Constrained Clustering using Column Generation

Behrouz Babaki[1], Tias Guns[1], and Siegfried Nijssen[1,2]

[1] Department of Computer Science, KU Leuven
{*firstname.lastname*}@cs.kuleuven.be
[2] LIACS, Universiteit Leiden

Abstract. In recent years, it has been realized that many problems in data mining can be seen as pure optimisation problems. In this work, we investigate the problem of constraint-based clustering from an optimisation point of view. The use of constraints in clustering is a recent development and allows to encode prior beliefs about desirable clusters. This paper proposes a new solution for minimum-sum-of-squares clustering under constraints, where the constraints considered are must-link constraints, cannot-link constraints and anti-monotone constraints on individual clusters. Contrary to most earlier approaches, it is exact and provides a fundamental approach for including these constraints. The proposed approach uses column generation in an integer linear programming setting. The key insight is that these constraints can be pushed into a branch-and-bound algorithm used for generating new columns. Experimental results show the feasibility of the approach and the promise of the branch-and-bound algorithm that solves the subproblem directly.

1 Introduction

One of the core problems studied in the data mining and machine learning literature is that of *clustering*. Given a database of examples, the clustering task involves identifying groups of similar examples; such groups are for instance indicative for patients with similar clinical observations, customers with similar purchase behaviour, or website visitors with similar click behaviour.

While the clustering problem is common in the data mining literature, it is only recently realized in the data mining community that this problem is closely related to problems studied in the optimization literature, and hence that open problems in clustering may be solved using generic optimization tools. In this paper, we study one such open problem: *Optimal Constrained Minimum Sum-of-Squares Clustering* (MSSC). We show that a generic optimization strategy can be used to address this problem.

Many types of clustering problems are known in the literature; however, MSS clustering is arguably one of the most popular clustering settings. In MSS clustering, the task is to find a clustering in which each example is put into *exactly* one cluster. Clusters do not overlap and together they cover all the available data. Clusters should be chosen such that points within a cluster have small sum-of-squared distances.

The popularity of the MSS clustering setting is partially due to the *k-means* algorithm. K-means is a heuristic algorithm which quickly converges to a local minimum and is included in most data mining toolkits. Even though successful, basic k-means has several disadvantages. One is its randomized nature: each run of the algorithm may yield a different clustering. Another is its lacking ability to take into account prior knowledge of a user.

There are many types of prior knowledge that a user may have. A common perspective is to formalize prior knowledge in terms of *constraints* on the clusters one wishes to find [6], where the most popular constraints are *must-link* and *cannot-link* constraints. A must-link constraint enforces that examples that are known to be related, are part of the same cluster. A cannot-link constraint, on the other hand, enforces that examples that are not related are not part of the same cluster. These constraints are popular as many other constraints can be transformed into must-link and cannot-link constraints [9]. The maximum cluster diameter constraint, for instance, requires that each cluster must have a diameter of at least distance α; hence, any two points that are further than α apart cannot link together. The minimum cluster separation constraint requires that clusters must be separated by at least distance β; hence, any two points that are less than β apart must link together.

Other clustering settings that can be seen as constraint-based clustering problems are problems in which clusters need to have a minimum or maximum size, or where one is looking for alternative clusterings [15].

An important question is how to find a clustering that satisfies constraints. Here, most algorithms in the data mining literature take a heuristic approach. Arguably, the most well-known example is the *COP-k-means* algorithm [23], which modifies the *k-means* algorithm to deal with must-link and cannot-link constraints. Unfortunately, even though the algorithm is fast, it may not find a solution that satisfies all constraints even if such a solution exists [9]. In itself, this is not surprising as the problem is known to be NP hard and hence a polynomial solution is not likely to exist [2, 9]. As a result, the problem of how to solve the MSSC problem under constraints is still open.

This paper addresses this challenge and develops a generic approach that can find an optimal solution to constrained MSSC problems. While we will focus on must-link and cannot-link constraints, the approach allows for the inclusion of several other constraints as well; we will show that the approach works for all constraints that are *anti-monotone.*

Our approach builds on earlier work that showed the feasibility of *unconstrained* optimal MSS clustering [14, 4] by using *column generation* in an *integer linear programming* setting. The column generation process is here responsible for identifying candidate clusters that can be put into a clustering. We will show that most clustering constraints can be dealt with by *pushing* the constraints in a branch-and-bound algorithm for column generation.

This paper is organized as follows. Section 2 introduces MSS clustering and MSS clustering under constraints. Section 3 gives an overview of how to find a solution using a column generation process, building on the earlier work of [14, 4].

In Section 4 we introduce a branch-and-bound approach for generating columns under constraints. Section 5 discusses practical considerations in the implementation of this algorithm. Section 6 provides experiments, Section 7 discusses related work and Section 8 concludes.

2 MSSC

Assumed given is a dataset D with n data points. Each example in the dataset is a point p in an m-dimensional space and is represented by a vector with m values. One cluster is defined as a set of data points $C \subseteq D$. A cluster*ing* consists of k such clusters, and corresponds to a partitioning of the data into k groups. The number of clusters k is typically given upfront by the user. In MSS clustering, the clusters in a clustering are usually non-overlapping, that is, each data point belongs to exactly one cluster.

Given a cluster $C \subseteq D$, the cluster center or *centroid* is the mean of the data points that belong to that cluster:

$$z_C = \text{mean}(C) = \frac{\sum_{p \in C} p}{|C|} \tag{1}$$

The quality of a clustering can be measured in many different ways. In MSS clustering, the quality of a clustering is measured using the sum of squared distances between each point in a cluster and the centroid of the cluster: $SSC(\mathcal{C}) = \sum_{C \in \mathcal{C}} \sum_{p \in C} d^2(p, z_C)$, where $d(\cdot, \cdot)$ calculates the distance between two points, for example, the Euclidean distance.

Note that for a cluster C, the sum of squared distances to its centroid equals the sum of all pairwise distances between the points of that cluster, divided by the size of the cluster:

$$\sum_{p \in C} d^2(p, z_C) = \frac{\sum_{p_1, p_2 \in C} d^2(p_1, p_2)}{|C|} \tag{2}$$

For simplicity of notation, when we write $p_1, p_2 \in C$ we assume that every pair of two points in C is included in the sum exactly once. To summarize the MSSC problem, a mathematical programming formulation is given in Table 1.

The best known clustering algorithm that uses sum of squared distances is the k-means algorithm. It is an approximate algorithm that starts with an initial random clustering and iteratively minimizes the sum-of-squares using the following two steps: 1) add each data point to the cluster with closest cluster centre; 2) compute the new cluster centre of the resulting clusters. These two steps are iterated until convergence, that is, the cluster centres do not change any more. This procedure can get stuck in local minima and it is not uncommon that two different runs (e.g. with different initial clusters) produce different clusterings.

Constraints. The most well-known constraints are must-link and cannot-link constraints. Let ML and CL be subsets of $D \times D$. Then a cluster C satisfies a must-link constraint $(p_1, p_2) \in ML$ iff $|\{p_1, p_2\} \cap C| \neq 1$; it satisfies a cannot-link constraint $(p_1, p_2) \in CL$ iff $|\{p_1, p_2\} \cap C| \leq 1$.

Note that both constraints can be evaluated on the individual clusters in a clustering. This is a key observation for our work.

In a seminal paper by Wagstaff et al [23], the COP-k-means algorithm is proposed. COP-k-means is an extention of the k-means algorithm towards must-link and cannot-link constraints. It modifies the k-means algorithm by not assigning each point to its closest cluster centre, but rather to the closest centre that satisfies all constraints. If no such centre exists, the algorithm terminates. An alternative approach is to continue running the algorithm, even though the final solution might then not satisfy all constraints; in any case, the algorithm is not guaranteed to find a solution even if there exists one.

Many other constraints are possible. We will not give a complete overview here (see Section 7 and [6]). For this work it is however important to observe that many problems can be formalized using constraints that are *anti-monotone*. We call a boolean constraint $\varphi(C)$ on a cluster C of data points *anti-monotone* iff $\varphi(C)$ implies $\varphi(C')$ for all $C' \subseteq C$. The cannot-link constraint is anti-monotone: if a cluster C satisfies a cannot-link constraint, every subset also satisfies this constraint. There are many other anti-monotone constraints:

- a maximum cluster size constraint on clusters $|C| \leq \theta$, which can be used to avoid that one cluster dominates a clustering;
- a maximum overlap constraint $|C \cap X| \leq \theta$, which can be used to avoid that any cluster found is too similar to a given set of points X; this generalizes the cannot-link constraint;
- a minimum difference constraint $|C \backslash X| \leq \theta$, which requires a certain similarity to cluster X;
- a soft cannot-link constraint, which requires that the number of pairs of points in a cluster that have a cannot-link constraint among them is bounded;
- conjunctions or disjunctions of anti-monotone constraints.

A conjunction of anti-monotone constraints can for instance be used to find an alternative clustering: starting from a clustering $\mathcal{C}$, we can enforce that in a new clustering every cluster is different from all clusters in the earlier clustering.

Must-link constraints are an example of constraints that are not anti-monotone.

In the following sections, we will show how to solve the MSS problem under a combination of anti-monotone constraints and must-link constraints, by adapting a state-of-the-art unconstrained optimal clustering algorithm. A feature of the algorithm is that it exploits the anti-monotonicity of cluster constraints.

3 Column generation framework

In this section we give a brief overview of an ILP formulation of MSSC and a column generation method for solving it, based on the (unconstrained) MSSC column generation framework of Aloise et al. [4]. The next section will introduce our proposed approach for taking constraints into account.

$$\underset{\mathcal{C}}{\text{minimize}} \quad \sum_{C \in \mathcal{C}} \sum_{p \in C} d^2(p, z_C), \tag{3}$$

$s.t.$

$$C_1 \cap C_2 = \emptyset \qquad \forall C_1, C_2 \in \mathcal{C} \tag{4}$$

$$|\bigcup_{C \in \mathcal{C}} C| = n \tag{5}$$

$$|\mathcal{C}| = k \tag{6}$$

Table 1. MSS clustering

$$\underset{x}{\text{minimize}} \quad \sum_{t \in T} c_t x_t, \tag{7}$$

$s.t.$

$$\sum_{t \in T} x_t a_{it} = 1 \quad \forall i \in \{1, \dots, n\} \tag{8}$$

$$\sum_{t \in T} x_t = k \tag{9}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{10}$$

Table 2. An ILP model for MSS clustering

An ILP formulation of MSSC. Given a dataset with n data points, the number of possible clusters is 2^n. In principle, we can hence reformulate the clustering problem using a Boolean n by 2^n matrix A that represents all possible clusters: each column is a cluster where $a_{it} = 1$ if data point p_i is in cluster t and $a_{it} = 0$ otherwise. We define the *cost* of a cluster (column) as the sum of squared distances of the points in the cluster to its mean: $c_t = \sum_{i=1}^{n} d^2(p_i, z_t) a_{it}$.

The problem in equations 3-6 can then be formulated as an Integer Linear Program as in Table 2 [14], where $T = \{1, \dots, 2^n\}$ denotes all possible clusters. Equation 7 corresponds to the SSC criterion. Equation 8 states that each data point must be covered exactly once. Hence it enforces both that the clusters are not overlapping and that all points are covered. Equation 9 finally ensures that exactly k clusters are found. Note that the k-means (and COP-k-means) algorithm can return empty clusters and hence less than k clusters in some occasions. This can not arise in the above formulation.

For even moderate sizes of n the number of clusters will be too large to solve the above ILP by first materializing A. However, we can use a column generation approach in which the master problem (Eq. 7-10) is *restricted* to a smaller set $T' \subseteq T$ and columns (clusters) are incrementally added until the optimal solution is provably found.

Column Generation iterates between solving the restricted master problem and adding one or multiple columns. A column is a candidate for being added to the restricted master problem if adding it can improve the objective function. If no such column can be found, one is certain that the optimal solution of the restricted master problem is also the optimal solution of the full master problem. Whether a column can improve on the objective can be derived from the dual.

The dual of the master problem (Table 2) is given in Table 3. Here λ_i indicates a dual value corresponding to the constraint in equation 8 and σ a dual value corresponding to equation 9. One column in the master problem corresponds to one constraint in the dual (Equation 12).

$$\underset{\lambda,\sigma}{\text{maximize}} \quad -k\sigma + \sum_{i=1}^{n} \lambda_i \tag{11}$$

s.t.

$$-\sigma + \sum_{i=1}^{n} a_{it}\lambda_i \leq c_t \quad \forall t \in T \tag{12}$$

$$\lambda_i \geq 0 \qquad \forall i \in \{1, \ldots, n\} \tag{13}$$

$$\sigma \geq 0 \tag{14}$$

Table 3. Dual of the optimization problem.

$$\underset{x}{\text{minimize}} \quad \sum_{t \in T} c_t x_t + \sum_{i=1}^{n} \theta_i y_i, \tag{15}$$

s.t.

$$\sum_{t \in T} x_t a_{it} + y_i = 1 \quad \forall i \in N \tag{16}$$

$$\sum_{t \in T} x_t = k \tag{17}$$

$$x_t \in \{0, 1\} \qquad \forall t \in T \tag{18}$$

$$-\mu \leq y_i \leq \mu \qquad \forall i \in N \tag{19}$$

Table 4. Model with stabilization included ($N = \{1, \ldots, n\}$).

Given values for λ and σ, obtained by solving a restricted master problem, we need to determine whether there are columns for which $\sigma - \sum_{i=1}^{n} a_{it}\lambda_i + c_t < 0$, that is, whether there are columns with a *negative reduced cost*. If no such column can be found, the current solution is optimal.

Finding a column with negative reduced cost is called *pricing*. While a pricing routine can return any column with a negative reduced cost, one typically searches for the smallest one; hence we are interested in finding:

$$\arg\min_{t \in T} \quad \sigma - \sum_{i=1}^{n} a_{it}\lambda_i + c_t. \tag{20}$$

Solving this pricing problem is not trivial, given the large number of columns. The details of solving the pricing subproblem will be discussed in more detail in Section 4.

When solving the restricted master problem, it is possible that it has no feasible solution. In this case, Farkas' Lemma [22] can be used to add columns that gradually move the solutions of the restricted master problems closer to the feasible region, or to prove infeasibility of the master problem. This Farkas pricing is similar to the regular pricing explained above. In this case, the problem to optimize is:

$$\arg\min_{t \in T} \quad \sigma' - \sum_{i=1}^{n} a_{it}\lambda'_i \tag{21}$$

where σ' and λ' are the dual Farkas values. Note that this is the same problem as the regular pricing problem above, with the exception that the cost c_t of the cluster does not need to be taken into account.

4 Column generation with constraints.

Given the earlier observations, one can see that enforcing constraints on clusters C amounts to removing from the cluster matrix A all clusters that do not satisfy these constraints. In a column generation scheme, this means that it is sufficient to add these constraints to the subproblem solver; they do not need to be added to the master problem. The rest of this section explains our proposed branch-and-bound method for solving the (constrained) subproblem.

4.1 Subproblem solving

Essentially, in each iteration of the column generation process we need to solve a constrained minimisation problem. The objective function to minimize is given by equation 20 (equation 21 in case of infeasibility). By removing the constant σ and using equation 2, we can rewrite the objective as:

$$\arg\min_{t\in T} \quad \sum_{i=1}^{n} d^2(p_i, z_t)a_{it} + \sigma - \sum_{i=1}^{n} a_{it}\lambda_i \tag{22}$$

$$= \arg\min_{t\in T} \quad \frac{\sum_{i=1}^{n}\sum_{j=i+1}^{n} d^2(p_i, p_j)a_{it}a_{jt}}{\sum_{i=1}^{n} a_{it}} - \sum_{i=1}^{n} a_{it}\lambda_i \tag{23}$$

Let us represent the cluster $t \in T$ and its corresponding column $a_{\cdot t}$ as a set X. We define $d(X) = \sum_{i,j\in X} d^2(p_i, p_j)$, where every pair is only considered once in the sum, $d(X,Y) = \sum_{i\in X, j\in Y} d^2(p_i, p_j)$ and $\lambda(X) = \sum_{i\in X} \lambda_i$. We can now rephrase our problem as that we wish to search for a cluster X:

$$\arg\min_{X} \quad \frac{d(X)}{|X|} - \lambda(X) \tag{24}$$

and such that all constraints on clusters are satisfied.

Blocks. A first simple observation is that the must-link constraints are transitive and hence the must-link relation is an equivalence relation. We will refer to the equivalence classes as *blocks*. We can rephrase our optimization problem as an optimization problem over the blocks. Let $X = [p_i]_{ML}$ denote the block that point $p_i \in D$ belongs to (a point can never belong to two blocks) and let $D/ML = \{[p_i]_{ML} \,|\, p_i \in D\}$ denote the blocks in the data. We are looking for a subset of the blocks $\bar{X} \subseteq D/ML$ such that the following criterion is minimized:

$$f(\bar{X}) = \left(\sum_{X\in\bar{X}} d(X) + \sum_{X,Y\in\bar{X}} d(X,Y) \right) / \sum_{X\in\bar{X}} |X| - \sum_{X\in\bar{X}} \lambda(X). \tag{25}$$

Note that we can precompute the terms $d(X)$, $d(X,Y)$, $|X|$ and $\lambda(X)$ for all $X, Y \in D/ML$. Note furthermore that if $ML = \emptyset$ then $\forall X \in \bar{X} : |X| = 1$ and this formula is identical to the one without constraints.

In addition, the choice of $\bar{X}$ has to satisfy the cannot-link constraint: for no two $X, Y \in \bar{X}$ it may be the case that $i \in X, j \in Y, (i,j) \in CL$.

Algorithm 1 Branch-and-bound(Set: $\bar{X}$, Set: $\bar{C}$)

$\bar{X}$ is the current set of blocks under consideration, $\bar{C}$ the possible extensions to $\bar{X}$.

```
1: $\bar{C}$ := reduce-candidates($\bar{X}$, $\bar{C}$)
2: if not prunable($\bar{X}, \bar{C}$) then
3:     Store $\bar{C}$ in a stack
4:     Process $\bar{X}$ as candidate cluster
5:     while $\bar{C}$ is not empty do
6:         $C$ := $\bar{C}$.pop ()
7:         Branch-and-bound ($\bar{X} \cup \{C\}$, $\bar{C}$)
8:     end while
9: end if
```

Algorithm. We propose to use a branch-and-bound algorithm to solve this problem. This algorithm performs a set-enumeration and is given in Algorithm 1 (initialized with Branch-and-bound($\{\}$, D/ML)). It uses newly developed pruning strategies to make the search feasible and is easily extended to include a wide range of constraints. In order to prune candidates, we either remove some candidates from consideration (line 1) or discard a branch of the search tree using bounds on the objective function (line 2).

The removal of candidates in line 1 corresponds to propagation in a constraint programming setting [8]. However, we will show that the proposed bound used in line 2 is not valid in the presence of arbitrary constraints and hence cannot be used in general.

4.2 Reducing the number of candidates

We employ three strategies to reduce the set of candidates in line 1 of Algorithm 1:

Cannot-link constraints. The cannot-link constraint is easily taken into account: when there is a cannot-link constraint between a block in $\bar{C}$ and a block in $\bar{X}$, the block is removed from $\bar{C}$.

Anti-monotone constraints other than cannot-link constraints are easily included as well: if a set $\bar{X} \cup \{C\}$ does not satisfy an anti-monotone constraint, the candidate C can be removed in line 1.

Block compatibility. Assume that we have a block $C_1 \in \bar{X}$ and a block $C_2 \in \bar{C}$ and the following holds:

$$\frac{d(C_1) + d(C_2) + d(C_1, C_2)}{|C_1| + |C_2|} - \lambda(C_1) - \lambda(C_2) > 0,$$

then any cluster $\bar{X}'$ we could build that includes both C_1 and C_2 can be improved by removing both C_1 and C_2:

$$\begin{aligned}f(\bar{X}) = \sum_{p_i \in \cup\bar{X}} d(p_i, z_{\cup\bar{X}})^2 - \sum_{X \in \bar{X}} \lambda(X) \geq \\ \sum_{p_i \in \cup\bar{X}\setminus\{C_1,C_2\}} d(p_i, z_{\cup\bar{X}\setminus\{C_1,C_2\}})^2 + \sum_{p_i \in C_1 \cup C_2} d(p_i, z_{C_1 \cup C_2})^2 - \sum_{X \in \bar{X}} \lambda(X) \geq \\ \sum_{p_i \in \cup\bar{X}\setminus\{C_1,C_2\}} d(p_i, z_{\cup\bar{X}\setminus\{C_1,C_2\}})^2 + - \sum_{X \in \bar{X}\setminus\{C_1,C_2\}} \lambda(X). \quad (26)\end{aligned}$$

Note that this argument is only valid in the presence of anti-monotone constraints in combination with must-link constraints. We refer to this test as a *compatibility test.* When a block in $\bar{C}$ is incompatible with a block in $\bar{X}$, the block is removed from $\bar{C}$.

4.3 Pruning using a bound on the objective function

For the remaining set of candidates, a more elaborate test is carried out to determine whether to continue the search (line 2). This test consists of calculating a bound on achievable solutions and comparing it with the best solution found so far. A key feature of this bound is that it can be calculated efficiently.

The key idea is as follows. Let $\bar{X}'$ be a set that is found below a set $\bar{X}$ in the search tree, that is, $\bar{X}' \subseteq \bar{C} \cup \bar{X}$. We can write its quality as follows:

$$(\underbrace{d(\cup\bar{X})}_{\text{old}} + \underbrace{\sum_{X \in \bar{X}'\setminus\bar{X}} \beta(\bar{X}, X)}_{\text{(1) between old and new}} + \underbrace{\sum_{X,Y \in \bar{X}'\setminus\bar{X}} d(X,Y)}_{\text{(2) between new blocks}})/ \underbrace{\sum_{X \in \bar{X}'} |X|}_{\text{(3) sizes}} - \underbrace{\sum_{X \in \bar{X}'} \lambda(X)}_{\text{(4) lambdas}},$$

where $\beta(\bar{X}, X) = d(X) + \sum_{Y \in \bar{X}} d(X,Y)$.

Essentially, we need to have a bound on the best $\bar{X}'$. An important first concern is that we do not know the size of the best $\bar{X}'$ and hence we do not know term (3). We simplify this problem by iterating over all cluster sizes $\sum_{X \in \bar{X}} |X| \leq s \leq \sum_{X \in \bar{X}} |X| + \sum_{C \in \bar{C}} |C|$ and calculating a bound on the quality assuming the best cluster has size s, i.e., we calculate a bound on the above formula assuming part (3) is iteratively fixed. The overall bound is the best bound among all the sizes considered.

Calculating a lower bound for a fixed value s of (3) requires a lower bound on (1) and (2), and an upper bound on (4). We discuss each in turn. A lower bound on part (1) for a given size s is obtained as follows:

- sort all $C \in \bar{C}$ increasing in their $\beta(\bar{X}, C)/|C|$ values, yielding order $C_1, \ldots, C_m$;
- determine the largest value k such that $\sum_{i=1}^{k} |C_i| \leq s$;
- determine $\sum_{i=1}^{k} \beta(\bar{X}, C_i)$ as bound.

The argument for this is as follows. All additional points that are selected by the algorithm above in $C_1, \ldots, C_k$ are characterized by the $\beta(\bar{X}, C)/|C|$ value their corresponding block has. If we sum these characteristic values over all points, the result is $\sum_{i=1}^{k} \beta(\bar{X}, C_i)$. Choosing the lowest possible characteristic values is a lower bound as the sum of characteristic values of the points in the optimum $\bar{X}^*$, and hence also the value $\sum_{X \in \bar{X}^* \setminus \bar{X}} \beta(\bar{X}, X)$, can never be better.

A similar algorithm can be used to determine an upper bound for term (4):

- sort all $C \in \bar{C}$ decreasing in their $\lambda(C)/|C|$ values, yielding order $C_1, \ldots, C_m$;
- determine the smallest value k such that $\sum_{i=1}^{k} |C_i| \geq s$;
- determine $\sum_{i=1}^{k} \lambda(C_i)$ as bound.

A simple lower bound on term (2) is that it is always higher than zero. Calculating a good bound is hard, as we essentially need to solve an edge-weighted clique problem.

While the overall bound obtained is not very tight, also because term (1) and term (4) are sorted independently, it has important computational advantages. First, we can sort the $\lambda(C)/|C|$ and $\beta(\bar{X}, C)/|C|$ values before iterating over potential sizes; hence, we can avoid doing this repeatedly for each size s. Second, we do not need to consider all sizes s indicated earlier. If we consider the sorted ranges of λ and β values, there are ranges of sizes in which the bound does not change; the bound only changes when either a lambda value changes or a β value changes. It hence suffices to consider $2|\bar{C}|$ different sizes for s. Finally, we can maintain the bounds incrementally.

As a result, the overall bound over all sizes s can be calculated in $O(|\bar{C}| \log |\bar{C}|)$ time. As furthermore all required counts can be maintained incrementally in $O(|\bar{C}|)$ time, the overall time spent in one call of the Branch-and-bound algorithm (excluding recursive calls) is $O(|\bar{C}| \log |\bar{C}|)$; in other words, the complexity of the algorithm is *not* dependent on the number of points in the data, but *only* on the number of blocks that the must-link constraints identify in it.

5 Practical considerations

The column generation approach, in combination with the branch-and-bound algorithm, provides a fundamental approach for finding optimal solutions under constraints. However, several practical considerations are of importance when implementing the column generation approach.

5.1 Initialisation

Initially, there are no columns in the restricted master problem. This means that Farkas pricing needs to be performed until a feasible solution is found, which can be time consuming. However, assuming a heuristic solver such as COP-k-means finds a solution, one can initialize the restricted master problem with this known (sub-optimal) solution. This avoids the need for Farkas pricing, provides a number of good initial columns (cuts to the dual problem) as well as an upper bound for the master problem.

5.2 Branching

Integer linear programs are typically solved by solving a number of LP relaxations and using branching to enforce integrality. So far, we have described how we employ the column-generation method for solving the LP relaxations. In theory, if the solution to the linear program is fractional any type of branching can be used. In previous work [14] a Ryan-Foster branching scheme was employed. In this scheme, in the restricted problem two columns are determined that have a corresponding fractional value and that cover the same data point (p_1). Branching will enforce that in subsequent problems only one of these two columns can cover that point. Observe that no two columns cover exactly the same data points and hence they must differ in at least one data point (p_2). We can now branch by enforcing that in one branch points p_1 and p_2 are in the same cluster and that in the other branch p_1 and p_2 are not in the same cluster.

This type of branching naturally fits our approach as it corresponds to adding a must-link or cannot-link constraint. Compared to [14], the proposed approach can hence handle both constrained and unconstrained cases in the same principled manner.

5.3 Slow convergence

Many large-scale column generation approaches suffer from slow convergence. Similar to [14], we also observed degeneracy in our experiments: even when given the optimal solution, a large number of column generation iterations is required before the optimality is proved. We implemented a dual stabilisation scheme similar to the one of [14]: adding a linear penalisation to the dual objective corresponds to adding a *perturbation* variable to each of the constraints in equation 8 and adding them to the objective function, given in Table 4. Here y_i are the perturbation variables, $+/-\mu$ its bounds and θ_i its coefficients in the objective function. The θ_i form a stabilisation centre in the dual that will penalize duals that are too far from it. A good choice for θ is the dual λ values from the best known solution so far. The value of μ has to be progressively decreased until 0. At this point, all perturbation variables are 0 and the problem is identical to the original restricted master problem.

We employ a scheme where the θ_i are given an equal initial value and μ is set to 0.99. Each time an optimal solution to the perturbed restricted master problem is found, the θ_i values are changed to the duals of that optimal solution and μ is divided by 2^ℓ where ℓ is a counter of the number of such updates.

6 Experiments

Data was obtained from the UCI machine learning repository [5]. Table 5 lists the properties of the datasets.

We used the open-source SCIP [1] system as column generation framework. The branch-and-bound pricer is written in C++. Source code is available at

name	# points	dimensions	# labels
Iris	150	4	3
Wine	178	13	3
Soybean	47	35	4

Table 5. Description of datasets

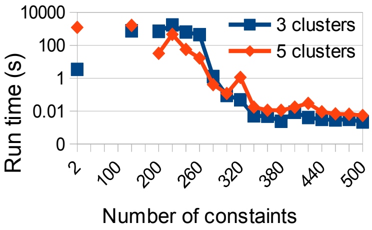

Fig 1. Run times on the Iris data set.

`http://dtai.cs.kuleuven.be/CP4IM/cccg/`. All experiments were run on quad-core Intel 64 bit computers with 16GB of RAM running Ubuntu Linux 12.04.3.

Constraints were generated according to the common methodology of [23]: two data points are repeatedly sampled randomly from labelled data; if they have the same label a ML constraint is generated, otherwise a CL constraint. This is repeated until the required number of constraints is generated. The code for generating these constraints and for the COP-k-means algorithm were obtained from `http://www.cs.ucdavis.edu/~davidson/constrained-clustering/` .

It is common practice to run (COP-)k-means multiple times to avoid that it is stuck in a local minimum. For each setting, we ran COP-k-means 500 times. The implementation obtained continues until convergence and is not guaranteed to satisfy all constraints. We will report on the number of runs that satisfy all constraint (COP sat). Only when at least one solution is found that satisfies all constraints will we report on its quality (COP max).

We initialized our column generation method with the *best* solution found by COP-k-means. Best is here defined by the clustering with the largest number of clusters satisfying all constraints. Among these clusterings, the one with the lowest MSS is selected. Note that in case COP-k-means did not find a solution satisfying all constraints, our column generation method started with the *best* infeasible solution. The stabilisation parameter μ was set to 0.99. Initial perturbation values θ_i can be set to any value; the update mechanism is explained in Section 3. In case a feasible solution is at hand, a good initial value for θ_i can be obtained from bounds on the dual variables. These bounds are calculated as in [14], and we used the lower bounds of the dual variables to initialize the corresponding θ_i.

The branch-and-bound method for solving the subproblem maintains a list of all clusters that improve the bound during search (including the final best one). All these clusters are added as columns to the restricted master problem.

Results. We compare the result of our column generation approach to that of repeated runs of COP-k-means. Our column generation approach is initialized as explained above, and a time-out of 30 minutes is used.

k=3			
#c	**COP sat**	**COP max**	**CG best**
2	100.00%	90.3725	90.3725
60	100.00%	83.6675	83.6675
100	37.20%	87.2082	87.2082
140	0%	-	87.8750*
200	0%	-	89.1496*
240	0%	-	85.2477*
300	0%	-	89.3868*
340	31.40%	89.3868	89.3868*
400	0%	-	89.3868*
440	31.00%	88.6409	88.6409*
500	0%	-	89.3868*

k=5				
#c	**COP sat**	**COP max**	**CG best**	**impr.**
2	100%	46.5616	46.5616	0%
60	100%	53.399	53.399	0%
100	100%	57.3827	57.3804*	0.004%
140	100%	63.1699	62.2115*	1.5%
200	100%	71.1401	69.3154*	2.56%
240	100%	72.7078	69.9776*	3.76%
300	83.6%	82.0819	81.9792*	0.13%
340	100%	85.9036	82.9945*	3.39%
400	100%	84.0495	84.0357*	0.02%
440	100%	82.6373	82.6373*	0%
500	100%	85.8908	85.8719*	0.02%

Table 6. Clustering with '#c' constraints, Iris dataset. *optimality proven

Table 6 shows the quality of the results for the Iris dataset, once for $k = 3$ (the true number of class labels, left) and once for $k = 5$ (right); Figure 1 gives an impression for the amount of run time it took to calculate these results.

A first observation is that in case of $k = 3$, and a low number of clusters, COP-k-means easily finds clusterings that satisfy the constraints (indicated by "COP sat"). For higher numbers of constraints, COP-k-means encounters more problems finding clusterings satisfying all constraints. In multiple cases none of the 500 runs finds a clustering satisfying all constraints. When we increase the number of clusters to $k = 5$, the constrained clustering problem becomes easier [9]; as a consequence, COP-k-means can find satisfying solutions more easily. Even when COP-k-means can not find a solution, our method finds acceptable clusterings; even optimal ones are found for higher numbers of constraints. The case of 140 constraints is an exception. For $k = 5$ and higher numbers of constraints, our method can find the optimal constrained clustering.

Table 7 shows the results for the bigger Wine dataset. This dataset is much harder, both for COP-k-means and for the column generation approach. In case of $k = 3$, the true number of class labels, COP-k-means is again rarely able to find a solution satisfying all constraints. The CG approach is able to find solutions for some cases, but can not prove them optimal within the time-out. In case of $k = 5$ the problem becomes easier, as was the case on Iris. We can see that the CG approach can sometimes greatly improve the best solution found in 500 COP-k-means runs, even without being able to prove its optimality.

Table 8 shows results on the Soybean dataset, a smaller dataset of higher dimensionality; its true number of labels is 4. Observe that for $k = 3$ and 80 constraints, CG is able to **prove** that this problem is infeasible. The heuristic COP-k-means simply does not find a solution, as happens for 40 and 60 constraints. We further note that in contrast to $k = 4$, for $k = 5$ COP-k-means is often not able to find the optimal solution.

	k=3		
#c	**COP sat**	**max**	**CG best**
240	0%	-	4860250*
300	0%	-	5133144*
340	0%	-	5214981*
380	0%	-	5220299*
420	0%	-	5232632*
460	0%	-	5232632*
500	0%	-	5232632*

	k=5			
#c	**COP sat**	**COP max**	**CG best**	**impr.**
240	100%	4021090	3327908*	17.24%
300	0%	-	4077296*	+
340	16.6%	4659910	4329603*	7.09%
380	66.6%	4729860	4450036*	5.92%
420	59.6%	4740180	4537678*	4.27%
460	94.2%	4819200	4540041*	5.79%
500	15%	4922560	4684355*	4.84%

Table 7. Clustering with '#c' constraints, Wine dataset. *optimality proven

	k=3		k=4		k=5	
# cons	COP sat.	CG quality gap	COP sat.	CG quality gap	COP sat.	CG quality gap
2	100.00%	0	100.00%	0	100.00%	0.00%
10	100.00%	0	100.00%	0*	100.00%	4.56%*
20	100.00%	0*	100.00%	0.12%*	100.00%	0.29%*
40	0.00%	339*	100.00%	0*	100.00%	1.25%*
60	0.00%	418*	52.60%	0*	81.20%	0.24%*
80	0.00%	INF	74.00%	0*	27.00%	0.38%*

Table 8. Soybean, different k and number of clusters (#c); GC quality gap = difference between best solution quality of cop-kmeans and the solution of CG, INF = infeasible.

7 Related work

We build on a column generation approach first described in [14] and improved in [4]. This earlier work only studies *unconstrained* clustering settings. We show that with modifications it can also be used in the presence of constraints. The main necessary modification is in the subproblem solver. We use a branch-and-bound approach that directly solves the subproblem and can be used in the presence of any constraint that is anti-monotone.

A feature of the first approach [14] is that it uses a heuristic *Variable Neighborhoud Search* method to solve a subproblem, and only when a solution can not be found in this way an exact method is used. The exact method uses Dinkelbach's lemma [13] to solve equation 23 through a series of unconstrained quadratic 0-1 problems. The latter are solved using a heuristic VNS combined with an exact branch-and-bound algorithm for verifying the stopping criterion of the Dinkelbach method.

This method is improved in [4]. One of these improvements is the introduction of a compatibility test. We adapted this test for use in the presence of must-link constraints.

Other exact methods for MSSC are branch and bound methods [18, 12, 7], a cutting plane algorithm that starts from the observation that MSSC is a concave optimisation problem [24], dynamic programming [16, 20] and a branch-and-cut semi-definite programming algorithm [3]. These methods do not consider the addition of extra constraints.

Exact methods for constrained-based clustering have been studied before. Typical is that they do not use MSS as optimisation criterion, but rather a function that is linear or quadratic. Saglam et al. [21] use an integer linear programming approach for minimizing the maximum cluster diameter. More recently, constraint programming has been used for solving constrained clustering tasks [8]. A range of constraints is supported including instance-level constraints, size of cluster constraints and constraints on the separation between clusters and maximum diameter of a cluster. As objective function the (non-normalized) sum of squared distances between clusters or maximum diameter is supported.

A large class of clustering methods are those that evaluate the quality of a cluster based on a cut-value. Also in such methods the use of column generation has been proposed [17]. The inclusion of constraints in this method may be a topic for further research.

Exact methods are also used as part of approximate constraint-based clustering methods. Demiriz et al. [11] propose to modify k-means such that the assignment step, where points are assigned to their nearest feasible cluster, corresponds to solving an LP. Constraints on minimum cluster size can be taken into account, as well as instance level constraints. Davidson et al. studied the use of SAT solvers, also using diameter as optimization criterion [10]. Müller and Kramer [19] use integer linear programming to solve constrained clustering tasks where a fixed number of candidate clusters is given upfront. The problem consists of selecting the right subset of clusters, which can be compared to solving one iteration of the restricted master problem. They investigate a number of different optimisation criterion, as well as constraints at the clustering level, such as the maximum amount of overlap between clusters or logical formula over entire clusters. These methods are not guaranteed to find globally optimal solutions.

8 Conclusions

We proposed a column generation strategy for solving the constrained MSS clustering problem. The main novelty is a branch and bound algorithm that directly solves the subproblem. Experiments showed its promise: in cases where the COP-k-means algorithm is not able to find a solution satisfying all constraints even in 500 runs, CG could find solutions and in several cases even prove their optimality.

Several open questions remain. Degeneracy was not a main concern in this study, however we observe that with the simple stabilisation scheme described in section 5 the master problem still converges very slowly. It is worth investigating if advanced stabilisation techniques work better [14]. Furthermore, the pruning strategy in the branch-and-bound algorithm could be improved and the branch-and-bound could be expanded to deal with additional constraints.

Acknowledgments. This work was supported by the European Commission under the project "Inductive Constraint Programming" contract number FP7-284715 and by the Research Foundation–Flanders by means of two Postdoc grants.

References

1. T. Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
2. D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.
3. D. Aloise and P. Hansen. A branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29:503 – 516, 12 2009.
4. D. Aloise, P. Hansen, and L. Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131(1-2):195–220, 2012.
5. K. Bache and M. Lichman. UCI machine learning repository, 2013.
6. S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications.* Chapman & Hall/CRC Press, 2008.
7. M. J. Brusco and S. Stahl. Minimum within-cluster sums of squares partitioning. In *Branch-and-Bound Applications in Combinatorial Data Analysis.* Springer, 2005.
8. T.-B.-H. Dao, K.-C. Duong, and C. Vrain. A declarative framework for constrained clustering. In *ECML/PKDD (3)*, pages 419–434, 2013.
9. I. Davidson and S. S. Ravi. The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data Min. Knowl. Discov.*, 14(1):25–61, 2007.
10. I. Davidson, S. S. Ravi, and L. Shamis. A sat-based framework for efficient constrained clustering. In *SDM*, pages 94–105, 2010.
11. A. Demiriz, K. Bennett, and P. Bradley. Using assignment constraints to avoid empty clusters in k-means clustering. In *Constrained Clustering: Algorithms, Applications and Theory.* Chapman & Hall/CRC, 2008.
12. G. Diehr. Evaluation of a branch and bound algorithm for clustering. *SIAM Journal on Scientific and Statistical Computing*, 6(2):268–284, 1985.
13. W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13(7):492–498, 1967.
14. O. du Merle, P. Hansen, B. Jaumard, and N. Mladenovic. An interior point algorithm for minimum sum-of-squares clustering. *SIAM J. Sci. Comput.*, 21(4):1485–1505, Dec. 1999.
15. D. Gondek and T. Hofmann. Non-redundant data clustering. In *ICDM*, pages 75–82, 2004.
16. R. E. Jensen. A dynamic programming algorithm for cluster analysis. *Operations Research*, 17(6):pp. 1034–1057, 1969.
17. E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62(1-3):133–151, 1993.
18. W. L. G. Koontz, P. M. Narendra, and K. Fukunaga. A branch and bound clustering algorithm. *IEEE Trans. Comput.*, 24(9):908–915, Sept. 1975.
19. M. Müller and S. Kramer. Integer linear programming models for constrained clustering. In B. Pfahringer, G. Holmes, and A. Hoffman, editors, *Proceedings of the 13th International Discovery Science DS 2010*, pages 159–173. Springer, 2010.
20. B. Os and J. Meulman. Improving dynamic programming strategies for partitioning. *Journal of Classification*, 21(2):207–230, 2004.
21. B. Saglam, F. S. Salman, S. Sayin, and M. Türkay. A mixed-integer programming approach to the clustering problem with an application in customer segmentation. *European Journal of Operational Research*, 173(3):866–879, 2006.

22. A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency.* Springer, 2003.
23. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *ICML*, pages 1103–1110, 2000.
24. Y. Xia and J. Peng. A cutting algorithm for the minimum sum-of-squared error clustering. In *SDM*, 2005.