which $v \in V_a$ should take u (line 9). After the assignment of u to v (lines 10, 11), E_a also needs to be updated (line 12). Because of the assignment of u to v, some element vertices, say u', that previously connected to v would lose the connection if $v \cup \{u, u'\}$ is inconsistent. Upon completion, $U_a = \emptyset$ and $E_a = \emptyset$. The set of condensed vertices V_a corresponds to a set of consistent subsets together covering X.

Initializing G_a**.** One way of initializing G_a is to derive the consistency graph G_c for (X, t) first then find a maximal independent set IS of G_c using some known heuristic, e.g., *Greedy-IS* introduced in [13]. With IS obtained, we let $V_a = IS$, $U_a = V_c \backslash IS$, and $(u,v) \in E_a$ if and only if $u \in U_a$, $v \in V_a$ and $(u,v) \in E_c$.

In some cases, e.g, for the MCP problem, G_c is identical to the input graph. In other cases, G_c needs to be derived and this takes $O(|X|^2 t_p)$ time where t_p denotes the consistency checking time for a pair of vertices in X. This is because each pair in X needs to be checked for consistency. To improve efficiency, we provide another way of initializing G_a without actually deriving G_c. Initially, V_a and E_a are empty and $U_a = X$. For each $u \in U_a$, we check each $v \in V_a$ and if $v \cup \{u\}$ is consistent, we add edge (u,v) into E_a. If no edge can be added, we add u into V_a. Upon completion, V_a also contains a maximal independent set IS of G_c. This procedure takes $O(|IS||X|t_p)$. Note that usually $|IS| \ll |X|$.

Choosing u and v. By varying the decision making schemes, i.e., how to choose u (line 3) and v (line 9), the generic CAG can have different instantiations that are suitable for different applications. By default, we use the least degree first criterion to choose u with the largest $weg(u)$ first criterion for tie-breaking. Also, we use the largest $weg(uv)$ first criterion to choose v.

With the default scheme, the time spent in choosing u would be on calculating $weg(u)$ for tie-breaking. As an approximation, we adopt a sampling technique. We only consider a constant number of element vertices with the tied least degree and for each of them, say u, we only consider a constant number of element vertices for consistency checking, each of which, say u', connects to some condensed vertex v that is adjacent to u in G_a. This sampling technique works well because the least degree first criterion greatly reduces the number of vertices in consideration in calculating $weg(u)$. Therefore, the runtime for the choice of u is $O(t_c)$, where t_c is the consistency checking time for $v \cup \{u, u'\}$ given that $v \cup \{u\}$ and $v \cup \{u'\}$ are consistent.

For the choice of v, the calculation of $weg(uv)$ is part of the calculation of $weg(u)$ and thus it takes $O(t_c)$ as well.

Updating G_a. By varying the G_a updating mechanisms (lines 7 and 12), the generic CAG can be adapted to different applications. These applications have different constraints requiring different consistency checking mechanisms.

In line 7, the edge set E_a of G_a is updated after an isolated element vertex u becomes a new condensed vertex. This may introduce some new edges to E_a. Note that line 7 will be executed no more than $|V_a| - |IS|$ times where V_a here represents its final stage after CAG terminates, thus the total time spent on line 7 is at most $O((|V_a| - |IS|)|X|t_p)$.

In line 12, the edge set E_a of G_a is updated after an element vertex u is assigned to a condensed vertex v. This may cause the loss of some edges incident to v. Each such update could take a worst case runtime of $O(|X|t_c)$. To reduce the runtime, we adopt a *late update* strategy that significantly improves efficiency without sacrificing much performance. This is based on the observation that the edge update operations do not have an impact on the assignment process if the corresponding element vertices are not considered in the next assignment. Given the least degree first criterion, we only need to perform edge update for those element vertices with the least (or small) degrees since they are likely to be considered in the next assignment. This late update strategy would leave some extra edges in G_a that should have been removed. However, these edges will be removed sooner or later when their corresponding member vertices get closer

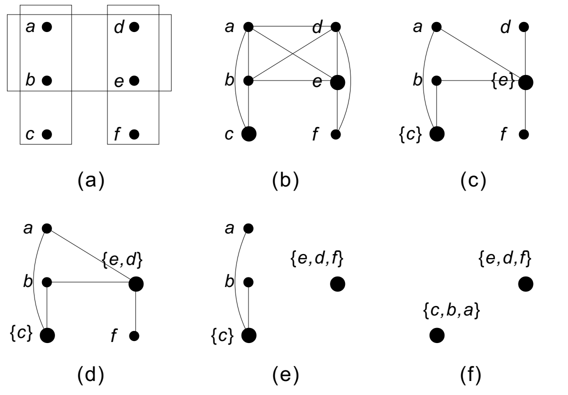

Figure 1: Running example.

and closer to be chosen along the assignment process. Therefore, by checking consistency for a constant number of vertices connected to v, we only spend $O(t_c)$ time for this update without sacrificing much performance.

Time complexity. As we have explained previously, initializing G_a (line 1) takes $O(|IS||X|t_p)$. The update of G_a for the $d^\circ(u) = 0$ case (line 7) takes in total $O((|V_a| - |IS|)|X|t_p)$ time. Together they take $O(|V_a||X|t_p)$ time.

In each iteration, the choice of u (line 3), the choice of v (line 9) and the update of G_a for the $d^\circ(u) \neq 0$ case each takes t_c time. There are $O(|X|)$ iterations and thus together they take $O(|X|t_c)$ time.

Overall, the worst case runtime is $O(|V_a||X|t_p + |X|t_c)$. In general, $|V_a| \ll |X|$ as can be observed from our experimental results in Table 1.

The consistency checking time t_p and t_c are different for different mechanisms and different applications. For the MCP problem and the converse pairwise clustering problem, both t_p and t_c are constant. Thus, the worst case runtime for the two problems is $O(|V_a||X|)$.

For the MRS problem, both t_p and t_c are $O(|X|)$ for the brute-force consistency checking mechanism, which can be improved to log time on average by employing indexing techniques. The worst case runtime for the MRS problem is $O(|V_a||X|^2)$. With the help of indexing, the average case runtime is $O(|V_a||X| \log |X|)$.

Running example. In Figure 1, we provide a running example demonstrating how CAG works on a set covering instance (X,t). By definition, (X,t) does not have an anti-monotonic constraint since we are allowed to use the given subsets, not their subsets, in covering X. However, if S' is a subset of some given subset $S \subseteq X$, S' can be considered "consistent" since it can always be replaced by S. Thus, our generic algorithm CAG is perfectly applicable.

In the figure, the (X,t) instance is given in (a). We can see that the greedy algorithm will select all the three given subsets. For clarity, we also show G_c for (X,t) in (b), where we can identify a maximal IS containing c and e. (c) is the initial G_a with c and e as the condensed vertices.

Now we start the assignment process. Both d and f have the least degree of 1, but assigning d to $\{e\}$ would not cause any edge loss, thus d is chosen and assigned to $\{e\}$, and (d) shows the updated G_a. Next, f is chosen and assigned to $\{e,d\}$, and (e) shows the updated G_a. We can see that G_a lost two edges since neither a nor b can join $\{e,d,f\}$ while maintaining its consistency, i.e., $\{e, d, f, a\}$ and $\{e, d, f, b\}$ are inconsistent. Afterwards, a and b are assigned to $\{c\}$ and (f) shows the final G_a, where both U_a and E_a are empty and V_a contains the condensed vertices corresponding to the consistent subsets that cover X.

Solving converse k-center. Given (X, t) for the converse k-center problem, we have explained previously that t is pivot-anti-monotonic. Also, (X, t) satisfies the additional condition specified in Theorem 3, that is, two consistent subsets of (X, t) sharing the same pivot can merge into a single consistent subset. Based on the theoretical study, we can use the following approach to solve (X, t). First, we apply CAG where we use the pseudo consistency graph G'_c instead of G_c to derive G_a. Then we assign the needed pivots to the learned condensed vertices to obtain consistent subsets. Then we can obtain a partition of X by following the constructive method described in the proof of Theorem 3.

6. EXPERIMENTAL EVALUATION

To experimentally evaluate the performance of CAG, we have considered two MCSC applications, the Minimum Rule Set (MRS) and Minimum Clique Partition (MCP) problems. For the MRS problem, we compared CAG with a popular rule learner *AQ21* [22], the latest release of the famous *AQ* family, on UCI datasets [2]. In this series of experiments, RGB achieved about 40% reduction in the number of rules and 30% reduction in the number of conditions.

Observation 2 and Theorem 2 show that the MCP problem is related to all the MCSC instances within the scope of this study. The converse pairwise clustering problem, which can also be applied to frequent pattern summarization, is equivalent to the MCP problem on the consistency graph. Since MCP is a dual problem of graph coloring, many well-known graph coloring heuristics and benchmarks set up an ideal platform to evaluate the performance of CAG. The comparison partner we chose, $DSATUR$ [3], is among the most popular ones with top performance. Experiments on over 100 DIMACS benchmarks [8] showed that CAG outperformed $DSATUR$ in general, and in particular for the 70 hard datasets, CAG used 5% fewer cliques on average.

6.1 *MRS* results

RGB is our proposed rule learner that first calls CAG to learn a set of rectangle rules and then extracts compact rules with redundant conditions removed. Our comparison partner, *AQ21* [22], is the latest release of the famous *AQ* family. Other popular rule learners such as *CN2* and *RIPPER* do not return perfect rule sets. Since most rule learners follow a separate-and-conquer approach originated from *AQ*, the performance of *AQ21* can well represent the performance of most rule learners.

Although RGB can work on both numerical and categorical data in principle, we have focused on numerical data in this implementation since rectangle rules provide good generalization on numerical attributes only. Thus, 21 numerical datasets without missing values were chosen from the UCI repository [2] for this series of experiments.

Table 1 presents the results. The columns in the table are *dataset*, *ins* (number of instances), *dim* (dimensionality), *cla* (number of classes), *AQ21* (*AQ21* results), RGB (RGB results), and *reduction* (reduction achieved by RGB compared to *AQ21*, where $reduction = \frac{AQ21\ result\ -\ RGB\ result}{AQ21\ result}$).

For the columns *AQ21* and RGB, *rul* and *con* indicate the total number of rules and conditions respectively. The *rul* and *con* reductions are given under the *reduction* column. In addition, the columns $|E_c|$, $|IS|$ and *cliques* indicate the edge set size, the maximal independent set size and the number of cliques of G_c discovered by RGB respectively.

From Table 1 we see that RGB achieved 37.1% and 30.8% averaged reductions over *AQ21* in terms of the total number of rules and conditions respectively. In the last row of the table, the reductions were calculated for the averaged number of rules and conditions, where we see that RGB achieved 44.2% and 51.4% reductions over *AQ21* in terms of the averaged number of rules and conditions respectively.

Consistent to Theorem 2, although many results are not optimal, each dataset in Table 1 exhibits the same trend, that is, $|IS| \leq$ *cliques* $\leq$ *RGB rul* $\leq$ *AQ21 rul.* As discussed in Section 3, Theorem 2 can help to confirm the optimality of MCSC results. In Table 1 we can see that the IS, *cliques* and RGB results for datasets balance-scale, glass, and iris are optimal. Also, the IS and *cliques* results for ionosphere, new-thyroid, sonar, vowel, waveform, and wine are optimal.

In addition, we studied the trend of *reduction* by varying data complexity measured by *ins*, $|E_c|$ and *dim*. While the detailed results can be found in [6], in summary, the reductions in *rul* and *con* both increase with the increase of data complexity, which also confirms that the two model complexity measures for rule sets, number of rules and number of conditions, are consistent with each other.

6.2 *MCP* results

The MCP problem is a dual problem of graph coloring. Algorithms for graph coloring (MCP) can be applied to complementary graphs to return solutions to the MCP (graph coloring) problem. In this series of experiments, we have used $DSATUR$ [3] as our comparison partner. $DSATUR$ is among the most popular construction heuristics for graph coloring with top performance reported in the literature.

The experiments were performed on DIMACS benchmarks [8], for some of which the optimal solutions are known. We divide the benchmarks into two categories: easy and hard. As the rule of thumb for the division, those datasets with optimal solutions relatively easy to obtain are considered easy; otherwise hard. We used 35 easy and 70 hard datasets for our experiments. Due to the page limit, we only provide a summary of the results in Table 2. From the table we can see that out of the 35 easy datasets, $DSATUR$ found 33 while CAG found all the 35 optimal solutions.

Table 2: MCP results

category	*# datasets*	$DSATUR$	CAG
easy	35	33 (optimal)	35 (optimal)
hard	70	41 (36 tied)	65 (36 tied)

The optimal solutions for many hard datasets are not known. Table 2 reports the winning times of the two methods. From the table we can see that out of the 70 hard datasets, $DSATUR$ won 41 times while CAG won 65 times where they tied for 36 datasets. To more accurately capture their performance difference, for each dataset, we calculated the reduction of cliques achieved by CAG using the formula $reduction = \frac{DSATUR\ result\ -\ CAG\ result}{DSATUR\ result}$. Then, we calculated the averaged reduction over the 70 hard datasets to obtain a value of 4.99%, meaning that CAG used about 5% fewer cliques than $DSATUR$ on average for each dataset.

Table 1: ***MRS* results**

dataset	*ins*	*dim*	*cla*	*AQ21*		$\lvert E_c \rvert$	$\lvert IS \rvert$	*cliques*	*RGB*		*reduction*	
				rul	*con*				*rul*	*con*	*rul*	*con*
balance-scale	625	4	3	154	577	31578	153	153	153	676	0.006	-0.172
bupa	345	6	2	66	344	22149	12	19	38	224	0.424	0.349
car	1728	6	4	66	343	386152	55	57	57	340	0.136	0.009
diabetes	768	8	2	112	744	142994	10	18	57	420	0.491	0.435
ecoli	336	7	8	36	128	13569	22	24	28	131	0.222	-0.023
glass	214	10	6	6	7	5921	6	6	6	9	0.000	-0.286
haberman	306	3	2	73	198	6425	48	52	55	214	0.247	-0.081
ionosphere	351	34	2	25	172	33058	4	4	11	57	0.560	0.669
iris	150	4	3	10	26	3562	7	7	7	19	0.300	0.269
letter	15000	16	26	1160	15701	3751964	137	221	552	6684	0.524	0.574
new-thyroid	215	5	3	10	33	12196	6	6	7	31	0.300	0.061
page-block	5473	10	5	112	687	11754498	28	37	60	369	0.464	0.463
satimage	4435	36	6	212	4379	1784464	17	20	99	1316	0.533	0.669
segment	2310	19	7	49	382	374323	13	15	27	183	0.449	0.521
sonar	208	60	2	11	186	10761	2	2	5	79	0.545	0.575
spambase	4601	57	2	118	2292	4686024	33	38	64	1286	0.458	0.439
vehicle	846	18	4	109	1133	80643	13	15	55	478	0.495	0.578
vowel	990	10	11	76	583	43164	20	20	47	367	0.382	0.370
waveform	5000	21	3	266	4989	4164798	6	6	107	2347	0.598	0.530
wine	356	13	3	7	29	5324	3	3	4	17	0.429	0.414
yeast	2968	8	10	250	1740	136416	66	117	194	1606	0.224	0.077
average				139.4	1651.1	1307142.0	31.5	40	77.8	802.5	**0.371**	**0.308**
reduction				0	0				**0.442**	**0.514**		

We did not compare the runtime partly because *CAG* and *DSATUR* work on different graphs, G and $\overline{G}$, and the complexities of the two graphs are opposite to each other. However, we note that while *DSATUR* has a cubic runtime [3], for the MCP problem, *CAG* has a subquadratic runtime of $O(|V_a||X|)$, where $|V_a| \ll |X|$ ($|V_a| = cliques$ and $|X| = ins$) as can be observed from Table 1.

7. CONCLUSION

In this paper, we introduced and studied the Minimum Consistent Subset Cover (MCSC) problem, which generalizes the set covering problem and has many practical data mining applications in the areas of rule learning, converse k-clustering, and frequent pattern summarization. For MCSC instances with anti-monotonic constraints, our generic algorithm *CAG* was provided in accordance with the theoretical insights featuring many novel design ideas, e.g., the construction of maximal optimal partial solution, the use of dynamically maintained bipartite assignment graph, and the example-driven simultaneous learning strategy. Our experimental evaluation on benchmark datasets justified the performance of *CAG*.

For future work, we would like to further explore and experimentally evaluate the MCSC applications on converse k-clustering and frequent pattern summarization. We are also interested in seeking other data mining applications that can be formulated as MCSC instances and solved by our proposed *CAG* search framework.

8. REFERENCES

[1] F. Afrati, A. Gionis, and H. Mannila. Approximating a collection of frequent sets. *SIGKDD'04*.

[2] C. Blake and C. Merz. UCI repository of machine learning databases. 1998.

[3] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[4] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *STOC'97*.

[5] J. Furnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[6] B. J. Gao. Hyper-rectangle-based discriminative data generalization and applications in data mining. *Ph.D. Thesis. Simon Fraser University*, 2006.

[7] B. J. Gao and M. Ester. Turning clusters into patterns: Rectangle-based discriminative data description. *ICDM'06*.

[8] D. B. Graphs. http://mat.gsia.cmu.edu/coloring02/index.html.

[9] D. Gunopulos, H. Mannila, R. Khardon, and H. Toivonen. Data mining, hypergraph tranversals, and machine learning. *PODS'97*.

[10] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. *ICDM'02*.

[11] S. J. Hong. R-MINI: An iterative approach for generating minimal rules from examples. *TKDE*, 9(5):709–717, 1997.

[12] D. Johnson. Approximation algorithms for combinatorial problems. *JCSS*, 9:256–278, 1974.

[13] D. Johnson. Worst-case behavior of graph-coloring algorithms. *SEI CGTC'74*.

[14] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[15] J. Krarup and P. Pruzan. The simple plant location problem: survey and synthesis. *EJOR*, 12:36–81, 1983.

[16] A. Kulkarni and L. Kanal. An optimization approach to hierarchical classifier design. *IJCPR'76*.

[17] S. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *DMKD*, 2:345–389, 1998.

[18] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. *ICDT'99*.

[19] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. *ICML'00*.

[20] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[21] C. Toregas, R. Swain, C. Revelle, and L. Bergman. The location of emergency service facilities. *Operations Research*, 19:1363–1373, 1971.

[22] J. Wojtusiak, R. Michalski, K. Kaufman, and J. Pietrzykowski. Multitype pattern discovery via AQ21: A brief description of the method and its novel features. *Technical Report, MLI 06-2, George Mason University*, 2006.

[23] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: A profile-based approach. *SIGKDD'05*.