Partition-Based Clustering using Constraint Optimization

Valerio Grossi, Tias Guns, Anna Monreale, Mirco Nanni, and Siegfried Nijssen

Abstract Partition-based clustering is the task of partitioning a dataset in a number of groups of examples, such that examples in each group are similar to each other. Many criteria for what constitutes a good clustering have been identified in the literature; furthermore, the use of additional constraints to find more useful clusterings has been proposed. In this chapter, it will be shown that most of these clustering tasks can be formalized using optimization criteria and constraints. We demonstrate how a range of clustering tasks can be modelled in generic constraint programming languages with these constraints and optimization criteria. Using the constraint-based modeling approach we also relate the DBSCAN method for density-based clustering to the label propagation technique for community discovery.

1 Introduction

Clustering [15] is the data analysis task of grouping sets of object. It is an unsupervised task, meaning that no information is known about the true grouping of the objects. In general, the goal is to find clusters whose objects are similar to each other

Valerio Grossi · Anna Monreale
University of Pisa, Largo B. Pontecorvo, 3 56127 Pisa; e-mail: vgrossi@di.unipi.it, annam@di.unipi.it

Mirco Nanni
ISTI - CNR, Via Moruzzi, 1, 56124 Pisa, Italy e-mail: mirco.nanni@isti.cnr.it

Tias Guns · Siegfried Nijssen
Department of computer science, KU Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium; e-mail: firstname.lastname@cs.kuleuven.be

Siegfried Nijssen
Leiden Institute for Advanced Computer Science (LIACS), Niels Bohrweg 1, 2333CA Leiden, The Netherlands; e-mail: s.nijssen@liacs.leidenuniv.nl

while different from the objects in the other clusters. Clustering can lead to better insights into data and to discoveries of previously unknown groupings.

Many different clustering settings have been studied in the literature. The focus of this chapter is on *partition-based clustering*. In partition-based clustering, the clustering must form a partition, that is, each object can only belong to one cluster. This is in contrast to for instance hierarchical clustering, where clusters form a tree in which one cluster can be a subset of another cluster.

An important aspect in partition-based clustering is the scoring function that is used to determine the quality of a clustering. In the literature many different methods for calculating the quality of a clustering have been proposed. A range of popular partition-based methods are based on the concept of a cluster *prototype*. Prototype-based techniques evaluate clusters based on the distance of points in the cluster to the prototype. These approaches provide clusters having spherical shapes. Other approaches consider the diameter of the clusters, or their distance to other clusters. Density-based techniques (e.g. *DBSCAN*) discover clusters of any shape, and are designed for discovering dense areas surrounded by areas with low density, typically formed by noise or outliers.

Another aspect of clustering methods is which constraints they support. Constraints can be used to specify additional requirements on the clusters that need to be found. The most well-known of such requirements are the *must-link* and *cannot-link* constraints, which specify that certain data points should or may not be clustered together [18, 3].

In this chapter, we will show that many of these clustering problems can be formalized as generic constraint optimization problems. Consequently, generic constraint optimization solvers can be used to address a wide range of clustering problems. One motivation for the use of generic constraint optimization techniques in this context is the large number of choices that need to be made in defining a clustering setting. Central questions are here:

- how do we define the coherence of a cluster?
- how do we define the number of clusters that we wish to find?
- what other properties must the clusters satisfy?

While such constraints and optimization criteria may sometimes be added in specialized techniques, generic techniques that allow for the specification of such constraints and optimiation criteria would be applicable more widely.

Within this chapter, we will distinguish two types of partitioning-based clustering settings: *direct* and *indirect* methods. These settings differ in how the number of clusters is determined. The *direct* methods require that a user specifies the number of clusters explicitly by setting a parameter k, which can be interpreted as a constraint on the number of clusters. For *indirect* methods, the number of clusters is indirectly specified through constraints on the coherence of a cluster; more clusters are created if a smaller number of clusters would not be sufficiently coherent [1].

We first discuss the direct approaches based on a parameter k (Section 2), followed by the indirect approaches (Section 3). Here, Section 2 first introduces several optimization criteria and then outlines common user-specified constraints in

clustering. Different modeling choices are presented and demonstrated on a range of clustering problems.

The section on indirect approaches (Section 2) shows how clusters can also be modeled as separated regions of high data density. This corresponds to the principle behind the *DBSCAN* algorithm. Furthermore, we draw a link between this data clustering task and the mechanism of *Label Propagation* as used in community detection in graphs.

2 Direct Methods

Characteristic for direct methods is that users need to specify the number of clusters in advance by means of a parameter k. These methods will subsequently focus on finding a good clustering with this number of clusters.

Of crucial importance is then how to evaluate the quality of one cluster. Here, several approaches are possible.

The most studied and applied approaches are those in which a cluster prototype is identified. Every cluster is represented by a prototype called the *centroid* of the cluster. Two popular algorithms employing this approach are *K-means* and *K-medoids*. In K-means each centroid represents the average of all points in the cluster, while in K-medoids the centroid is the most representative actual point in the cluster.

Other approaches do not identify an explicit prototype, but evaluate all pairwise distances between points in the cluster, or evaluate the pairwise distances between points inside and outside the cluster.

From an algorithmic perspective, most algorithms for finding clusters are heuristic. K-means and K-medoids are good examples. Given a user-specified value k, these algorithms select k initial centroids. Successively each point is assigned to the closest centroid based on a distance measure. Finally, the centroids are updated iteratively based on the points assigned to the clusters. This process stops when centroids do not change.

In this chapter, we take a step back from this algorithmic view and look at the underlying optimisation problems that clustering methods are trying to solve. We first describe the different optimization criteria that can be used, followed by constraints that can be put on clusters or the entire clusterings.

In the following, we assume given a set of data points D of size n. Each point p is represented by an m-dimensional vector. A cluster C is a set of points: $C \subseteq D$, and a clustering $\mathscr{C}$ is a partitioning of the data into clusters: $\forall C \in \mathscr{C} : C \subseteq D, \bigcup_{C \in \mathscr{C}} C = D, \forall C_1, C_2 \in \mathscr{C} : C_1 \cap C_2 = \emptyset$. Note that we consider non-overlapping clusters here.

2.1 Optimization Criteria

Intuitively, a clustering consists of clusters that are coherent and whose data points are similar to each other; on the other hand we also expect the clusters (and data points therein) to not be similar to the other clusters [15].

There are many different ways to characterize how good a clustering is, by measuring the (dis)similarity of its clusters and data points. We identify a number of these measures below. Each measure can be used as an optimisation criterium to find a 'good' clustering according to this measure.

Sum of squared inter-cluster distances. Given some distance function $d(\cdot,\cdot)$ over points, for example, the Euclidean distance, we can measure the sum of squared distances within each cluster as follows:

$$\sum_{C\in\mathscr{C}} \sum_{p,q\in C, p<q} d^2(p,q) \tag{1}$$

Here we assume that $p < q$ iff data point p is before point q in the database; this ensures that every pair of points is considered only once.

Sum of squared error to centroid. A more common approach is to measure the "error" of each cluster, that is, the distance of each point in the cluster to the mean (centroid) of that cluster.

We compute the *centroid* of a cluster by computing the mean of the data points that belong to it:

$$z_C = \text{mean}(C) = \frac{\sum_{p\in C} p}{|C|} \tag{2}$$

Here, we assume that the points p are represented as vectors and traditional vector algebra is used. The sum of squared error is then measured as:

$$\sum_{C\in\mathscr{C}} \sum_{p\in C} d^2(p,z_C) \tag{3}$$

Note that this is identical to the sum of all pairwise distances between the points of a cluster, divided by the size of that cluster: $\sum_{C\in\mathscr{C}} \sum_{p,q\in C, p<q} d^2(p,q)/|C|$.

Sum of squared error to medoids. Instead of using the mean (centroid) of the cluster, one can also use the medoid of the cluster, that is, the point that is most representative of the cluster. Let the medoid of a cluster be the point with smallest average distance to the other points:

$$y_C = \text{medoid}(C) = \arg\min_{y\in C} \sum_{p\in C} d^2(p,y). \tag{4}$$

The sum of squared error to the medoids is then measured as follows:

$$\sum_{C\in\mathscr{C}} \sum_{p\in C} d^2(p, y_C) \tag{5}$$

Cluster diameter. Another measure of coherence is to measure the diameter of the largest cluster, where the diameter is defined as the largest distance between any two points of a cluster. This leads to the following measure of maximum cluster diameter:

$$\max_{C\in\mathscr{C}} \max_{p,q\in C, p<q} d(p,q) \tag{6}$$

One can imagine other variants such as the sum of diameters.

Inter-cluster margin. The margin between two clusters is the minimal distance between any two points that belong to the different clusters. The margin gives an indication of how different the clusters are from each other (e.g. how far apart they are). This can be optimized using the following measure of minimum inter-cluster margin:

$$\min_{C,D\in\mathscr{C}} \min_{p\in C, q\in D} d(p,q) \tag{7}$$

2.2 Constraints

Using constraints for defining data mining tasks guarantees a high level of expressivity, since adding new constraints on the required output is quite easy and natural. Constraints typically specify background knowledge that the user has about the clustering. A famous example [18] is that clusters should group cars in lanes, and hence one can derive that some objects can certainly not be in the same cluster (e.g. when known to be driving side-by-side) while others certainly are (when driving in tandem).

The above example is an illustration of *instance-level* constraints, that is, constraints between specific points. *Must-link* constraints require that two points belong to the same cluster, while *Cannot-link* constraints require that two points belong to different clusters [18]. A *Must-link* constraint on two points p and q is expressed by: $\forall C \in \mathscr{C} : p \in C \leftrightarrow q \in C$; while a *Cannot-link* constraint is expressed by: $\forall C \in \mathscr{C} : p \in C \rightarrow q \notin C$.

Another type of constraints is *cluster-level* constraints [9]. The ε-constraint or maximal diameter constraint requires that the diameter of a cluster is at most ε, that is, each two points in a cluster are at most ε apart. This can also be formulated as requiring that each pair of points p and q that is further apart cannot be together in the same cluster: $\forall p,q : d(p,q) > \varepsilon \rightarrow (\forall C \in \mathscr{C} : p \in C \rightarrow q \notin C)$. The δ-constraint or minimal margin constraint requires that two points belonging to different clusters have to be at least δ apart. Alternatively formulated: any two points that are closer

than δ must belong to the same cluster: $\forall p,q:\ d(p,q) < \delta \rightarrow (\forall C \in \mathscr{C} : p \in C \leftrightarrow q \in C)$.

Other user-defined constraints can be expressed [7]. One can impose constraint on the clusters *size*, e.g. requiring clusters with a minimum or maximum number of points. Constraining the number of points to be minimum or maximum α is expressed as: $\forall C \in \mathscr{C} : |C| \geq \alpha$ and $\forall C \in \mathscr{C} : |C| \leq \alpha$.

Furthermore, any of the measures introduced in the previous section on optimization criteria can also be constrained to take a value within a certain interval. Other variants and combinations of these constraints can be employed as well, such as disjunctions of constraints or conditional must-link and cannot-link constraints. One can add constraints that certain individual clusters must be similar or different from predefined sets of points, or add *soft* constraints such as a bound on the number of points that can have a cannot-link constraint [2].

The complexity of adding constraints to (k-means) clustering has been studied in [10]. A general overview of constraint-based methods in clustering is available in the book "Constrained Clustering: Advances in Algorithms, Theory, and Applications" [3]. Furthermore, the chapter "*Data Mining & Constraints: an Overview*" of this book provides several references to using constraints in data mining tasks also for clustering.

2.3 Modeling clustering as constraint optimization

A constraint optimization problem $P = (V, D, X, f)$ consists of variables V, a domain D that lists the possible values the variables can take, a set of constraints X over V and an optimization function f over V that must be minimized or maximized.

Building on the primitives introduced earlier, many well-known clustering problems can now be modelled as follows, for a given optimization criterion $quality(\mathscr{C})$:

$$\begin{aligned}
\text{maximize}_{\mathscr{C}} \quad & quality(\mathscr{C}), && (8)\\
s.t. \quad & \\
& C_1 \cap C_2 = \emptyset \qquad \forall C_1, C_2 \in \mathscr{C} && (9)\\
& |\bigcup_{C \in \mathscr{C}} C| = n && (10)\\
& |\mathscr{C}| = k && (11)
\end{aligned}$$

Here n is the total number of points. In this setting, the number of clusters to be found is fixed and has to be k.

Note that the model above uses a set notation for the clusters. Not all constraint solvers support sets; set variables may not always be the most efficient representation either. For these reasons, an *encoding* of the sets in variables of other types is sometimes necessary. There are various ways to model these sets, as well as different solving techniques that can be used on these models. We differentiate between three kinds of approaches:

- Constraint formulations that can directly be solved by most state-of-the-art constraint programming systems;
- Formulations that require an extension of a Constraint Programming system;
- Hybrid approaches with a specialized system that can deal with a range of clustering problems, but no other problems.

We will discuss these possibilities in more detail below.

2.3.1 Constraint solving formulations

To use constraint programming systems off-the-shelf, an important question that needs to be answered is how to encode a clustering in such systems. Next to a set-based notation, several representations have been proposed. We will use these representations to construct clustering models in the next section:

- a Boolean representation, in which a variable a_{it} with domain $\{0,1\}$ takes value 1 iff point i (with $1 \leq i \leq n$) is in cluster t (with $1 \leq t \leq k$), and takes value 0 otherwise. Constraints
$$\sum_{t=1}^{k} a_{it} = 1$$
for all points i ensure that a point is in only one cluster [14];
- a Boolean representation, in which a variable a_t with domain $\{0,1\}$ takes value 1 iff possible cluster t (with $1 \leq t \leq 2^n$, i.e. each possible cluster is given an index) is in the clustering; constraint $\sum_{t=1}^{2^n} a_t = k$ ensures exactly k possible clusters are selected and constraints
$$\sum_{t=1}^{2^n} [i \in C_t] a_t = 1$$
ensure that every point i is in exactly one chosen cluster; here $[i \in C_t]$ is an indicator that takes value 1 iff point i is in possible cluster t [16, 11];
- an integer representation, in which a variable a_i with domain $\{1,\ldots,k\}$ indicates that point i (with $1 \leq i \leq n$) is in cluster a_i [8];
- an integer representation, in which a variable g_i with domain $\{1,\ldots,n\}$ identifies the point with the smallest index that is in the same cluster as point i; note that $g_i = i$ iff there is no point $j < i$ in the same cluster as point i [7].

An important benefit of the first Boolean representation is that it is easy to formalize additional constraints in this representation. A *Must-link* constraint on two points p_i and p_j is expressed by a set of $a_{it} = a_{jt}$ constraints, where $1 \leq t \leq k$; a *cannot-link* constraint is expressed by: $\forall t \in \{1,\ldots,k\} : a_{it} + a_{jt} \leq 1$. A *size* constraint can be expressed by:

$$\forall t \in \{1,\ldots,k\} : \sum_{i=1}^{n} a_{it} \geq \alpha$$

$$\forall t \in \{1,\ldots,k\} : \sum_{i=1}^{n} a_{it} \leq \alpha.$$

A drawback of this representation is that additional constraints are required to ensure that a point is not in two clusters; this is not necessary in the integer representations.

The second Boolean representation has as most important drawback that its number of variables is very large. One way to address this is to limit the number of possible clusters apriori; ideas for this were presented in [16].

The main difference between the integer representations is that in the second representation the indexes of representative points are used to identify clusters, while in the first cluster indexes are used. In the second representation the number of clusters is not fixed; to achieve a fixed number of clusters, additional variables c_j with domain $\{1,\ldots,n\}$, where $1 \leq j \leq k$, can be used, with the constraints:

- $g_{c_j} = c_j$ for all clusters j, i.e., variable c_j points to the identifying point for each cluster;
- $\sum_{j=1}^{k} [g_i = c_j] = 1$ for each point i; this ensures that each point i also belongs to one of the k clusters identified by c.

A remaining challenge is how to represent the optimization criterion. In many cases, additional variables are needed. This is illustrated for a number of cases below.

K-medoid clustering

In $k-$medoid clustering, an important aspect is that we need to identify the cluster medoids. One approach is to represent these medoids using Boolean variables m_{ij}, where $1 \leq i \leq n$ and $1 \leq j \leq k$; these variables indicate whether a point is the medoid of a cluster or not. Constraints enforce that each cluster has only one medoid.

The optimization criterion then becomes:

$$\sum_{i=1}^{n} \sum_{j=k}^{n} a_{ij} \sum_{h=1}^{n} m_{hj}\, d(p_i, p_h)^2$$

This leads to the overall optimization problem below:

$$\underset{a,m}{\text{minimize}} \quad \sum_{i=1}^{n} \sum_{j=k}^{n} a_{ij} \sum_{h=1}^{n} m_{hj}\, d(p_i, p_h)^2 \tag{12}$$

$$s.t.$$

$$\sum_{j=1}^{k} a_{ij} = 1 \qquad \forall i \in \{1,\ldots,n\} \tag{13}$$

$$\sum_{i=1}^{N} m_{ij} = 1 \qquad \forall j \in \{1,\ldots,k\} \tag{14}$$

Hence, this model assumes that both the *assignment of points to clusters* and *the actual medoids* are discovered by the constraint programming system. Note that this model does not explicitly constrain the medoid to its cluster, as in an optimal

solution, the chosen centers need to be medoids for their cluster in order to minimize the optimization criterion.

Furthermore, this model does not impose additional constraints. Constraints such as those discussed in Section 2.2 can be added to the model without modification.

Sum of squared inter-cluster distances

This clustering setting has been modeled in constraint programming using the integer representation where each variable g_i points to the 'identifying' point of the cluster, e.g. its point with smallest index [7]. The sum of squared inter-cluster distances is then expressed as:

$$\sum_{i,j\in\{1,\ldots,n\},i<j} [g_i = g_j] d^2(p_i, p_j)$$

Using variable c_j to represent the identifying point of cluster j, where the identifying point is the point with smallest index, this leads to the following constraint specification:

$$\underset{g,c}{\text{minimize}} \quad \sum_{i,j\in\{1,\ldots,n\},i<j} [g_i = g_j] d^2(p_i, p_j), \tag{15}$$

$$s.t.$$

$$g_i \leq i \qquad \forall i \in \{1,\ldots,n\} \tag{16}$$

$$g_{c_j} = c_j \qquad \forall c \in \{1,\ldots,k\} \tag{17}$$

$$\sum_{j=1}^{k} [g_i = c_j] = 1 \qquad \forall i \in \{1,\ldots,n\} \tag{18}$$

$$c_j < c_{j'} \qquad \forall j, j' \in \{1,\ldots,k\}, j < j' \tag{19}$$

$$c_1 = 1 \tag{20}$$

Equation 16 ensures that either it is the smallest (identifying) point of its cluster, or g_i points to another (smaller) identifying point. Equation 17 materializes the concept of identifying point in a variable c_j. The identifying point's index is the cluster identifier, so $g_{c_j} = c_j$. This constraint is known in the constraint solving literature as an element constraints. The last two constraints are symmetry breaking constraints.

Maximal diameter and minimal margin

The same integer representation with identifying points has been used to model the problem of minimizing the maximal diameter and maximizing the minimal margin [7].

The main difference is the optimization criterion. Instead of computing the maximal diameter or minimal margin explicitly and optimising this, it is possible to constrain each pair of points individually. Let D be a new variable representing the maximum diameter, then each pair of points p_i, p_j that is further than d apart may

not be in the same cluster: $d(p_i, p_j) > D \rightarrow (g_i \neq g_j)$. The model is shown below and shares a number of constraints with the previous model [11]:

$$\underset{D,g,c}{\text{minimize}} \quad D, \tag{21}$$

$$s.t.$$

$$d(p_i, p_j) > D \rightarrow (g_i \neq g_j) \qquad \forall i, j \in \{1, \ldots, n\} \tag{22}$$

Equations 16...20 in the previous model

Maximizing the minimal margin is specified in a similar way [11]:

$$\underset{M,c,g}{\text{maximize}} \quad M, \tag{23}$$

$$s.t.$$

$$d(p_i, p_j) < M \rightarrow (g_i = g_j) \qquad \forall i, j \in \{1, \ldots, n\} \tag{24}$$

Equations 16...20 in the above model

Squared error to the centroids (K-means)

K-means aims to find non-overlapping clusters that minimize the sum of squared errors to the centroid of the cluster. As pointed out earlier, one formulation of the optimization criterion is $\sum_{C \in \mathscr{C}} \sum_{i,j \in |C|, i<j} \frac{d^2(p_i, p_j)}{|C|}$. While we could model this with the Boolean or integer representations used so far, the division in this optimization criterion creates a non-linearity that makes the problem a lot harder to solve.

Instead, we can use the approach in which we have 2^n Boolean variables a_t, i.e., a variable for each possible cluster. Let m be an n by 2^n matrix of Boolean values, where each column is a cluster with $m_{it} = 1$ if data point p_i is in cluster t and $m_{it} = 0$ otherwise. For each cluster t, the squared error to the centroid can then be precomputed as

$$c_t = \frac{\sum_{i=1}^{n} m_{it} \sum_{j=i+1}^{n} m_{jt} d^2(p_i, p_j)}{\sum_{i=1}^{n} m_{it}}$$

Using these costs, the problem can be formulated as follows [11]:

$$\underset{a}{\text{minimize}} \quad \sum_{t \in T} a_t c_t, \tag{25}$$

$$s.t.$$

$$\sum_{t \in T} a_t m_{it} = 1 \qquad \forall i \in \{1, \ldots, n\} \tag{26}$$

$$\sum_{t \in T} a_t = k \tag{27}$$

where $T = \{1, \ldots, 2^n\}$ denotes all possible clusters. Equation 25 is the sum of the squared errors to the centroid of all clusters that are selected (e.g. $a_t = 1$). Equation

26 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 27 finally ensures that exactly k clusters are found.

2.3.2 Extending constraint solvers

The previous subsection introduced how to model many clustering problems using generic constraint programming formulations. These formulations can be decomposed into low-level constraints such as sum, (in)equality and implication. Such constraints are supported by most CP systems.

While correct, however, these decompositions and the corresponding *propagation* of the low-level constraints is often not efficient. To improve the performance one of the possible approaches is to add *global constraints* to these CP systems, which implement specialized propagation methods for specific constraints.

For example, Thi-Bich-Hanh et al [6] introduced a global constraint for the sum of squared inter-cluster distances:

$$\sum_{i,j\in\{1,\ldots,n\},i<j} [g_i = g_j] d^2(p_i, p_j) \tag{28}$$

In a standard constraint solver, this constraint is decomposed by introducing auxiliary variables $b_{ij} \leftrightarrow [g_i = g_j]$ and having a linear sum constraint over these b_{ij}: $s = \sum_{i,j\in\{1,\ldots,n\},i<j} b_{ij} v_{ij}$, where $v_{ij} = d^2(p_i, p_j)$ are precomputed constants.

Instead, the authors introduce a global constraint for the entire Equation 28, which can reason over the fact that each point can only belong to one cluster. In this way, a tighter lower bound on the sum can be computed than when using the standard decomposition.

Computing this tighter lower bound is achieved by splitting the sum into three distinct cases: a) cases for which g_i and g_j are already assigned, b) cases for which g_i or g_j is assigned, but not the other one, and c) cases for which both g_i and g_j are not assigned. Case a) can be deterministically computed. For case b), for each point, the minimum value is chosen among all existing clusters to which this point could be added. For case c) a clever heuristic is used to compute a lower bound based on the minimum number of possible connections that must still be added to obtain k clusters.

Apart from adding global constraints, efficiency improvements can typically also be obtained by adding redundant constraints or by breaking symmetries in the constraint formulation. Another important aspect is the order in which to search over the variables and their possible values. For example, one could use a furthest-point-first heuristic [7, 13].

2.3.3 Hybrid approach: column generation

Further problems of efficiency are posed by models that introduce an exponential number of variables, having one variable for each potential cluster. Global constraints can not solve this problem.

One approach to solve this challenge is to lazily add candidate clusters until the optimal subset of clusters is found. This is the idea behind *column generation* in Integer Linear Programming. This was first investigated for minimum sum of squared error clustering by DuMerle et al [11] and later extended to support additional constraints by Babaki et al [2].

The core idea is to only consider a subset of the clusters in the set T of the above model, and to *relax* the a_t variables such that they can take on real values instead of being Boolean. This problem is called the *restricted master problem*. Solving the restricted master problem can be done with standard (integer) linear programming solvers, and one obtains a real-valued solution to a_t. Then, using the dual values of this solution, one can search for the best cluster (column) to add, that is, the cluster that can best improve the objective function. This is called the *subproblem* and is typically done with a specialized method. If no such column can be found, the solution of the restricted master problem is also a solution of the original master problem [11].

A key observation in [2] is that most constraints considered in constraint-based clustering are constraints on individual clusters. Consequently, these constraints do not change the (restricted) master problem; they only affect the set T and hence the definition of the *subproblem*. Babaki et al. [2] have devised a method to solve the subproblem directly while taking must-link and cannot-link constraints into account, as well as other anti-monotone constraints such as cluster size and overlap constraints.

3 Indirect Methods

The approaches presented in Section 2 require to know in advance the number of clusters to be found. Moreover, they tend to provide clusters that are sphere-shaped. Unfortunately, in a number of real applications, the data points are grouped into non-spherical regions or regions that are quite dense surrounded by areas with low density, typically formed by noise. From this perspective, clusters can also be defined implicitly as regions of higher data density, separated from each other by regions of lower density. The price for this flexibility is a difficult interpretation of the obtained clusters. One of the most famous clustering algorithms based on the notion of density of regions is DBSCAN [12]. This algorithm does not rely on an optimization algorithm however, and in this chapter we present a constraint programming formulation (Section 3.1). Using this formulation, we show how re-defining this task as a community discovery problem in a network, this approach becomes very similar to the label propagation approach that finds clusters of nodes in networks [17].

3.1 Density-based Clustering

Density-based clustering is based on measuring the data density at a certain region of the data space and then defining clusters as regions that exceed a certain density threshold. The final clusters are obtained by connecting neighboring dense regions. Figure 1 shows an illustrative example in a two-dimensional space. Four groups are recognized as clusters and they are separated by an area where the data density is low.

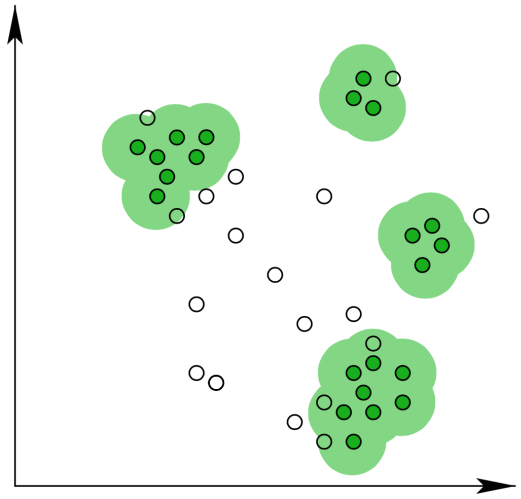

Fig. 1 Example of density-based clusters [4].

DBSCAN. DBSCAN [12] locates regions of high density that are separated from one another by regions of low density. The approach identifies three different classes of points:

> **Core points.** These points are in the interior of a density-based cluster. A point is a core point if the number of points within a given neighborhood around the point as determined by the distance function and a user- specified distance parameter, ε, exceeds a certain threshold, $MinPts$, which is also a user-specified parameter.
> **Border points.** These points are not core points, but fall within the neighborhood of a core point. A border point can fall within the neighborhoods of several core points.
> **Noise points.** A noise point is any point that is neither a core point nor a border point.

The DBSCAN algorithm works as follows:

1. Label all points as core, border, or noise points.
2. Eliminate noise points.

3. Put an edge between all core points that are within ε of each other.
4. Make each group of connected core points into a separate cluster.
5. Assign each border point to one of the clusters of its associated core points.

Below we introduce the constraint programming model for this clustering problem.

Constraint Programming Model for Density Based clustering

We reformulate the problem in the context of networks by considering the set of points D as nodes and setting an edge between two nodes i and j if the distance between the two points is less than a given ε. Clearly, in this way we have that the neighbors of a node (point) i are the set of points within a distance ε. Our intended objective is to capture the basic idea that "each node has the *same* label as all of its neighbors". Therefore, the problem can be modelled as follows:

$$\text{maximize}(\sum_{j\in L}\min(1,\sum_{i\in D}k_{i,j})), \tag{29}$$

$s.t.$

$$a_{i,j}=\begin{cases}1 \text{ if } d(i,j)\leq\varepsilon\\ 0 \text{ otherwise}\end{cases} \qquad \forall i,j\in D \tag{30}$$

$$k_{i,j}\in\{0,1\} \qquad \forall i\in D,\forall j\in L \tag{31}$$

$$r_i=\begin{cases}1 \text{ if } \sum_{j\in D}a_{i,j}\geq minp\\ 0 \text{ otherwise}\end{cases} \qquad \forall i\in D \tag{32}$$

$$\sum_{j\in L}k_{i,j}=1 \qquad \forall i\in D \tag{33}$$

$$r_h=1\wedge r_p=1\wedge a_{h,p}=1\wedge k_{h,j}=1\Rightarrow k_{p,j}=1 \qquad \forall h\in D,\forall r\in D,\forall j\in L \tag{34}$$

$$r_i=0\Rightarrow k_{i,j}=1 \qquad \forall i\in D \tag{35}$$

where

$$j=\min(\{j\in L\setminus\{n+1\}\mid \exists h: a_{h,i}=1\wedge r_h=1\wedge k_{h,j}=1\}\cup\{n+1\})$$

In more detail this model can be described as follows. Boolean variables $k_{i,j}$ denote the color of a point (node i), by setting $k_{i,j}=1$ if the point i has the color j. Variables $a_{i,j}$ indicate the presence or absence of an edge between two nodes. Variables r_i denote whether node i is a core point or not.

Assumed given are: (a) the set of points D, and (b) the ordered set of colors $L=\{1,\ldots,n\}$. Note that in the set of colors we have the color $n+1$ that is an additional color used for coloring the noise points. The model imposes that a point has one and only one color, and that all the connected core points must have the same color (Equation 34). Another requirement is that each point that is not a core point takes the same color of the core points that are connected to it. If it does not have any core point around it then this point takes the additional color $n+1$ because it is a noise (Equation 35). Such a constraint also captures the special case in which the point i can be connected to more than one core with different colors. In this case,

the model assigns to i the color of the core that in the ordered set $C \setminus \{n+1\}$ has a lower rank. Finally, the model is intended to maximize the number of different colors. Notice that a solution where all points have distinct colors does not satisfy Equation 34 because connected points do not have the same color.

This constraint programming formulation makes it easy to extend the standard method with other constraints. In principle, any constraint that requires to merge clusters as identified by DBSCAN can be added to the model above. As an example, we could specify constraints on the minimum cluster size: in this case, clusters will need to be merged in order to obtain a required cluster size; by enforcing a diameter constraint on the resulting clusters, it can be ensured that the resulting clusters are not arbitrary combinations of clusters as identified by traditional DBSCAN.

Moreover, we can also extend the above problem by changing one of the constraints of the standard formulation. In the following, we will show that by changing a constraint of the DBSCAN formulation we obtain a problem that corresponds to one of the most famous algorithms for discovering communities in network data.

3.2 Label Propagation

When considering graph or network data, a task very similar to clustering is *community discovery*, which can be seen as a network variant of standard data clustering. The concept of a "community" in a (web, social, or informational) network is intuitively understood as a set of individuals that are very similar, or close, to each other, more than to anybody else outside the community [5]. This has often been translated in network terms into finding sets of nodes densely connected to each other and sparsely connected with the rest of the network. An interesting community discovery algorithm is the Label Propagation algorithm [17] that detects communities by spreading labels through the edges of the graph and then labeling nodes according to the majority of the labels attached to their neighbors, iterating until a general consensus is reached.

Before introducing a constraint programming model for this algorithm we recall the details of the iterative label propagation algorithm presented in [17].

Iterative Label Propagation (LP). Suppose that a node v has neighbors $v_1, v_2, \ldots, v_k$ and that each neighbor carries a label denoting the community that it belongs to. Then, v determines its community based on the labels of its neighbors. [17] assumes that each node in the network chooses to join the community to which the maximum number of its neighbors belong to. As the labels propagate, densely connected groups of nodes quickly reach a consensus on a unique label. At the end of the propagation process, nodes with the same labels are grouped together as one community. Clearly, a node with an equal maximum number of neighbors in two or more communities will take one of the two labels by a random choice. For clarity, we report here the procedure of the LP algorithm. Note that, in the following $C_v(t)$ denotes the label assigned to the node v at time (or iteration) t.

1. Initialize the labels at all nodes in the network. For any given node v, $C_v(0) = v$.
2. Set $t = 1$.
3. Arrange the nodes in the network in a random order and set it to V.
4. For each $v_i \in V$, in the specific order, let $C_{v_i}(t) = f(C_{v_{i1}}(t-1), \ldots, C_{v_{ik}}(t-1))$. Function f here returns the label occurring with the highest frequency among neighbors and ties are broken uniformly randomly.
5. If every node has a label that the maximum number of its neighbors has, or t hits a maximum number of iterations t_{max} then stop the algorithm. Else, set $t = t + 1$ and go to (3).

The drawback of this algorithm is the fact that *ties are broken uniformly randomly*. This random behavior can lead to different results for different executions and some of these results cannot be optimal. In Figure 2 we show how given the same network as input of the LP algorithm we obtain four different results.

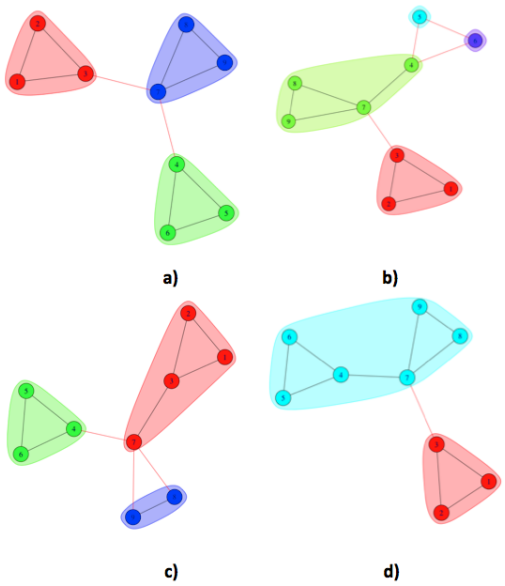

Fig. 2 The result of four executions of LP Algorithm

In the next section we propose a constraint programming model that solves this problem by providing the optimal solution.

3.2.1 Constraint Programming Model for Label Propagation

Let us now propose a constraint programming model for the community discovering problem based on label propagation. Our aim is to capture the basic idea that "each node takes the label of the majority of its neighborhood". Therefore, the model is the following:

$$\text{maximize}(\sum_{j\in L}\min(1,\sum_{i\in N}k_{i,j})), \tag{36}$$

$$s.t.$$

$$a_{i,j}=\begin{cases}1 \text{ if } (i,j)\in E\\ 0 \text{ otherwise}\end{cases} \tag{37}$$

$$k_{i,j}\in\{0,1\} \tag{38}$$

$$\sum_{j\in L}k_{i,j}=1 \qquad \forall i\in N \tag{39}$$

$$n_{i,h}=\sum_{\forall j:a_{i,j}=1}k_{j,h} \qquad \forall i\in N,\forall h\in L \tag{40}$$

$$k_{i,l}=1\Rightarrow n_{i,l}=\max_{h\in L}n_{i,h} \qquad \forall i\in N,\forall l\in L \tag{41}$$

Here, variables $a_{i,j}$ indicate the presence or absence of an edge between two nodes. Variables $k_{i,j}$ denote the color (label) of a node in the network. Assumed given is (a) the set of nodes N, (b) the set of edges E, and (c) an ordered set of colors L. A node can be assigned one and only one color. Variables $n_{i,h}$ denote the number of neighbors of node i with assigned color h. The model assigns to the node i the color h if it is the most popular among its neighbors, as shown in Equation 41. Such a constraint also captures the case of ties. In such a case, node i is assigned the color that has the lowest rank in the ordered set L. Finally, the model maximizes the number of different colors in the network, as shown in Equation 36.

This model highlights the similarity between Label Propagation and the Density-based clustering problem, and thanks to the constraint programming formulation we can note that the model for density-based clustering is a variant of the standard label propagation. Indeed, the only difference is due to the fact that the Density-based model requires that "each node has the *same* label of all its neighbors", and not the *most frequent* label. Equations 34 and 35 in the DBSCAN model and Equation 41 in the LP model express this difference. By executing our model we obtain the optimal solution depicted in Figure 3, where we consider as input the same network in Figure 2.

4 Conclusions

In this chapter, we have presented how different well-established approaches to partition-based clustering can be modeled and optimized via constraints. In particular, we investigated two main families of partition-based methods, i.e. *direct*

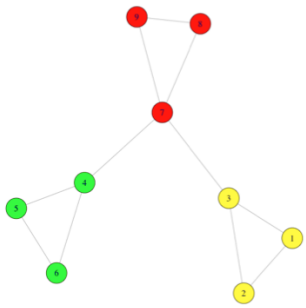

Fig. 3 The result of the execution of CP-LP model

and *indirect*. In this perspective, the chapter has presented several examples where the clustering methods are explicitly modeled by constraints. In this way, it has parted from the more traditional *algorithmic* view on clustering. We discussed different optimization criteria and constraints, showed different modeling choices for *direct* methods and related the *indirect* methods of DBSCAN and label propagation through a constraint formulation.

References

1. C. C. Aggarwal and C. K. Reddy. *Data Clustering: Algorithms and Applications*. Chapman & Hall/CRC, 1st edition, 2013.
2. B. Babaki, T. Guns, and S. Nijssen. Constrained clustering using column generation. In H. Simonis, editor, *Lecture Notes in Computer Science, 11th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming (CPAIOR 2014), Cork, Ireland, 19-23 May 2014*, pages 438–454. Springer, 2014.
3. S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC, 1 edition, 2008.
4. M. R. Berthold, C. Borgelt, F. Hppner, and F. Klawonn. *Guide to Intelligent Data Analysis: How to Intelligently Make Sense of Real Data*. Springer Publishing Company, Incorporated, 1st edition, 2010.
5. M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining*, 4(5):512–546, 2011.
6. T. Dao, K. Duong, and C. Vrain. A filtering algorithm for constrained clustering with within-cluster sum of dissimilarities criterion. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pages 1060–1067, 2013.
7. T.-B.-H. Dao, K.-C. Duong, and C. Vrain. A declarative framework for constrained clustering. In *ECML/PKDD (3)*, pages 419–434, 2013.
8. T.-B.-H. Dao, K.-C. Duong, and C. Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, pages –, 2015.
9. I. Davidson and S. Ravi. The complexity of non-hierarchical clustering with instance and cluster level constraints. *Data Mining and Knowledge Discovery*, 14(1):25–61, 2007.

10. I. Davidson and S. S. Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*, pages 138–149, 2005.
11. 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.
12. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In E. Simoudis, J. Han, and U. M. Fayyad, editors, *KDD*, pages 226–231. AAAI Press, 1996.
13. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.
14. P. Hansen and D. Aloise. A survey on exact methods for minimum sum-of-squares clustering. *http://www.math.iit.edu/Buck65files/msscStLouis.pdf*, pages 1–2, Jan. 2009.
15. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.
16. M. Mueller and S. Kramer. Integer linear programming models for constrained clustering. In *Discovery Science - 13th International Conference, DS 2010, Canberra, Australia, October 6-8, 2010. Proceedings*, pages 159–173, 2010.
17. U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(2):036106+, 2007.
18. K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 1103–1110, 2000.