Constrained Clustering Using SAT

Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni

University of Caen Basse-Normandie – GREYC (CNRS UMR 6072)
Campus II, Côte de Nacre, 14000 Caen - France
{firstname.lastname}@unicaen.fr

Abstract. Constrained clustering - finding clusters that satisfy user-specified constraints - aims at providing more relevant clusters by adding constraints enforcing required properties. Leveraging the recent progress in declarative and constraint-based pattern mining, we propose an effective constraint-clustering approach handling a large set of constraints which are described by a generic constraint-based language. Starting from an initial solution, queries can easily be refined in order to focus on more interesting clustering solutions. We show how each constraint (and query) is encoded in SAT and solved by taking benefit from several features of SAT solvers. Experiments performed using `MiniSat` on several datasets from the UCI repository show the feasibility and the advantages of our approach.

1 Introduction

Clustering is one of the core problems in data mining. Clustering aims at partitioning data into groups (clusters) so that transactions occurring in the same cluster are similar but different from those appearing in other clusters [12]. The usual clustering problem is designed to find clusterings satisfying a nearest representative property while constrained clustering [3,19] aims at obtaining more relevant clusters by adding constraints enforcing several properties expressing background information on the problem at hand. Constraints deal with various types: (1) data objects' relationships (e.g., a set of objects must be (or not) in a same cluster [20]), (2) the description of the clusters (e.g., a cluster must have a minimal or a maximal size [2]), (3) both objects and clusters (e.g., a given object must be in a given cluster), (4) the characteristics of the clustering (e.g., the number of clusters),... Traditional clustering algorithms do not provide effective mechanisms to make use of this information. The goal of this paper is to propose a generic approach to fill this gap.

Recently, several works have investigated relationships between data mining and constraint programming (CP) to revisit data mining tasks in a declarative and generic way [6,14,15]. The user models a problem and expresses his queries by specifying *what* constraints need to be satisfied. The process greatly facilitates the search of knowledge and models such as clustering. The approach is enforced by the use of a constraint-based language [17]: it is sufficient to change

the specification in term of constraints to address different pattern mining problems. In the spirit of this promising avenue, we propose an effective constrained clustering approach handling a large set of constraints.

The paper brings the following contributions. First, we use the declarative modeling principle of CP to define a constrained clustering approach taking into account a large set of constraints on objects, a description of the clusters and the clustering process itself. By nature, clustering proceeds by iteratively refining queries until a satisfactory solution is found. Our method integrates in a natural way this stepwise refinement process based on the queries in order to focus on more interesting clustering solutions. Contrary to very numerous clustering methods that use heuristics or greedy algorithms, our method is complete. Second, we define an efficient SAT encoding which integrates features of SAT solvers (e.g., binary clauses, unit propagation, sorting networks) to solve the queries. Finally, an experimental study using `MiniSat` shows the feasibility and the effectiveness of our method on several datasets from the UCI repository.

Section 2 provides the background on the constraint-based language. Section 3 describes our method on constrained clustering with examples of constraints coming from the background information of the problem at hand. Section 4 addresses the point of how queries and constraints of the language are encoded and solved with SAT. Section 5 shows the effectiveness of our approach through several experiments. Section 6 presents related work.

2 Background: Constraint-Based Language

The constraint programming methodology is by nature declarative. It explains why studying relationships between CP and data mining has received a considerable attention to go towards generic and declarative data mining methods [6,14,15]. This section sketches our constraint-based language that enables us to specify in term of constraints different pattern mining problems [17]. This language forms the first step of our constrained clustering method proposed in Section 3. In the remainder of this section, we only focus on primitives of the language that will be used in this paper.

Let $\mathcal{I}$ be a set of n distinct literals called items, an itemset (or *pattern*) is a non-null subset of $\mathcal{I}$. The language of itemsets corresponds to $\mathcal{L}_{\mathcal{I}} = 2^{\mathcal{I}} \backslash \emptyset$. A *transactional dataset* $\mathcal{T}$ is a multi-set of m itemsets of $\mathcal{L}_{\mathcal{I}}$. Each itemset, usually called a *transaction* or *object*, is a database entry. For instance, Table 1 gives a transactional dataset $\mathcal{T}$ with m=11 transactions $t_1, \ldots, t_{11}$ described by n=10 items. This toy dataset is inspired by the `Zoo` dataset from the UCI repository.

Terms are built from constants, variables, operators, and function symbols. Constants are either numerical values, or patterns, or transactions. Variables, noted X_j, for $1 \leq j \leq k$, represent the unknown patterns (or clusters). Operators can be set ones (as $\cap, \cup, \backslash$) or numerical ones (as $+, -, \times, /$). Built-in function symbols involve one or several terms:

- $\texttt{cover}(X_j) = \{t \mid t \in \mathcal{T}, X_j \subseteq t\}$ set of transactions covered by X_j.
- $\texttt{freq}(X_j) = \mid \{t \mid t \in \mathcal{T}, X_j \subseteq t\} \mid$ is the frequency of pattern X_j.

- $\texttt{size}(X_j) = \mid \{i \mid i \in \mathcal{I}, i \in X_j\} \mid$ is the size of pattern X_j.
- $\texttt{overlapItems}(X_i, X_j) = \mid X_i \cap X_j \mid$ is the number of items shared by both X_i and X_j.
- $\texttt{overlapTransactions}(X_i, X_j) = \mid \texttt{cover}(X_i) \cap \texttt{cover}(X_j) \mid$ is the number of transactions covered by both X_i and X_j.

Constraints are relations over terms that can be satisfied or not. There are three kinds of built-in constraints: numerical constraints (like $<$, $\leq$, $=$, $\neq$, $\geq$, $>$), set constraints (like $=$, $\neq$, $\in$, $\notin$, $\subset$, $\subseteq$), and dedicated constraints like:

- $\texttt{isNotEmpty}(X_j)$ is satisfied iff $X_j \neq \emptyset$
- $\texttt{coverTransactions}([X_1, ..., X_k])$ is satisfied iff each transaction is covered by at least one pattern ($\bigcup_{1 \leq i \leq k} \texttt{cover}(X_i) = \mathcal{T}$)
- $\texttt{noOverlapTransactions}([X_1, ..., X_k])$ is satisfied iff all i, j s.t. $1 \leq i < j \leq k$, $\texttt{cover}(X_i) \cap \texttt{cover}(X_j) = \emptyset$
- $\texttt{coverItems}([X_1, ..., X_k])$ is satisfied iff each item belongs to at least one pattern ($\cup_{1 \leq i \leq k} X_i = \mathcal{I}$)
- $\texttt{noOverlapItems}([X_1, ..., X_k])$ is satisfied iff $\forall\ i, j$ s.t. $1 \leq i < j \leq k$, $X_i \cap X_j = \emptyset$
- $\texttt{canonical}([X_1, ..., X_k])$ is satisfied iff for all i s.t. $1 \leq i < k$, pattern X_i is less than pattern X_{i+1} with respect to the lexicographic order.

Finally, a query is a conjunction of constraints as illustrated in the next section.

Table 1. `Animal` dataset

Species	trans	Fur	Feather	Scale	Milk	Egg	Beak	Bone	Meat	Grass	Fish
Cat	t_1	1	0	0	1	0	0	1	1	0	1
Cow	t_2	1	0	0	1	0	0	1	0	1	0
Crow	t_3	0	1	0	0	1	1	1	1	1	1
Daulphin	t_4	0	0	0	1	0	0	1	0	0	1
Dog	t_5	1	0	0	1	0	0	1	1	0	0
Goose	t_6	0	1	0	0	1	1	1	0	1	0
Platypus	t_7	1	0	0	1	1	1	1	1	0	0
Salmon	t_8	0	0	1	0	1	0	0	0	1	1
Shark	t_9	0	0	0	0	0	0	0	1	0	1
Trout	t_{10}	0	0	1	0	1	0	0	0	1	0
Vulture	t_{11}	0	1	0	0	1	1	1	1	0	1

3 Constrained Clustering: Modeling

3.1 Introduction: Modeling a Clustering Query

A clustering problem can be thought of as a scenario in which a user wishes to obtain a partition $\pi_T = (X_1, ..., X_k)$ of a dataset $\mathcal{T}$, containing m objects, into k (non-empty) clusters. A clustering problem intrinsically owns a lot of symmetrical solutions: any permutation of π_T is a solution. The $\texttt{canonical}([X_1, ..., X_k])$ constraint is used to avoid symmetrical solutions.

So, we can define the $\texttt{isClustering}([X_1, ..., X_k])$ constraint:

$$\texttt{isClustering}([X_1, ..., X_k]) \equiv \begin{cases} \wedge_{1 \leq i \leq k} \texttt{isNotEmpty}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapTransactions}([X_1, ..., X_k]) \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

3.2 Integrating Background Information in the Clustering Process

In many application domains, background information on the domain and/or dataset is often available and the data analyst would like to integrate it in the process to improve the clustering results. Such a knowledge is usually expressed as *transaction-level* constraints (like the `mustLink` and `cannotLink` constraints [20], and as *cluster-level* constraints (like *Maximum Diameter* and *Minimum Separation* contraints [7,8]).

We start by describing how these information can be modeled thanks to the constraint-based language (see Section 2). Then, we show how our method allow to combine them to achieve more relevant queries. Let $\mathcal{T}$ be a dataset of m transactions. Let $d(t_1, t_2)$ be a distance over transactions.

Transaction-Level Constraints consist in `mustLink` and `cannotLink` constraints [20]:

- $\texttt{mustLink}(t_1, t_2)$ ensures that transactions t_1 and t_2 belong to the same cluster.
- $\texttt{cannotLink}(t_1, t_2)$ ensures that transactions t_1 and t_2 do not belong to the same cluster.

Cluster-Level Constraints. The *diameter of a cluster* X_j is the maximum distance between a pair of transactions in X_j [7,8]. The cluster-level constraint *maximum diameter* requires that the diameter of any cluster be at most a given value α. To achieve this, we must ensure that any pair of transactions (t_i, t_j) with $d(t_i, t_j) > \alpha$ are in different clusters. So, for $1 \leq i < j \leq m$, if $d(t_i, t_j) > \alpha$ then the constraint $\texttt{cannotLink}(t_i, t_j)$ must be added.

The *separation between two clusters* X_i *and* X_j is the minimum distance between a pair of transactions, one from X_i and the other from X_j. The cluster-level constraint Minimum Separation requires that the separation between two clusters be at least a given value β. To achieve this, we must ensure that any pair of transactions (t_i, t_j) with $d(t_i, t_j) < \beta$ are in the same cluster. So, for $1 \leq i < j \leq m$, if $d(t_i, t_j) < \beta$ then the constraint $\texttt{mustLink}(t_i, t_j)$ must be added. Cluster-level constraints can be combined together (query q_1).

$$q_1([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i, t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i, t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \end{cases}$$

The method can be performed with any distance between transactions. For instance, when transactions are described with numerical values, a numerical distance such as the euclidian distance can be used.

Seeding. Background information both on transactions and clusters is easily modeled in the same way. Let t_{i_0} and t_{j_0} be two transactions and X_{j_1} a cluster. t_{i_0} and t_{j_0} must be (resp. must not be) in a same cluster is modeled by adding the constraint $\texttt{mustLink}(t_{i_0}, t_{j_0})$ (resp. $\texttt{cannotLink}(t_{i_0}, t_{j_0})$). t_{i_0} must be (resp. must not be) in the cluster X_{j_1} is modeled by adding the constraint $t_{i_0} \in X_{j_1}$ (resp. $t_{i_0} \notin X_{j_1}$).

3.3 Stepwise Refinements for Clustering

A major strength of our approach is to provide a simple and efficient way to declare and refine queries, that is usually the process conducted by a data analyst when he performs clustering tasks. Starting from an initial query (like q_1), the data analyst can express that he prefers solutions with a minimal size of the clusters, in which the sizes of clusters do not differ too much from each other, etc. In practice, the data analyst successively refines the query (deriving q_{i+1} from q_i) until he considers that relevant information has been extracted. This stepwise refinement process is easily handled by our constrained clustering approach as illustrated below.

Removing Clusterings with Small Size Patterns. A clustering including at least one cluster X_i of small size is not considered as useful because X_i does not ensure enough similarity between transactions associated to X_i. Adding a minimal size threshold solves this drawback (query q_2).

$$q_2([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i \leq k} \texttt{size}(X_i) \geq \delta \end{cases}$$

Balanced Clustering. Clustering solutions in which the sizes of clusters do not differ too much from each other are generally preferred. For any pair of clusters (X_i, X_j), their difference of sizes must be lower than a threshold $\Delta \times m$ where Δ is a percentage (query q_3).

$$q_3([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq k} \mid \texttt{size}(X_i) - \texttt{size}(X_j) \mid \leq \Delta \times m \end{cases}$$

3.4 An Example of Stepwise Refinements

Let k=3 and $d(t_1, t_2)$ be the Hamming distance between transaction t_1 and transaction t_2. Using the dataset $\mathcal{T}$ described in Table 1, query q_1' provides 966 solutions for α=9 and β=1. By refining these thresholds (decreasing the maximal diameter to α=8 and enlarging the minimal separation to β=2), there remain four solutions (see Table 2).

$$q_1'([X_1, X_2, X_3]) \equiv \begin{cases} \texttt{isClustering}([X_1, X_2, X_3]) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) < \beta} \texttt{mustLink}(t_i, t_j) \wedge \\ \wedge_{1 \leq i < j \leq m, d(t_i,t_j) > \alpha} \texttt{cannotLink}(t_i, t_j) \end{cases}$$

Table 2. Set of clusterings for q_1' (α=8 and β=2)

Sol.	X_1	X_2	X_3
s_1	$\{t_1,t_2,t_4,t_5,t_7\}$	$\{t_3,t_6,t_8,t_{10},t_{11}\}$	$\{t_9\}$
s_2	$\{t_1,t_2,t_4,t_5,t_7\}$	$\{t_3,t_6,t_{11}\}$	$\{t_8,t_9,t_{10}\}$
s_3	$\{t_1,t_2,t_4,t_5,t_7\}$	$\{t_3,t_6,t_9,t_{11}\}$	$\{t_8,t_{10}\}$
s_4	$\{t_1,t_2,t_4,t_5,t_7,t_9\}$	$\{t_3,t_6,t_{11}\}$	$\{t_8,t_{10}\}$

Clusters with a small size (e.g., $\{t_9\}$ in solution s_1) are considered irrelevant. By adding δ=2 as a minimal cluster size threshold, we get query q_2' and there remains three solutions (s_2, s_3, and s_4, see Table 2).

$$q_2'([X_1, X_2, X_3]) \equiv \begin{cases} q_1'([X_1, X_2, X_3]) \wedge \\ \bigwedge_{1 \leq i \leq 3} \texttt{size}(X_i) \geq \delta \end{cases}$$

The user may want to indicate that the *shark* (t_9) must be in the same cluster as the *salmon* (t_8). This is done with a `mustlink` constraint. The solution s_2 is then the unique solution for the query q_3' where the k=3 clusters respectively denote mammals, birds and fish.

$$q_3'([X_1, X_2, X_3]) \equiv \begin{cases} q_2'([X_1, X_2, X_3]) \wedge \\ \texttt{mustlink}(t_8, t_9) \end{cases}$$

3.5 Other Clustering Problems

In the same way, it is easy to express other clustering problems [5] such as soft clustering and co-clustering, the latter being well-used in bioinformatics for exploring gene expression data. **Soft clustering** is a relaxed version of the clustering where small overlaps on transactions (less than a threshold δ_T) are allowed.

$$q_4([X_1, ..., X_k]) \equiv \begin{cases} \wedge_{1 \leq i \leq k} \texttt{isNotEmpty}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k} \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

Co-clustering consists in finding k clusters covering both the set of transactions and the set of items, without any overlap on transactions or on items.

$$q_5([X_1, ..., X_k]) \equiv \begin{cases} \texttt{isClustering}([X_1, ..., X_k]) \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \texttt{noOverlapItems}([X_1, ..., X_k]) \end{cases}$$

Soft co-clustering is a relaxed version of the co-clustering, allowing small overlaps on transactions (less than δ_T) and on items (less than δ_I).

$$q_6([X_1, ..., X_k]) \equiv \begin{cases} \wedge_{1 \leq i \leq k} \texttt{isNotEmpty}(X_i) \wedge \\ \texttt{coverTransactions}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k} \texttt{overlapTransactions}(X_i, X_j) \leq \delta_T \wedge \\ \texttt{coverItems}([X_1, ..., X_k]) \wedge \\ \wedge_{1 \leq i < j \leq k} \texttt{overlapItems}(X_i, X_j) \leq \delta_I \wedge \\ \texttt{canonical}([X_1, ..., X_k]) \end{cases}$$

4 Constrained Clustering: SAT Encoding

4.1 A Short Overview of SAT Solvers

Satisfiability (SAT) is the problem of determining if the variables of a given boolean formula can be assigned in such a way as to make the formula be evaluated to `True`. A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals.

SAT solvers perform filtering using unit propagation. If a CNF $\mathcal{F}$ contains a unit clause (composed of a single literal l), then $\mathcal{F}$ is satisfied, if and only if, l is assigned to `True`. So every clause containing l can be removed and $\neg l$ can be deleted in every clause it occurs. Binary clauses are well suited for unit propagation. If one of its two literals is assigned, a binary clause is either removed or becomes unitary giving raise to another filtering step (by unit propagation).

Efficient and scalable algorithms for SAT, that were developed over the last decade, have contributed to dramatic advances in the ability to automatically solve problem instances involving tens of thousands of variables and millions of constraints. That is why, we have chosen to encode a query as a CNF and then use a SAT solver to answer it.

4.2 Variables and Encoding of Partitioning Constraints

The data analyst formulates his queries by using the constraint-based language introduced above. Let $\mathcal{T}$ be the dataset to be proceeded. The CNF encoding a query q_i is the conjunction of the CNFs of the constraints involved in q_i.

Each unknown cluster is modeled using m boolean variables $T_{t,j}$ such that ($T_{t,j} = \texttt{True}$) iff transaction t belongs to cluster j. A cluster is referenced by its index between 1 and k.

- `coverTransactions`$([X_1, ..., X_k])$ ensures that each transaction $t{\in}\mathcal{T}$ belongs to at least one cluster. So, for each t, there exists at least one cluster X_j s.t. $t{\in}X_j$.
$$\bigwedge_{t\in\mathcal{T}}(\bigvee_{j\in[1..k]} T_{t,j})$$
- `noOverlapTransactions`$([X_1, ..., X_k])$ ensures that each transaction $t \in \mathcal{T}$ belongs to at most one cluster. So, a transaction t belongs to (at least) two clusters iff there exist X_i and X_j s.t. $(t \in X_i) \wedge (t \in X_j)$, i.e. $\bigvee_{1\leq i<j\leq k}(T_{t,i} \wedge T_{t,j})$. So, the negation must hold for each transaction t.
$$\bigwedge_{t\in\mathcal{T}}(\bigwedge_{1\leq i<j\leq k}(\neg T_{t,i} \vee \neg T_{t,j}))$$
- `isNotEmpty`(X_j) ensures that there exists at least one transaction $t \in \mathcal{T}$ that belongs to cluster X_j and is modeled by the clause: $\vee_{t\in\mathcal{T}} T_{t,j}$
- `canonical`$([X_1, ..., X_k])$ ensures that, for all i s.t. $1{\leq}i{<}k$, cluster X_i is less than cluster X_{i+1}. This constraint is encoded using a binary comparator.

A constraint `coverItems`$([X_1, ..., X_k])$ (resp. `noOverlapItems`$([X_1, ..., X_k])$) is encoded in the same way as a constraint `coverTransactions`$([X_1, ..., X_k])$ (resp. `noOverlapTransactions`$([X_1, ..., X_k])$).

4.3 Encoding `mustLink` and `cannotLink` Constraints

`mustLink`(t_1, t_2) ensures that the transactions t_1 and t_2 belong to the same cluster. So, for each $j \in [1..k]$, $T_{t_1,j} \Leftrightarrow T_{t_2,j}$.

$$\bigwedge_{1 \leq j \leq k} (\neg T_{t_1,j} \vee T_{t_2,j}) \wedge (T_{t_1,j} \vee \neg T_{t_2,j})$$

In the same way, `cannotLink`(t_1, t_2) is encoded as:

$$\bigwedge_{1 \leq j \leq k} (\neg T_{t_1,j} \vee \neg T_{t_2,j}) \wedge (T_{t_1,j} \vee T_{t_2,j})$$

So each transaction level constraint is encoded using $(2 \times k)$ binary clauses.

4.4 Encoding Threshold Constraints Using Sorting Networks

Threshold constraints are directly modeled using cardinality constraints. So, `size`$(X_j) \geq \delta_1$ is modeled as $\#(T_{1,j}, T_{2,j}, ..., T_{m,j}) \geq \delta_1$. This cardinality constraint states that at least δ_1 variables $T_{t,j}$ must be assigned to `True`. Other threshold constraints involving function symbols are encoded in the same way.

Several efficient CNF encodings of cardinality constraints have been proposed [1,18]. Cardinality constraints are encoded thanks to unary adders in order to perform filtering by unit propagation. But, such encodings require a quadratic number of clauses [1] or they depend on the value of the threshold [18]. Moreover, for clustering, or other data mining tasks, thresholds can have rather large values, so the size of such encodings can quickly become prohibitive.

We used sorting networks to encode threshold constraints because the size of the resulting encoding does not depend on the value of the threshold. Moreover, using sorting networks to implement cardinality constraints preserves arc-consistency [11]. The odd-even *Batcher sort* [4] proved to be very efficient compared to other encodings of cardinality constraints [11].

4.5 Transitive Inference of `mustLink` and `cannotLink` Constraints

Let $G{=}(V, E)$ be the `mustLink` graph where $V{=}\mathcal{T}$. There is an edge between t_i and t_j iff there exists a constraint `mustLink`(t_i, t_j) [3,20]. Let CC_1 and CC_2 be two connected components of G. *(i) If there exists a constraint* ***mustLink***(t_1, t_2) *with* $t_1 \in CC_1$ *and* $t_2 \in CC_2$*, then we can infer the constraints* ***mustLink***(x, y) *for all* $x \in CC_1$ *and* $y \in CC_2$*.* Contrary to `mustLink`, `cannotLink` is not an equivalence relation, but *(ii) If there exists a constraint* ***cannotLink***(t_1, t_2) *with* $t_1 \in CC_1$ *and* $t_2 \in CC_2$*, then we can infer the constraints* ***cannotLink***(x, y) *for all* $x \in CC_1$ *and* $y \in CC_2$*.*

Such entailments are usually performed by adding all the inferred `mustLink` and `cannotLink` constraints [3,20]. But, using our SAT encoding (see Section 4.3), there is no need to perform those addings: all inferred constraints are implicitly stated and will be taken into account by the SAT solver. In fact, `mustLink` and `cannotLink` constraints are encoded using equivalence between boolean variables (see Section 4.3).

Table 3. Dataset's characteristics

dataset	Australian	Mushroom	P.-Tumor	Soybean	Zoo
m	653	8124	336	630	101
n	125	119	36	50	36
density	0.40	0.19	0.48	0.32	0.44

4.6 Ensuring Completeness

Given a CNF, SAT solvers either find one instantiation (and only one) for the variables evaluating the formula to `True`, or prove there is no such an instantiation. In order to ensure the completeness of our approach, restarts are performed. Let $\mathcal{F}$ be the CNF modeling a query q. Resolution begins with $\mathcal{F}$. Then, after having obtained the i-th solution s_i, its negation $\neg s_i$ is added to the (current) CNF and resolution is restarted in order to look for another solution. The process ends when a failure occurs, i.e. when all solutions have been found. Using restarts may seem too naive, but in practice is efficient enough. As CNFs contain much binary clauses, filtering by unit propagation is very effective (see experiments performed in Section 5).

5 Experiments

The goal of the experiments is to provide better insights on our constrained clustering method according to several constraints and datasets. We used the `MiniSat`[1] solver [10] to implement our method. We performed experiments on several datasets from the UCI repository[2] (see Table 3). Experiments were conducted on a Core2Duo E8400 (2.83GHz) with 4GB of RAM. For each experiment, we report the CPU-times needed to compute the first and the first ten solutions[3] according to the required number of clusters k.

We used the Hamming distance[4] between transactions. The maximum diameter α has been set to $n/2$ (if two transactions differ more than 50%, they cannot belong to the same cluster) and the minimum separation β to $n/20$ (if two transactions differ less than 5%, they must belong to the same cluster).

Fig. 1 reports CPU-times needed to compute the first and the first ten solutions for query q_1 (see Section 3.2). For each dataset, the first ten solutions (if there exist) are obtained very quickly, even for the dataset `Mushroom` which is the largest one.

Fig. 2 reports CPU-times needed to compute the first solution for balanced clustering (query q_3, Section 3.2). The balancing ratio Δ has been set to 10% (Fig. 2 left) and to 20% (Fig. 2 right). Even with additional threshold constraints, our approach is still efficient. Note that with such restrictive thresholds, few queries have a solution.

[1] `http://minisat.se/`

[2] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

[3] Symbol `'-'` denotes the absence of solution for a query.

[4] Any distance can be used since distance is only used to state `mustLink` and `cannotLink` constraints.

dataset	k=4	k=8	k=12	k=16	k=20
Australian	-	0.02	0.25	0.89	1.59
Mushroom	0.96	1.38	3.94	6.64	-
P.-Tumor	-	0.03	0.08	0.12	0.13
Soybean	0.01	0.03	0.12	0.16	-
Zoo	0.01	0.01	0.02	0.03	0.03

k=4	k=8	k=12	k=16	k=20
-	0.05	0.31	0.96	1.70
1.28	1.53	5.38	8.24	-
-	0.03	0.11	0.15	0.17
0.02	0.05	0.14	0.18	-
0.01	0.01	0.03	0.04	0.04

Fig. 1. (q_1) Time in s. to obtain the 1st sol. (left) and the first ten sol. (right)

dataset	k=4	k=8	k=12	k=16	k=20
Australian	-	19.7	27.46	45.23	180.5
Mushroom	16.9	-	-	-	-
P.-Tumor	-	1.06	2.22	6.16	6.05
Soybean	-	-	-	-	-
Zoo	0.01	0.09	-	-	-

k=4	k=8	k=12	k=16	k=20
-	7.20	13.10	46.15	69.76
19.6	-	-	-	-
-	0.91	3.30	3.88	6.08
-	-	-	-	-
0.03	0.10	-	-	-

Fig. 2. (q_3) Time in s. for the 1st sol. with Δ=10% (left) and with Δ=20% (right)

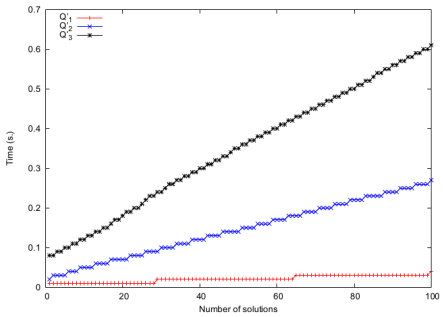

Fig. 3. Time for Zoo (k=6)

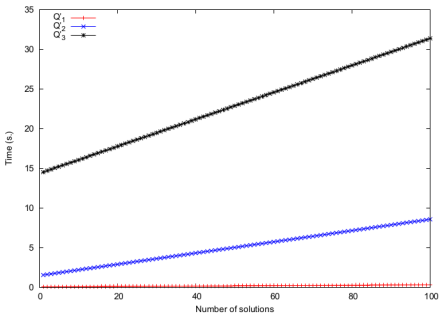

Fig. 4. Time for Australian (k=6)

Australian		($k=2$)	($k=6$)	($k=10$)
q_1	#literal	2.6	10.4	18.2
	#clause	29.8	107.8	196.3
	r%	86.9	81.2	83.93
q_2	#literal	98.8	299.2	499.5
	#clause	318.7	974.5	1640
	r%	68.6	68.3	68.5
q_3	#literal	98.8	299.2	499.5
	#clause	318.7	974.5	1640
	r%	68.6	68.3	68.5

Mushroom		($k=2$)	($k=6$)	($k=10$)
q_1	#literal	32.4	129.9	227.4
	#clause	1141	3651	6291
	r%	95.7	93.1	92.9
q_2	#literal	1343	4062	6781
	#clause	5075	15452	25960
	r%	73.2	72.9	73.1
q_3	#literal	1343	4062	6781
	#clause	5075	15452	25960
	r%	73.2	72.9	73.1

Fig. 5. Number of literals and clauses for the three queries (in thousands)

Fig. 3 depicts the CPU-times according to the number of solutions for dataset Zoo with k=6. A curve is associated to each of the three queries q_1 (red), q_2 with δ=$m/10$ (blue) and q_3 with Δ=20% (black). These curves *seem to be quasi-linear*. All the three queries mainly involve mustLink and cannotLink constraints which are encoded using binary clauses. As soon as a transaction is assigned to a cluster,

a lot of deductions are performed using unit propagation and transitive inferences (see Section 4.5). Nevertheless, further investigations are required to confirm this result. Fig. 4 provides similar results for dataset `Australian` with k=6.

Fig. 5 reports the size of the encodings of queries q_1, q_2, and q_3 for datasets `Australian` and `Mushroom`, as well as the ratio (r) of binary clauses constituting the CNF. The encoding of a query could be large (several millions of clauses), but the ratio r always remains very high. The size of q_2 is similar as the size of q_3 since only the bounds of the cardinality constraints are changed (see Section 4.4). **To sum up,** these experiments show that SAT solvers allow to solve efficiently this clustering task. They can find the first solutions (or prove there is none) in affordable times, even for medium scale size datasets as `Mushroom`. However, most of the clustering queries need threshold constraints which require more computational efforts.

6 Related Work

SAT for Clustering. Constraints on clusters (e.g., *Maximum Diameter* and *Minimum Separation*) into the `k-means` clustering algorithm [16] have been introduced by [7]. A formal complexity analysis of constraints on transactions and clusters is performed in [8]. Davidson *and al.* have proposed the first approach using SAT for clustering [9], but it deals only with a strong limited setting (k=2). The authors show how constraints both on transaction and cluster levels can be modeled and solved as instances of the 2-SAT problem. Gilpin and Davidson consider hierarchical constrained clustering and describe how dendograms can be modeled and solved as instances of the Horn-SAT problem [13]. We have also used SAT to implement primitives of our constraint-based language [17].

SAT for Mining Patterns. a SAT approach for enumerating all frequent patterns with wildcards in a given sequence has been proposed in [6].

7 Conclusion and Future Work

We have proposed a constrained clustering approach handling a large set of constraints. Solving is performed thanks to a SAT encoding for clustering and we have described how queries can be solved by taking benefit from features of SAT solvers. Experiments performed using `MiniSat` show the feasibility and the effectiveness of our approach. An insight of our work is that when a certain clustering task is modeled, many variants of that task can be modeled as well, changing or adding a few constraints is sufficient to allow this to happen.

As future work, we want to enrich our constraint-based language with further primitives to determine and/or constrain the cluster center locations. We also want to improve the current encoding (e.g., defining labeling orderings, exploiting backdoors, nogoods...). Another challenge is to propose an alternative encoding consuming less space and yet having relevant properties for an efficient solving. We want to conduct an in-depth study of the scalability of the approach to larger values of k and larger datasets. Finally, another promising direction is to integrate optimization criteria in our framework.

References

1. Bailleux, O., Boufkhad, Y.: Efficient CNF Encoding of Boolean Cardinality Constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003)
2. Banerjee, A., Ghosh, J.: Scalable clustering algorithms with balancing constraints. Data Min. Knowl. Discov. 13(3), 365–395 (2006)
3. Basu, S., Davidson, I., Wagstaff, K.L.: Constrained Clustering: Advances in Algorithms, Theory, and Applications. Chapman & Hall (2008)
4. Batcher, K.E.: Sorting networks and their applications. In: AFIPS Spring Joint Computing Conference. AFIPS Conference Proceedings, vol. 32, pp. 307–314. Thomson Book Company, Washington, D.C (1968)
5. Berkhin, P.: Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, USA (2002)
6. Coquery, E., Jabbour, S., Sais, L.: A constraint programming approach for enumerating motifs in a sequence. In: Workshop on Declarative Pattern Mining, ICDM 2011, Vancouver, Canada, pp. 1091–1097 (December 2011)
7. Davidson, I., Ravi, S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: SDM (2005)
8. Davidson, I., Ravi, S.: Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. Data Min. Knowl. Discov. 18(2), 257–282 (2009)
9. Davidson, I., Ravi, S., Shamis, L.: A SAT-based framework for efficient constrained clustering. In: SDM, pp. 94–105. SIAM (2010)
10. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
11. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. JSAT 2(1-4), 1–26 (2006)
12. Fisher, D.H.: Knowledge acquisition via incremental conceptual clustering. Machine Learning 2(2), 139–172 (1987)
13. Gilpin, S., Davidson, I.: Incorporating SAT solvers into hierarchical clustering algorithms: an efficient and flexible approach. In: KDD 2011, pp. 1136–1144 (2011)
14. Guns, T., Nijssen, S., De Raedt, L.: Itemset mining: A constraint programming perspective. Artif. Intell. 175(12-13), 1951–1983 (2011)
15. Khiari, M., Boizumault, P., Crémilleux, B.: Constraint Programming for Mining n-ary Patterns. In: Cohen, D. (ed.) CP 2010. LNCS, vol. 6308, pp. 552–567. Springer, Heidelberg (2010)
16. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, pp. 281–297. University of California Press (1967)
17. Métivier, J.-P., Boizumault, P., Crémilleux, B., Khiari, M., Loudni, S.: A constraint-based language for declarative pattern discovery. In: 27th Annual ACM Symposium on Applied Computing (SAC 2012), March 2012, pp. 438–444 (2012)
18. Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)
19. Tung, A.K.H., Han, J., Lakshmanan, L.V.S., Ng, R.T.: Constraint-Based Clustering in Large Databases. In: Van den Bussche, J., Vianu, V. (eds.) ICDT 2001. LNCS, vol. 1973, pp. 405–419. Springer, Heidelberg (2000)
20. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: ICML 2000, pp. 1103–1110. M. Kaufmann (2000)