A Constraint Programming Approach for Clustering

Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain

{*thi-bich-hanh.dao, khanh-chuong.duong, christel.vrain*}*@univ-orleans.fr*

LIFO, University of Orléans
45067 Orléans cedex 02, France

November 4, 2012

Abstract

In this paper, we address the problem of constraint-based clustering based on Constraint Programming (CP). We consider the problem of optimizing the maximum diameter of the clusters. We propose a model for this task in CP and we also show the importance of search strategies for improving the efficiency. Our model is based on the distance between objects, thus allowing to consider qualitative and quantitative data. Cluster-based and instance-based constraints can be directly added. Experiments on classical datasets show the interest of our approach.

1 Introduction and Motivation

Cluster analysis has been extensively studied for many years, with well-known approaches like k-means, k-medoids, etc. In general, the problem consists in partitioning a set of n objects into k non-empty disjoint clusters. Clustering is a difficult field of research for several reasons: the definition of the underlying distance, the choice of the criterion to optimize, the size of the search space and as a consequence the complexity and the need to define heuristics, often lead to a local optimum. Constraints put on the solution allow to better model real-world applications and as a consequence they allow to restrict the size of the search space. Nevertheless, most of the classical clustering algorithms have not been developed for constraint-based clustering and have to be adapted, when possible, to handle constraints. Developing general solvers applicable to a wide range of clustering problems is a challenging task.

On the other hand, recent advances in Constraint Programming (CP) make this framework much more powerful to solve Data Mining problems. Several works ([4, 3, 1]) have investigated the use of CP for modeling Data Mining problems and they have shown the interest of declarativity inherent to CP. In this paper, we propose a framework for modeling constraint-based clustering in CP. We present a new model for *distance-based cluster problem* with both instance-level and cluster-level constraints. The number k

of classes is fixed (in theory no bound is given on k but the larger k, the higher the complexity). The model takes as input a matrix of distances between objects, and thus can deal with qualitative and quantitative databases. For the time being we focus on the criterion of minimizing the maximum diameter. We show that our framework can be directly extended to instance-level and cluster-level constraints. Although the choice of the model is fundamental, we also emphasize on the fact that the search strategy is really important for improving efficiency. Several strategies are studied, differing on the way points are ordered or on the use of theoretical bounds. Experiments on classical databases show the interest of the approach. Our framework allows to find a global optimum, we compare our system with the FPF method introduced by Gonzalez [6], which is a very efficient method for diameter-based clustering, but is a greedy algorithm and leads to an approximate solution.

Recent works relying on CP ([7], [10]) consider the conceptual clustering problem in a framework for k-pattern set mining problem on qualitative databases. The problem consists in finding k frequent patterns which do not overlap and cover all the transactions. Transactions can be viewed as objects and patterns as clusters, two objects are similar if they share the same pattern. Some criterion functions are considered such as maximizing the minimum size of cluster or minimizing the difference between cluster sizes. Our approach considers distance between objects and is capable to proceed on qualitative or quantitative databases. A SAT framework is introduced in [2] for constraint-based clustering: it uses instance-level constraints (must-link and cannot-link) and cluster-level constraints (diameter of clusters, separation between clusters). This algorithm is guaranteed to converge to a global optimum but it works only with 2-cluster problems. Our approach based on CP is more general since the number of clusters is not limited to 2.

The paper is organized as follows. In Section 2, we present preliminary notions on constraint-based clustering and on CP. Section 5 is devoted to the presentation of our model and Section 6 to experiments. Conclusion and discussion on future works are given in Section 7.

2 Preliminaries

2.1 Definition of Clustering

Clustering is the process of grouping data into classes or clusters, so that objects within a cluster have high similarity but are very dissimilar to objects in other clusters.

More formally, let us consider a database of n objects $\mathcal{O} = \{o_1, \ldots, o_n\}$, belonging to a space $\mathcal{X}$ and a dissimilarity (or a distance) measure $d(o_i, o_j)$ between two objects o_i and o_j of $\mathcal{O}$, the aim is to find a structure grouping the objects and optimizing a given criterion. Clustering can therefore be considered as an optimization problem.

The objets are usually described by attributes, which may be qualitative or quantitative attributes. In case of quantitative attributes, the Euclidean measure is classically

used. It is defined by

$$d(o_i, o_j) = \sqrt{\sum_{t=1}^{m} (x_{it} - x_{jt})^2} \tag{1}$$

where $o_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ and $o_j = (x_{j1}, x_{j2}, \ldots, x_{jm})$ are two m-dimensional objects.

In some cases, the system is only provided with a matrix given the dissimilarity between pairs of objects.

There are many variations of the clustering problem. Several points of view can be used to classify clustering methods, such as the form of the learned structure (partition, hierarchy, overlapping or not, ...), or the criterion to optimize. In this paper we are interested in partitioning methods that learn a partition of the set of objects. It means finding a set of k classes $C_1, \ldots, C_k$ such that: (1) for all i, $C_i \neq \emptyset$, (2) $\cup_i C_i = \mathcal{O}$, (3) for all i, j such that $i \neq j$, $C_i \cap C_j = \emptyset$, and (4) a criterion E is optimized. Optimized criteria may be, among others:

- Least-square criterion

$$E = \sum_{c=1}^{k} \sum_{o_i \in C_c} d(m_c, o_i)^2 \tag{2}$$

  where m_c is the center of each cluster C_c.

- Intra-class variance criterion

$$E = \sum_{c=1}^{k} \sum_{o_i, o_j \in C_c} d(o_i, o_j)^2 \tag{3}$$

  It is equivalent with least-square criterion when the Euclidean distance is used.

- Absolute-error criterion

$$E = \sum_{c=1}^{k} \sum_{o_i \in C_c} d(o_i, r_c) \tag{4}$$

  where r_c is a representative object of the cluster C_c.

- Diameter-based criterion

$$E = max_{c \in [1,k]}(max_{o_i, o_j \in C_c} d(o_i, o_j)))$$

  E is the maximum diameter of the clusters, where the diameter of a cluster is the maximum distance between any two of its objects.

2.2 Partitioning methods

In this section, we present most common partitioning methods: k-means, k-medoids and FPF [6]

- k-means[8]: The k-means algorithm is a simple iterative method to partition a dataset into k clusters. The algorithm is initialized by picking k objects from $\mathcal{O}$ as the k initial cluster representatives or centroids. Then each object is assigned to the closest centroid, forming a partition of the dataset. Next, each representative is relocated by the mean value of all objects assigned to it. Using the new cluster representatives, objects are assigned again. This process iterates until there is no change of assignments of objects, or equivalently, until there is no change of centroids. The k-means algorithm always converges to a solution where the centroids are stable.

  The algorithm aims at minimizing the Sum of Squared Error (SSE) as defined in Equation (2). In each iteration of the algorithm, the SSE reduces and converges to a local optimum. This method is efficient if clusters are separated from each other. Now we will show how the update of cluster centroids minimizes the SSE, specialized for one-dimensional data. It is shown in [12] that: the k^{th} centroid of the cluster m_k minimizes the SSE, by setting the differentiating of the SSE to 0 and solving:

$$\begin{aligned}
\frac{\partial}{\partial m_k} SSE &= \frac{\partial}{\partial m_k} \sum_{c=1}^{k} \sum_{o_i \in C_c} d(m_c, o_i)^2 && (5)\\
&= \frac{\partial}{\partial m_k} \sum_{c=1}^{k} \sum_{o_i \in C_c} (m_c - o_i)^2 && (6)\\
&= \sum_{c=1}^{k} \sum_{o_i \in C_c} \frac{\partial}{\partial m_k} (m_c - o_i)^2 && (7)\\
&= \sum_{o_k \in C_k} 2 * (m_k - o_k) = 0 && (8)\\
& && (9)
\end{aligned}$$

$$\sum_{o_k \in C_k} 2 * (m_k - o_k) = 0 \Rightarrow |c_k| m_k = \sum_{o_k \in c_k} o_k \Rightarrow m_k = \frac{1}{|c_k|} \sum_{o_k \in c_k} \quad (10)$$

  So, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

  When the Euclidean distance is used, the SSE is equivalent to the intra-class variance defined in Equation (3), since by simple manipulation in [9], the SSE

criterion may be written as:

$$SSE = \frac{1}{2}\sum_{c=1}^{k} |C_c|\overline{S_c} \tag{11}$$

where:

$$\overline{S_c} = \frac{1}{|C_c|^2}\sum_{o_i,o_j \in C_c} \|o_i - o_j\|^2 \tag{12}$$

Algorithm 1 k-Means algorithm

1: Generate randomly k objects as initial cluster centers
2: **repeat**
3: Assign each object to the cluster with the closest center.
4: Re-calculate the new center of each cluster as the mean value of the objects in each cluster
5: **until** no change of cluster center

The k-means is efficient if clusters are separated from each other. It can handle large dataset because the complexity of the algorithm is $O(nkt)$ where t is the number of iterations. However, this method can be applied only when the mean value of clusters can be computed. Moreover, this method is sensible to noises and outliers because those values affect the mean value of clusters.

- k-medoids[8]: In this method, an actual object is chosen in each cluster as a representative object. This differs from k-means where the representattive is computed as the mean value of the objets. The objective of k-medoids is minimize the sum of the dissimilarities between each object and its corresponding representative object. That is, it uses the absolute-error criterion defined in Equation (4). It starts by choosing k objects as the initial cluster representative. Each object is assigned to its nearest representative object. It then chooses new representative object for each cluster allowing to minimizing the absolute error criterion. With the new representative, the method reassigned the objects. The algorithm iterates until each representative object is actually the medoid, or most centrally located object of its cluster. It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

  The method FPF (Furthest Point First) introduced in [6] uses the maximum diameter criterion. In this method, each cluster has an object as the head and every objects in a cluster are closer to their head than other heads. The method starts by choosing an object as the head of the first cluster and assigns all objects to it. In the next iteration, the furthest object from the first head is chosen as the head of the second cluster. Any object which is closer to the second head than the first one will be moved to the second cluster. The algorithm iterates: choosing the

Algorithm 2 k-Medoids algorithm for partitioning

Randomly generate k objects as initial cluster representatives
repeat
  Assign each object to the cluster with the closest representative.
  Randomly select a non representative object o_i
  Compute the cost S when swap the object o_i with a representative o_j
  if $S < 0$ **then**
    swap o_i with o_j, o_i is now a new representative
  end if
until no change of cluster representatives

object that is furthest to its head as the head of the new cluster and reassigning objects. It stops after k iterations since k clusters are needed. The time complexity is $O(kn)$. Let d_{opt} be the global optimal, then this heuristic is guaranteed to find a clustering with a maximum diameter d such that $d_{opt} \leq d \leq 2d_{opt}$ if the distance satisfies the triangle inequality. Moreover, Gonzalez showed that this approximation ratio is tight: finding d such that $d \leq (2-\epsilon)d_{opt}$ is NP-hard for all $\epsilon > 0$ when objects are in a three dimensional space.

Algorithm 3 FPF algorithm for partitioning

Randomly select an object, mark it as $head_1$, the head of cluster 1
for i from 2 to k **do**
  Select the object that is furthest to its head, mark it as $head_i$
  Assign every object o to cluster i if o is closer to $head_i$ than to its head
end for

- DBSCAN [5]: Most partitioning methods find spherical-shaped clusters. The method DBSCAN is based on the notion of density, which allows to discovery arbitrary shaped clusters. In this method, the number of k is not given, but additional parameters defining the density must be specified. DBSCAN requires two input parameters $MinPts$ and Eps and its complexity is O(n^2). It discovers clusters with arbitrary shape but with the same density. This approach is based on the following notions:

- The Eps neighborhood of a point p is defined by:

$$N_{Eps}(p) = \{q \in \mathcal{O} | dist(p,q) \leq Eps\}$$

- Core point: It has at least $MinPts$ points within a radius Eps. A point p is a core point iff $|N_{Eps}(p)| \geq MinPts$

- Border point: It doesn't have at least $MinPts$ points within a radius Eps but it is the neighborhood of one or more core points. A point p is a border point iff $|N_{Eps}(p)| < MinPts$ and $\exists q : p \in N_{Eps}(q), |N_{Eps}(q)| \geq MinPts$

- Noises: Point which is neither a core point nor border point.

- Directly Density Reachable: A point p is directly density reachable from a point q if q is a core point and p is the neghborhood of q:

$$p \in N_{Eps}(q), |N_{Eps}(q)| \geq MinPts$$

- Density Reachable: A point p is density reachable from a point q if there is a chain of point $p_1, \ldots, p_n$, such that $p_1 = q, p_n = p$ and p_{i+1} is directly density reachable from p_i.

- Density connected: A point p is density connected to a point q if there is a point o such that both p and q are density reachable from o

  The method DBSCAN starts with an abitrary point p and it retrieves the neighborhood of p to verify if p is a core point or not. If p do not have at least $MinPts$ within a radius Eps, it is marked as a noise (it can be remarked as a border point later). Otherwise, p is a core point and a new cluster is formed from p and its neighborhoods. DBSCAN iteratively expend the new cluster by retrieving all points which are density-reachable from p. After that, the algorithm visit the next unvisited point. DBSCAN stops when all the points are visited.

Most of partitioning methods are heuristic, they are iterative relocation clustering algorithms. The quality of the solutions is measured by the clustering criterion. At each iteration, the iterative relocation algorithms reduce the value of the criterion function until convergence.
The convergence is local and the globally optimal solution cannot be guaranteed. The problem of local minimal could be avoided by using exhaustive search methods. However, finding the globally optimal partition is known to be a NP-hard problem.

2.3 Constraint-based Clustering

Constraint-based Clustering aims at finding clusters that satisfy user-specified constraints. It is highly desirable in many real applications. However, few algorithms adapt to user-specified constraints. There is no general solution to add constraints to traditional algorithms (k-Means, k-Medoids,...). User-constraints can be classified into cluster-level and instance-level constraints.

2.3.1 Cluster-level Constraints

Cluster-level constraints impose requirements on the shape, size or other features of the clusters. The most common type of cluster-level constraint is based on the size of clusters, it requires that each cluster has at least a minimum number of objects or has at most a maximum number of objects. Users may also limit the size of clusters by adding a maximum diameters constraint that requires the distance between any two points of

Algorithm 4 DBSCAN algorithm

```
function DBSCAN(𝒪,MinPts,Eps)
    C = 0
    for each unvisited point p in dataset 𝒪 do
        mark p as visited
        NeighborPts = regionQuery(p, Eps)
        if sizeof(NeighborPts) < MinPts then
            mark p as NOISE
        else
            C = C + 1
            expandCluster(p, NeighborPts, C, Eps, MinPts)
        end if
    end for
end function
function expandCluster(p, NeighborPts, cluster, Eps, MinPts)
    add p to cluster C
    for each point p' in NeighborPts do
        if p' is not yet member of any cluster then
            add p' to cluster C
        end if
        NeighborPts' = regionQuery(p', eps)
        if sizeof(NeighborPts') ≥ MinPts then
            add NeighborPts' to NeighborPts
        end if
    end for
end function
function regionQuery(p,Eps)
    return all points within the radius Eps from p
end function
```

the same class to be less than a threshold. Another type of cluster-level constraint is the minimum separation constraint that requires the distance between any two points of different classes to be superior to a threshold.

- Minimum capacity constraint: it requires that each cluster has at least a minimum number of objects
$$\forall c \in [1, k], |C_c| \geq \alpha$$
where α is a given parameter.
- Maximum capacity constraint: each cluster has at most a maximum number of objects
$$\forall c \in [1, k], |C_c| \leq \beta$$
where β is a given parameter.
- Maximum diameter constraint: it limits the diameter of clusters
$$\forall c \in [1, k], \forall o_i, o_j \in C_c, d(o_i, o_j) \leq \gamma$$
where γ is a given parameter.
- Minimum separation constraint: it requires the distance between any two points of different clusters to be superior to a given threshold θ
$$\forall c \in [1, k], \forall c' \neq c, \forall o_i \in C_c, o_j \in C_{c'}, d(o_i, o_j) \geq \theta$$

2.3.2 Instance-level Constraints

Instance-level constraints impose constraints on individual pairs of objects. Commonly, two kinds of constraints are used: must-link and cannot-link.

- A must-link constraint specifies that two objects o_i and o_j have to appear in the same cluster.
$$\forall c \in [1, k], o_i \in C_c \Leftrightarrow o_j \in C_c$$
- A cannot-link constraint specifies that two objects must not be put in the same cluster
$$\forall c \in [1, k],\ \neg(o_i \in C_c \wedge o_j \in C_c)$$

3 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for solving combinatorial search problems that relies on a wide range of techniques from Artificial Intelligence, Computer Science and Operational Research. The main principles in CP are: (1) users specify the problem declaratively in a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search. A Constraint Satisfaction Problem (CSP) is a triple $\langle X, D, C \rangle$ where:

- $X = \{x_1, x_2, ..., x_n\}$ is a set of variables,
- $D = \{D_1, D_2, ..., D_n\}$ is a set of domains, with $x_i \in D_i$,
- $C = \{C_1, C_2, ..., C_t\}$ is a set of constraints, each C_i is a condition on a subset of X.

A solution of the CSP is a complete assignment of a value $a_i \in D_i$ to each variable x_i, such that all constraints in C are satisfied. A Constraint Optimization Problem (COP) is a CSP associated to an objective function. In this case, the optimal solution is a solution of the CSP that optimizes the objective function.

In general, CSPs are NP-hard. However, techniques used in CP solvers allow to solve many practical applications efficiently. Most popular techniques are constraint propagation, branching and search.

3.1 Constraint Propagation

Constraint propagation consists of removing values from the domain of some variables when it can be determined that the values cannot be in any solution of a constraint. Propagation is done by propagators: each constraint is realized by a set of propagators, depending on which kind of consistency the constraint is considered. If propagators implement the arc-consistency for a constraint, they can remove any inconsistent values of the domain with respect to the constraint. If they implement the bound-consistency for a constraint, they only modify the upper or the lower bound of the variable domains.

The objective of the consistency techniques is to detect values of variables that can not be in the solution with the help of constraints. The consistency techniques try to detect inconsistent assignments as soon as possible to prune the search space.

Example 1 *Assume we have 2 variables x_1, x_2 taking integer values in [1, 4] with the constraint: $x_1 < x_2 - 1$. Consistency techniques can reduce the domain of x_1 and x_2 without loss of any solution, $D_1 = \{1, 2\}$ and $D_2 = \{3, 4\}$.*

We present in this section basic consistency algorithms for unary and binary constraints. Normally, a CSP can be represented by a constraint graph where variables are nodes and constraints corresponds to hyper-arcs. To express domain of variables, a value graph can be used. It is a bipartie graph whose vertices are divided into two disjoint sets: a set of vertices from variables and a set of vertices from values. Every edge in the value graph connect a vertice of variable to a vertice of value.

Example 2 *Assume we have 4 variables x_1, x_2, x_3 and x_4 and their domains are: $x_1 \in [1, 2], x_2 \in [2, 3], x_3 \in [1, 3], x_4 \in [1, 2]$ with the constraints: $x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4$. Constraint graph and value graph of this CSP are expressed in figure 1. The consistency techniques try to detect inconsistent assignments, or they try to remove as many edges as possible in the value graph. The value graph of this CSP after constraint prograpation is expressed in figure 2*

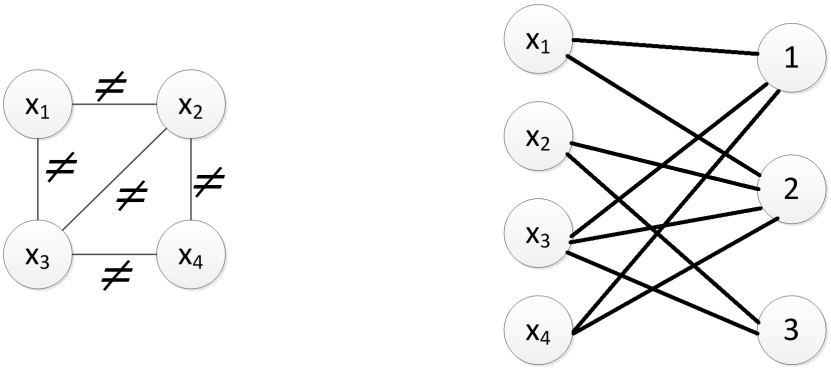

Figure 1: constraint graph and value graph of exemple 2

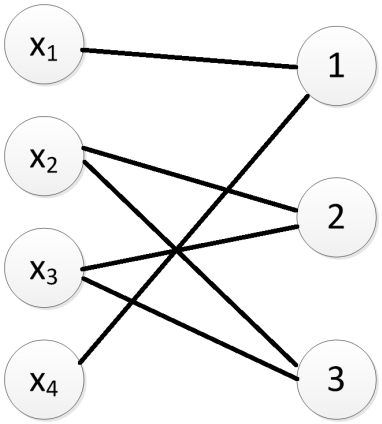

Figure 2: value graph after constraint propagation

Algorithm 5 Node consistency algorithm

```
for all variable x_i do
    for all value a in D_i do
        if unary constraint on X is inconsistent with a then
            delete a from D_i
        end if
    end for
end for
```

- Node Consistency is the simplest technique. A variable x_i is node consistency iff for every value of the current domain D_i, each unary constraint on x_i is satisfied. A CSP is node consistent iff all variables x_i are node consistent.

  Example 3 *Assume the variable x_1 taking integer values in [1, 10] with the unary constraint: $x_1 > 3$. By applying node consistency algorithm, we can delete values [1, 3] from D_1. D_1 is now [4, 10].*

- Arc Consistency is used to guarantee the consistency of each binary constraint. An arc (x_i, x_j) is arc consistent iff for every value a_i in the current domain of D_i there are some values a_j of D_j so that (a_i, a_j) satisfy the binary constraint between x_i and x_j. A CSP is arc consistent iff every arc (x_i, x_j) is arc consistent.

Algorithm 6 Basic Arc Consistency algorithm

```
function REVISE(x_i, x_j)
    DELETED ← false
    for all value a_i in D_i do
        if there is no a_j in D_j such that (a_i, a_j) satisfies all binary constraints on
(x_i, x_j) then
            delete a_i in D_i
            DELETED ← true
        end if
    end for
    return DELETED
end function
function ARC CONSISTENCY-1
    repeat
        CHANGED ← FALSE
        for all arc(x_i, x_j) do
            CHANGED ← Revise(x_i, x_j) or CHANGED
        end for
    until not Changed
end function
```

In this basic arc consistency algorithm, all the arcs will be revised again if any inconsistency is detected and removed. Many other arc consistency algorithms (AC-2, ..., AC-7) therefore proposed to reduce the number of revises.

Example 4 *Assume the variables x_1, x_2 and x_3 taking integer values in [1, 4] with constraints: $x_1 < x_2$ and $x_2 < x_3$. Suppose that arcs will be revised in the order (x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2).*

- *we first select the arc (x_1, x_2) and remove the value 4 from D_1 because if $x_1 = 4$, we cannot find a value for x_2 such that $x_1 < x_2$*

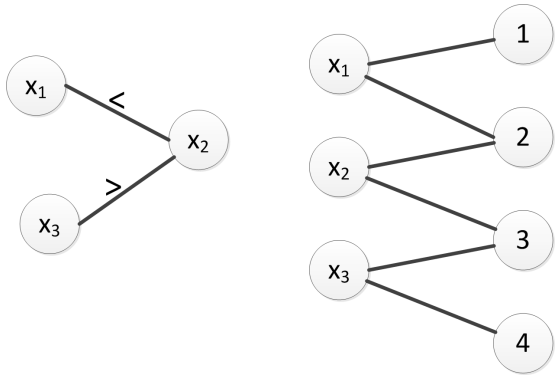

Figure 3: constraint graph and value graph of exemple 4

- *arc (x_2, x_1) is revised and the value 1 is removed from D_2 for the same reason.*
- *arc (x_2, x_3) is revised and the value 4 is removed from D_2. Now $D_1 = [1, 3], D_2 = [2, 3], D_3 = [1, 4]$*
- *arc (x_3, x_2) is revised and the value 1 and 2 is removed from D_3. Now $D_1 = [1, 3], D_2 = [2, 3], D_3 = [3, 4]$*
- *all the arcs are revised again: when revising (x_1, x_2), the value 3 is removed from D_1*
- *Finally $D_1 = [1, 2], D_2 = [2, 3], D_3 = [3, 4]$*

Constraint graph and value graph of this CSP after constraint propagation are expressed in figure 3.

The Arc Consistency techniques reduce significant by the search space in many CSPs but there still exists many other possible inconsistencies.

Example 5 *Assume the variables x_1, x_2 and x_3 taking integer values in [1, 2] with constraints: $x_1 \neq x_2$, $x_2 \neq x_3$ and $x_3 \neq x_1$. The Arc Consistency techniques can not detect any inconsistencies in this case, where there is no solution.*

- Path Consistency is stronger than Arc Consistency, it detects inconsistencies in every path in the constraint graph. In fact, the Path Consistency Techniques remove more inconsistencies but it is too costly and is not efficient in most of CSPs. Moreover, the Path Consistency is not able to detect all of inconsistencies.

3.2 Search and Strategies

Solvers search for solutions by recursively combining two steps: constraint propagation and branching. The solver propagates all the constraints until reaching a stable state, where either the domain of a variable is reduced to the empty set or no variable domain

can be reduced. In the first case, there is no solution and the solver backtracks. In the other case, if all the variable domains are singleton, a solution is reached, otherwise the solver chooses a variable whose domain is non-singleton and splits the domain into two or more parts, which creates two or more branches in the search tree. The solver explores then each branch, where constraint propagation becomes reactive again, because of the modifications of a variable domain.

The search tree exploration strategy can be defined by the user. In a depth-first search strategy, the solver orders the branches in the order specified by the user and explores in depth each branch to find solutions. In an optimization problem, a branch-and-bound strategy is obtained from a depth-first search: each time the solver reaches a solution of the CSP, the value of the objective function on the solution is calculated and a new constraint is added that forbids all solutions which are not better than this one.

Strategies of choosing variables and of choosing values are extremely important, since they can reduce drastically the search tree. For more details, we refer to the text book on Constraint Programming by [11].

3.2.1 Variable Ordering

The ordering in which variables are instantiated can affect the search space of backtrack search. The underlying idea is to detect the failure early to avoid useless branchings. Since all variables have to be instantiated, variable is chosen in order to maximally reduce the search space or add constraints. Therefore different heuristics may be used during the search to determine the next variable to be instantiated.

- The variable with the smallest domain is selected for instantiation
- The variable that participated in a higher number of constraints is selected
- The variable that has the largest number of constraints with instantiated variables.

The order of values of variables is important, a correct choice of values can help to reduce search tree. The example below demonstrates the importance of the variable ordering.

Example 6 *Find distinct digits for the letters* S, E, N, D, M, O, R, Y *such that*

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline = & M & O & N & E & Y \end{array}$$

In this problem, we can define a variable with the domain [0, 9] for each letter with following constraints:

- *AllDifferent(S, E, N, D, M, O, R, Y)*
- $S \neq 0, M \neq 0$

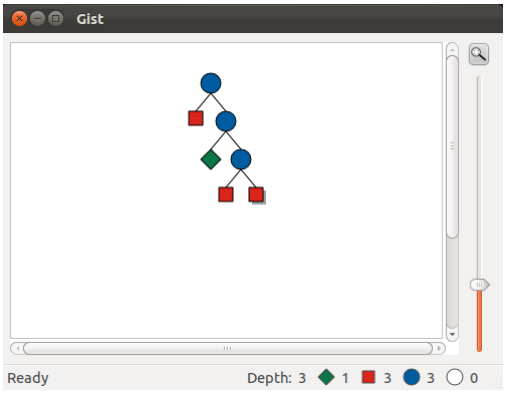

Figure 4: Search space with the strategy: The variable with the smallest domain is selected for instantiation

- $1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 1000O + 100N + 10E + Y$

By using consistency techniques, we can reduce the domain of variables to: $D_S = \{9\}, D_E = [4, 7], D_N = [5, 8], D_D = [2, 8], D_M = \{1\}, D_O = \{0\}, D_R = [2, 8], D_Y = [2, 8]$. *Figures 4 and 5 show different search trees with different strategies for the next variable to be instantiated. These search trees are generated by Gist environment of the Gecode solver where blue circle is a stable state but not yet a solution, red square is a fail state (there is no solution), green diamond is an intermediate solution and the orange diamond is the optimal solution.*

3.2.2 Branching strategy

In the backtracking algorithm, a node $p = \{x_1 = a_1, \ldots, x_j = a_j\}$ in the search tree is a set of assignments of instantiated variables. A branching extends p by selecting a variable x_i that is not instantiated and adding the branches for assignments of x_i. Three popular branching strategies are often used:

- Enumeration: A branch is generated for each value in the domain of the variable.
- Binary choice points: The variable x_i is instantiated by a 2-way branching. Assuming the value a_i is chosen for branch, two branches are generated with $x_i = a_i$ and $x_i \neq a_i$.
- Domain splitting: Here the domain of the variable is split to create the branching. For example, assume the value a_i is chosen for the split, two branches are generated with $x_i \leq a_i$ and $x_i > a_i$.

3.2.3 Branch-and-bound technique

To solve optimization CSPs, the common approach is to use branch-and-bound techniques. Initially, a backtracking search is used to find any solution p which satisfies all

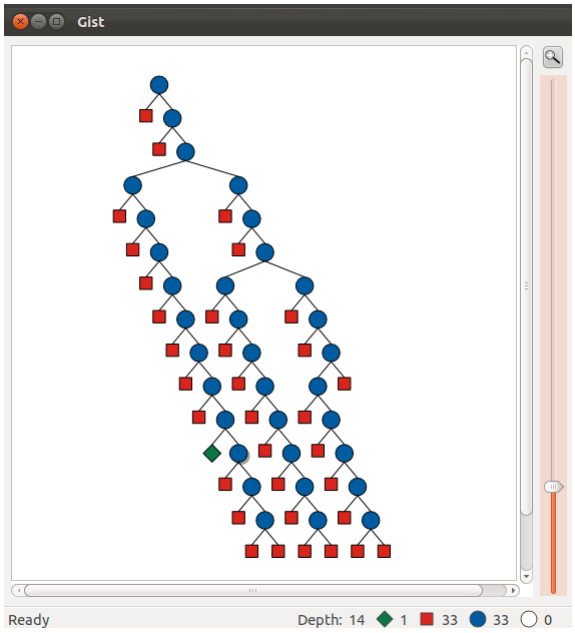

Figure 5: Search space with the strategy: The variable with the biggest domain is selected for instantiation

the constraints. An additional constraint is added to the CSP when a solution is found allowing to forbid solutions that are not better than this solution. The solvers continue to search for solutions of the new CSP until the CSP is unsatisfiable. The last solution found has been proven optimal. Moreover, heuristic can be used to prune more the search space. The heuristic here is a function h that calculate the bound from values of instantiated variables. The efficiency of this technique is affected how quickly we can improve the bound:

- Whether a good solution is found early
- Quality of the heuristic function

Let us consider the following example for an illustration of Constraint Optimization Problem and search strategies.

Example 7 *We have to find an assignment of digits to letters such that*

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & S & T \\ \hline = & M & O & N & E & Y \end{array}$$

and where the value of MONEY is maximized. We can model this problem as a Constraint Optimization Problem, where we use eight variables S, E, N, D, M, O, T, Y*, whose domain is the set of digits* $[0, 9]$*. Constraints which specify the problem are:*

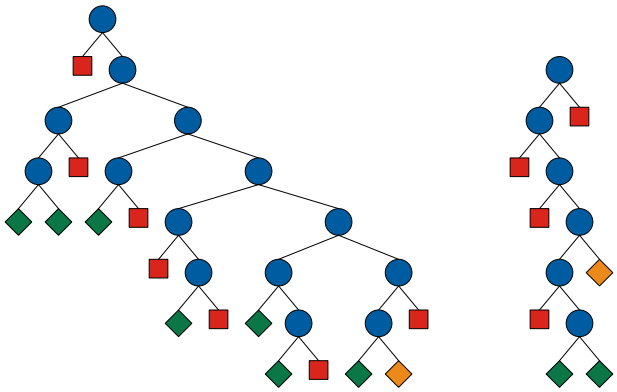

Figure 6: Search trees

- *the letters* S *and* M *must not be 0:* $S \neq 0,\ M \neq 0$
- *all the letters are pairwise different:* $\texttt{alldifferent}(S, E, N, D, M, O, T, Y)$

  (note that instead of using the constraint $\neq$ *on each pair of variables, we use this very one constraint, which is stated on all variables, this constraint is called a "global constraint" in CP, the same kind as the following linear constraint)*
- $(1000S + 100E + 10N + D) + (1000M + 100O + 10S + T) = 10000M + 1000O + 100N + 10E + Y$
- $maximize(1000M + 1000O + 100N + 10E + Y)$

The optimal solution of this problem is the assignment $S = 9, E = 7, N = 8, D = 2, M = 1, O = 0, T = 4, Y = 6$*, with* $MONEY = 10876$*. Initial propagation from these constraints leads to a stable state with the domains* $D_S = \{9\}$*,* $D_E = \{2, 3, 4, 5, 6, 7\}$*,* $D_M = \{1\}$*,* $D_O = \{0\}$*,* $D_N = \{3, 4, 5, 6, 7, 8\}$ *and* $D_D = D_T = D_Y = \{2, 3, 4, 5, 6, 7, 8\}$*. Strategies consist of the way to chose variables and for each chosen variable, the way to chose values. If variables are chosen in the order* S, E, N, D, M, O, T, Y *and for each variable, the remaining values in its domain are chosen in an increasing order, the search tree has 29 states with 7 intermediate solutions (solution which is better than the precedent). However, if variables are chosen in the order* S, T, Y, N, D, E, M, O*, the search tree has only 13 states with 2 intermediate solutions. These two search trees are showed in Figure 6. For each stable state, the left branch is the case where the chosen variable is assigned to the chosen value, the right branch is the other case where the chosen value is removed from the domain of the chosen variable. It is worth to notice that it is an exhaustive search, the returned solution is guaranteed to be optimal.*

4 Related works

Recent advances in CP make the framework powerful to solve data mining problems. Several works aims at modeling data mining problems, and particularly clustering problems in a CP framework.

L. De Raedt, T. Guns and S. Nijssen present in [4] a CP framework for modeling the itemset mining problem. They present in [7] a framework for k-patterns set mining and show how it can be applied to the conceptual clustering. The problem consists of finding k-patterns that do not overlap and cover all the transactions: each transaction can be viewed as an object and each pattern can be viewed as a cluster, two objects are similar if they share the same itemset. Conceptual clustering generates a concept description for each cluster and here, the concept is the itemset. For modeling the problem, boolean variables are used to model the presence of each item and each transaction in each pattern. Constraints and reified constraints express the relationship of covering, non-overlapping, etc. Additionnal constraints can be added to express the criterion function: maximize the minimum size of cluster or minimize the difference between cluster sizes.

P. Boizumault, B. Crémilleux *et al.* present in [1] and [10] a constraint-based language expressing queries to discover patterns in data mining. Queries are expressed by constraints over terms that are built from constraints, variables, operators and function symbols. The constraint conceptual clustering problem can be expressed in this language. The language is translated into a set of clauses and a SAT solver is used to solve the clustering task.

These works consider the conceptual clustering problem, where clusters are non-overlapping partitions on qualitative databases. Our approach considers the clustering problem in its original form and is capable to proceed on qualitative or quantitative databases.

A SAT-based framework for constraint-based clustering has been proposed by I. Davidson, S. Ravi and L. Shamis in [2]. The problem is however limited to only two clusters and is then transformed to a 2-SAT problem, which is polynomial. In this work, both instance-level constraints and cluster-level constraints are modeled. The model also allows users to add criterion function based on the diameter or the separation of clusters. The optimization algorithm is based on the binary search for the criterion value, at each step the upper-bound or the lower-bound of this value is modified, and a 2-SAT solver is used to determine if all the constraints are satisfied with these bounds. Our approach based on constraint programming includes all the search and the constraint satisfaction, and is more general since the number of clusters is not limited to 2.

5 Distance-based constraint clustering

We now introduce our model for *distance-based constraint clustering* that can be provided to CP systems. Assume that we are given a collection of n points (objects). Without loss of generality, we assume that the points are indexed and we name points by their index. The distance between any two points i, j is calculated with $d(i, j)$. We also assume that

the number of clusters, k, is known in advance. The model aims at finding a partitioning in k clusters, minimizing the maximum diameter of clusters.

5.1 Model

5.1.1 Modeling Variables

For each cluster, the point which has the smallest index is designated as the representative of the cluster. Identifying cluster becomes identifying its representative. For each cluster $c \in [1, k]$ we introduce then an integer variable $I[c]$, which is its representative. The domain of each $I[c]$ is the interval $[1, n]$.

The assignment of a point to a cluster is then an association of this point with the representative of the cluster. We introduce for each point $i \in [1, n]$ an integer variable $G[i] \in [1, n]$ which is its representative.

To represent the maximum diameter, we introduce a variable D. This variable must be an integer variable, since we use CP solvers on integer variables. The domain of D is the interval between the minimum and the maximum distance between any two objects. In the experiments where distance is not integer, it is multiplied by 100 and rounded to integer.

Our model searches for a global optimal solution, which means a complete assignment of all variables of I, G and D that satisfies the constraints below.

5.1.2 Modeling Distance-based Clustering Constraints

The relationship between points and their cluster is expressed by the following constraints:

- The representative of a representative point is itself:

$$\forall c \in [1, k],\ G[I[c]] = I[c]$$

- The representative of each point must be one and only one of all the representatives:

$$\forall i \in [1, n],\quad \#\{c \mid G[i] = I[c]\} = 1$$

- The representative must be the smallest index in the cluster:

$$\forall i \in [1, n],\quad G[i] \leq i$$

A same set of clusters could be represented by different solutions, differing by the order with whom the clusters are enumerated. The following constraints allow to avoid this symmetry:

- The representatives are in an ascending order:

$$\forall c < c' \in [1, k],\ I[c] < I[c']$$

- The representative of the first cluster must be the first point, since a representative must have the smallest index:

$$G[1] = 1,\ I[1] = 1$$

The diameter constraint:

- If the distance between any two points is superior to the diameter, they must be in different clusters. This is expressed by the following reified constraints: $\forall i < j \in [1, n]$,

$$d(i, j) > D \rightarrow (G[j] \neq G[i] \wedge\ G[j] \neq i) \tag{13}$$

- The maximum diameter is aimed to be minimized: minimize D

5.1.3 Modeling User Constraints

The advantage of modeling with CP is that many different types of user constraints can be directly added. Both cluster-level constraints and instance-level constraints can be formulated within our model. For cluster-level constraints:

- Minimum capacity α for each cluster is expressed by:

$$\forall c \in [1, k],\quad \#\{i \mid G[i] = I[c]\} \geq \alpha$$

- Maximum capacity β for each cluster is expressed in the same way:

$$\forall c \in [1, k],\quad \#\{i \mid G[i] = I[c]\} \leq \beta$$

- Minimum separation constraint means the separation between any two clusters must be at least θ. That is, if the distance of two points is inferior to θ, they must be in the same cluster: for all $i < j \in [1, n]$ such that $d(i, j) < \theta$, we add the constraint $G[i] = G[j]$.

- Maximum Diameter of any cluster must be at most a given value γ. That is, if the distance of two points is greater than γ, they must be in different clusters: for all $i < j \in [1, n]$ such that $d(i, j) > \gamma$, we add the constraints $G[j] \neq G[i]$ and $G[j] \neq i$.

For instance-level constraints:

- A must-link constraint on i, j means that they share the same representative: $G[i] = G[j]$.

- A cannot-link constraint on i, j with $i < j$ is expressed by:

$$G[i] \neq G[j],\ G[j] \neq i$$

5.2 Search Strategy

A search strategy must be given to indicate to the solver the choice of variables and values. For the choice of variables, we use two different orderings:

- Variables will be chosen in the order I, D then G. This order represents the fact that the cluster representatives must be identified first, then with each bound on the maximum diameter the solver will try to determined the assignment of points to clusters.
- The order is I, G then D. The representative of clusters must be always identified first. Next, objects are assigned to clusters and the branching of D is used to calculate the maximum diameter of clusters. The benefit of this approach is to quickly find a solution to have a good upper bound D.

For the choice of variables, they will be chosen in the order I, D then G. This order represents the fact that the cluster representatives must be identified first, then with each bound on the maximum diameter the solver will try to determined the assignment of points to clusters.

Variables in I are chosen from $I[1]$ to $I[k]$, that means identifying the representative of the first till the last cluster. For the choice of values for each $I[c]$, since the representative is the smallest index in each cluster, the values are enumerated in the ascending order.

For the variable D, we use the idea of binary search. At each choice point for D, the remaining domain $[b_{min}, b_{max}]$ is split to two parts $[b_{min}, \lceil (b_{min} + b_{max})/2 \rceil]$ and $[\lceil (b_{min} + b_{max})/2 \rceil + 1, b_{max}]$. Since D needs to be minimized, the lower part is considered first.

Variables in G are chosen in the ascending order of the size of their remaining domain. For the choice of values for each $G[i]$, the closest representative is enumerated first.

5.3 Model Improvements

This model is complete for it allows to find the optimal solution. In order to improve the efficiency, different aspects have been studied.

5.3.1 Search strategy improvement by point ordering

In order to identify cluster representatives, the variables in I are taken in the order $I[1], \ldots, I[k]$ when enumerating with values. Since a representative must have the smallest index in the cluster, for each variable, values (point indexes) are enumerated in the ascending order. The index of points is then really important. Points should be reordered such that those which are more probably to be a representative should have smaller index. We present below two versions which use heuristics to reorder points : weight heuristic (version 1.1) and FPF heuristic (version 1.2).

Version 1.1 uses an heuristic where points are ordered such that: the first point is far from all the other points, the second point is far from the first point but also far enough from the rest, the third point is far from both the first and the second point, also far

enough from the rest, and so on. To realize this heuristic, assume that the first p points have been already ordered, the next $(p+1)^{th}$ point is the $\arg\max_{i\in[p+1,n]} f(i)$ where

$$f(i) = \frac{\sum_{j\in[1,p]} d(i,j)}{p} \frac{\min_{j\in[1,p]} d(i,j)}{\max_{j\in[1,p]} d(i,j)} + \frac{\sum_{j\in[p+1,n]} d(i,j)}{n-p}$$

Version 1.2 uses the FPF heuristic to reorder points. The FPF algorithm is used with $k = n$ (each point is in a cluster), so that each point will be taken in turn. The choice order of the FPF algorithm gives the order of points.

5.3.2 Constraint improvements

For a given k, without any user-constraints to satisfy, the diameter d_{FPF} returned by the FPF algorithm has been proved by [6] to satisfy $d_{opt} \leq d_{FPF} \leq 2d_{opt}$, where d_{opt} is the optimal diameter, which is aimed to be found. This entails bounds for the variable D, whose domain will be $[d_{FPF}/2, d_{FPF}]$. Moreover, if the distance between any two points i, j is greater than d_{FPF}, we can directly add constraints to indicate that they are in different clusters, instead of the reified constraint (13). That is, for all points $i < j \in [1, n]$:

- if $d_{FPF}/2 \leq d(i,j) \leq d_{FPF}$ then the reified constraint (13) is added,
- otherwise, if $d(i,j) > d_{FPF}$, then the constraints $G[j] \neq G[i]$ and $G[j] \neq i$ are added.

Lots of reified constraints can be removed from the model, which improves the model efficiency. Versions 2.1 and 2.2 extend versions 1.1 and 1.2, respectively, with these improvements.

6 Experimentations

Due to the complexity of the clustering problem, classical algorithms always find local optimum. Moreover, when extended with user constraints, the performance of these methods often decreases. Our model is guaranteed to find a global optimum and both cluster-level constraints and instance-level constraints can be integrated directly. In this section, we present: experiments on different databases, a comparison between the global optimum and the approximate solution found by the FPF algorithm, and the performance of the model for constraint-based clustering.

We have implemented our model in Gecode solver version 3.7.3[1]. Experiments are realized on a 2.4GHz Core i5 Intel running Ubuntu 12.04. For the experiments we use

[1] http://www.gecode.org

the 5 datasets, with the following characteristics:

Dataset	#obj.	#attr.	#classes
iris	150	4	3
ionosphere	351	34	2
kdd_synthetic_control	600	60	6
vehicle	846	18	4
yeast	1484	8	unknown

6.1 Distance-based clustering

For the distance-based clustering problem, the four versions 1.1, 1.2, 2.1 and 2.2 are experimented. Here we test the performance of 4 versions: 1.1, 1.2, 2.1, 2.2 with 2 branching techniques: branching in I, D, G and branching in I, G, D. The time limit for each test is 10 minutes. Results are shown in detail in the appendix.

The FPF heuristic used in versions 1.2 and 2.2 gives in general better performance except for `ionosphere` and `yeast` datasets. Constraint improvements make the versions 2.1 and 2.2 much more efficient. We can see that the versions 1.x cannot proceed for $k > 6$ on `kdd_synthetic_control` and `vehicle` datasets, or for $k > 1$ on `yeast`, while the versions 2.x can proceed much further.

From the result of experimentations, we can see that there is not much different in performance in the ordering I, G, D and I, D, G. On `iris` and `kdd_synthetic_control` datasets in figures 7 and 9, the performance of the two orderings are almost the same. There is more different in `ionosphere` and `vehicule` datasets but the trend is not very clear to determine what ordering is better. The reason here is : the solution is always the same in both cases. In contrast, when we change the ordering technique, the different between version 1.1 and 1.2 (or 2.1 and 2.2) is quite significant, since the solution change completely.

6.2 Solution quality

For the distance-based clustering, the FPF algorithm is very fast since it uses only heuristic, but it returns a 2-approximation of the optimum. We recall that it is proved in [6] that a $(2 - \epsilon)$-approximation for any $\epsilon > 0$ is NP-hard for a three dimensional space. Our model is guaranteed to find the global optimum so takes evidently more time. We compare the quality of solutions in tables below. We can see that when the number of clusters is small enough, the solution of FPF algorithm is quite comparable. However, while increasing the number of clusters, the difference between FPF solution and our method solution becomes significant.

The figures 15 to 19 express the results of the method FPF and our version 2.2 with the branching strategy of $I, G.D$. Results in detail are expressed in the Appendix.

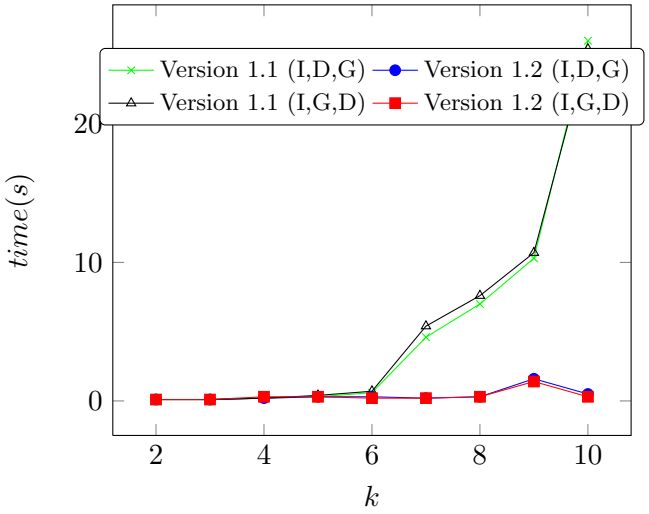

Figure 7: Dataset iris with branching I, D, G and I, G, D

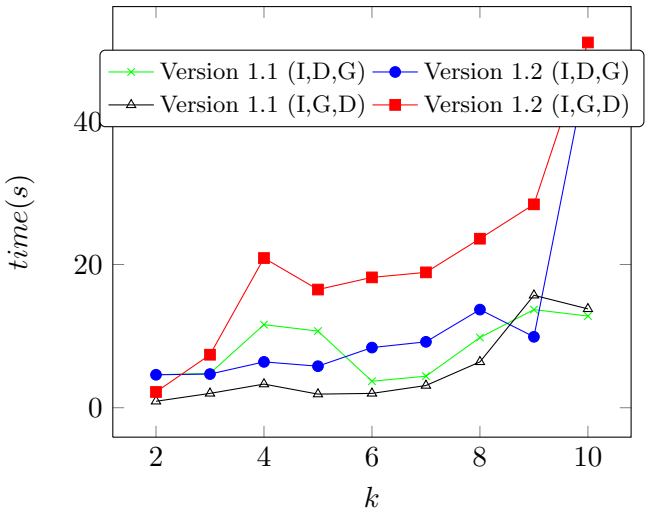

Figure 8: Dataset ionosphere with branching I, D, G and I, G, D

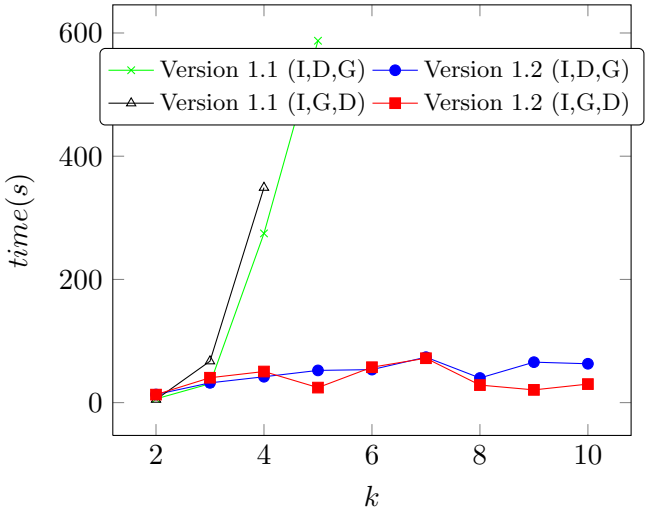

Figure 9: Dataset kdd_synthetic_control with branching I, D, G and I, G, D

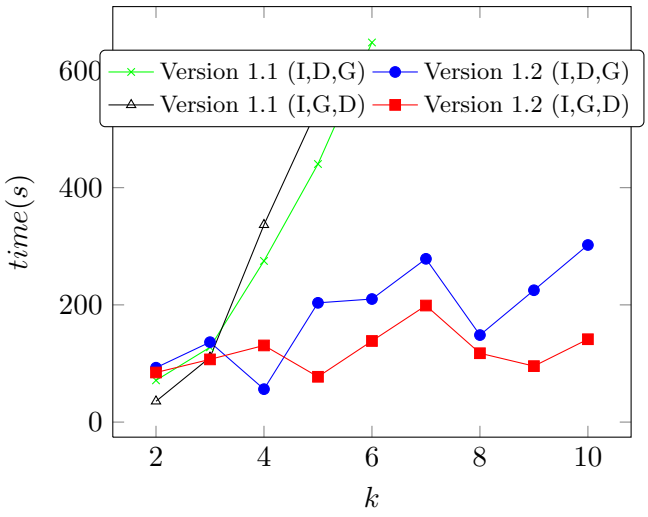

Figure 10: Dataset vehicule with branching I, D, G and I, G, D

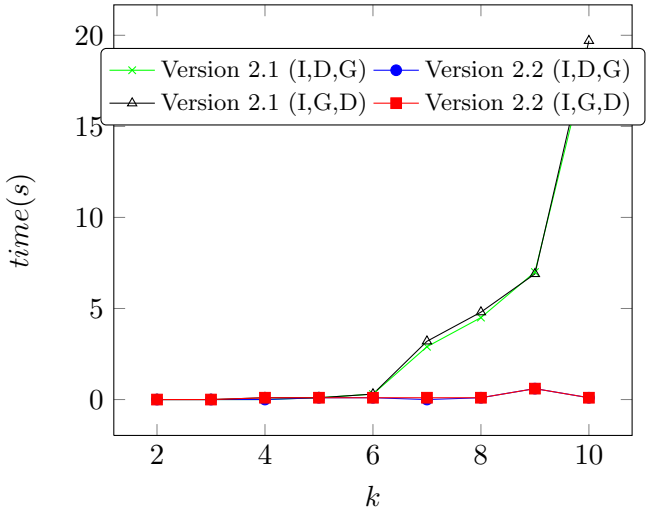

Figure 11: Dataset iris with branching I, D, G and I, G, D

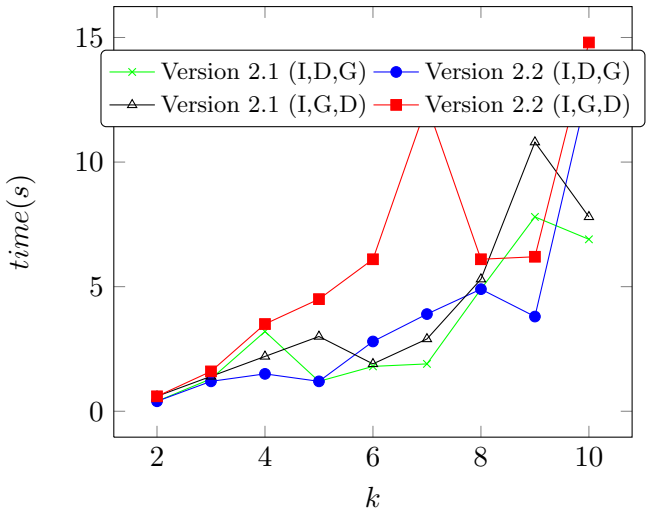

Figure 12: Dataset ionosphere with branching I, D, G and I, G, D

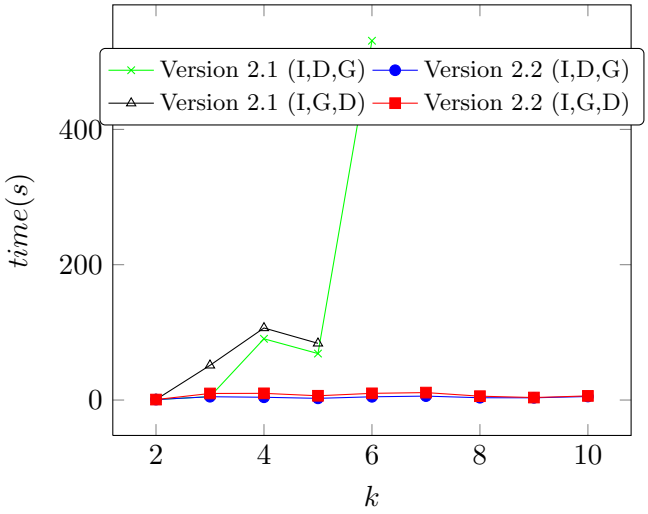

Figure 13: Dataset kdd_synthetic_control with branching I, D, G and I, G, D

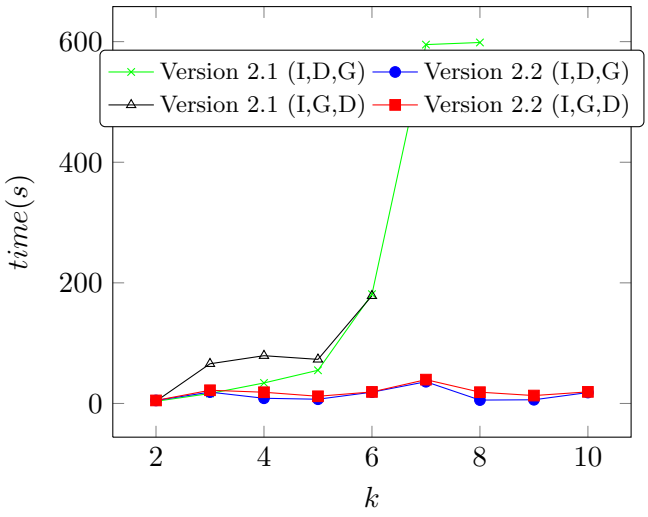

Figure 14: Dataset vehicule with branching I, D, G and I, G, D

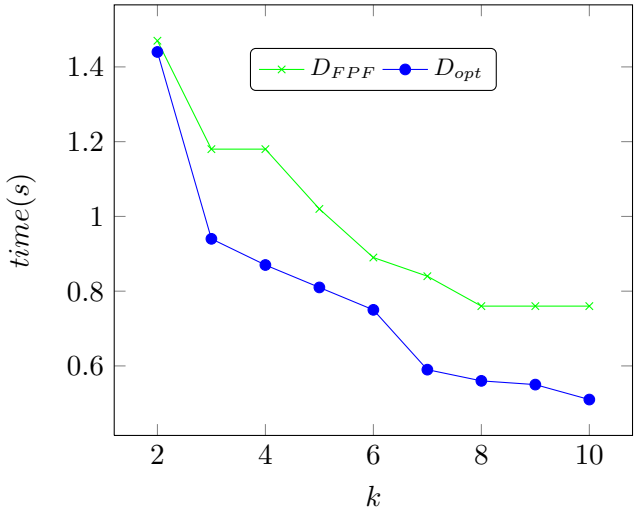

Figure 15: Dataset iris

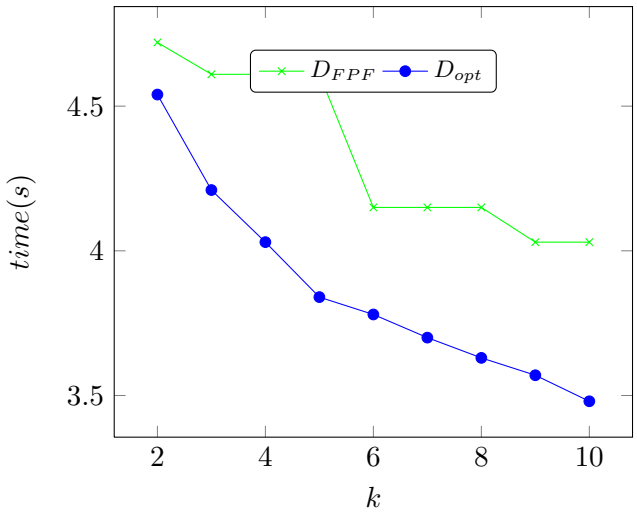

Figure 16: Dataset ionosphere

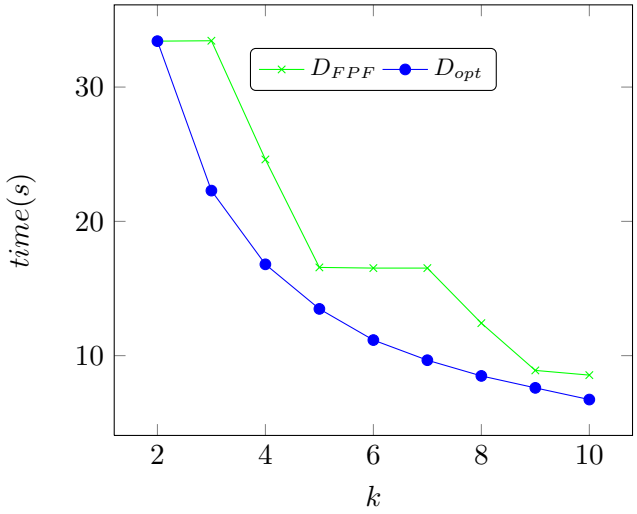

Figure 17: Dataset kdd_synthetic_control

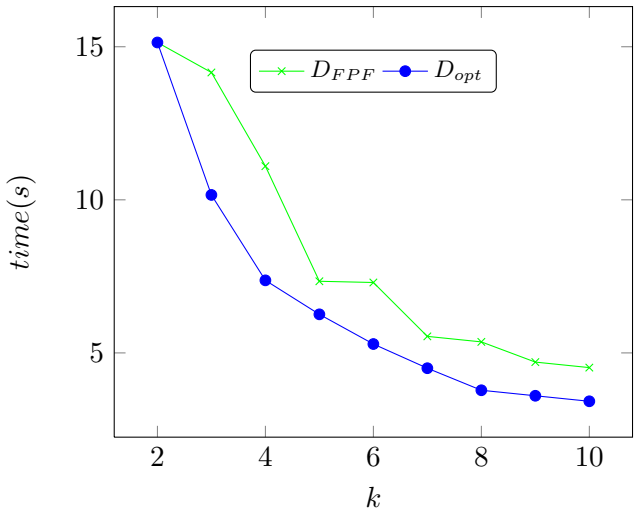

Figure 18: Dataset vehicle

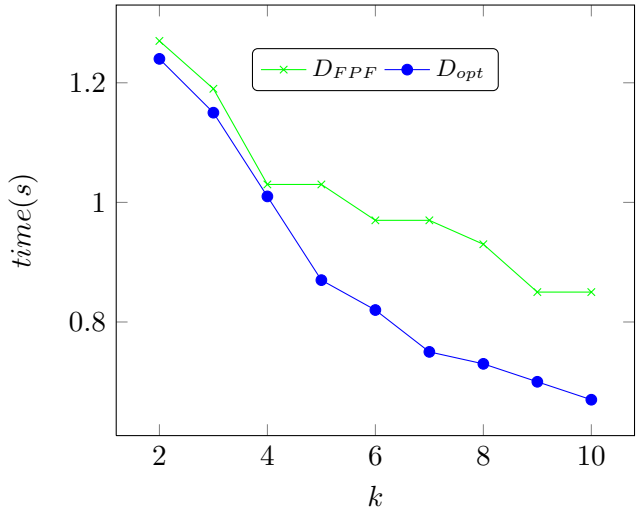

Figure 19: Dataset yeast

6.3 Constrained clustering

The other advantage of our model is that one can post other constraints within the same framework. Experiments with instance-level constraints and cluster-level constraints are realized.

6.3.1 Instance-level constraints

In some datasets, we know in advance the class of objects, therefore instance-level constraints can be generated in a simple way: we choose several random pairs of objects and post the constraints Must-Link (ML) or Cannot-Link (CL) depending on their classes. Experiments are made with different combinations of the numbers of ML et CL constraints. For each combination, the test is realized 10 times. Versions 1.2 is used for experiments with and without user constraints. We record the gap between the average time in case user-constraint and the taken time in case without constraint.

Experiments show that for `iris` and `ionosphere` databases, ML constraints in general help to reduce search space and search time, which is not the case with CL constraints. However, for `kdd_synthetic_control` dataset, user added constraints slow down the performance, since the gap between the optimal diameter without user constraints and the one with user constraints is usually significant. The optimal diameter with user-constraints usually increases twice with respect to the diameter in case without constraint. Results are shown in detail in the appendix.

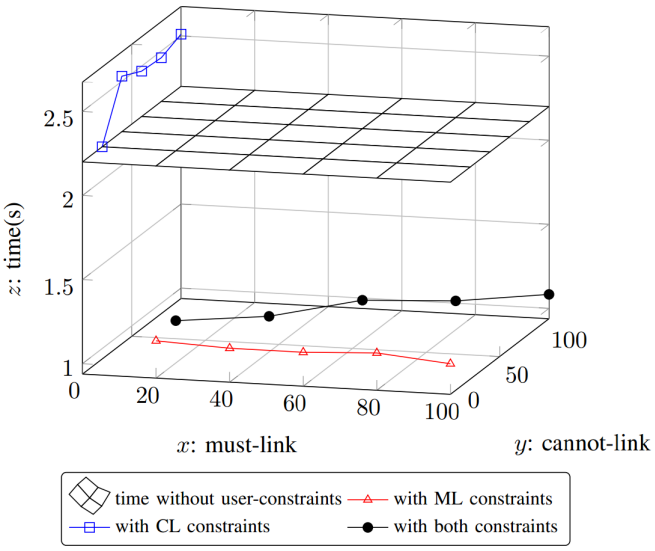

Figure 20: dataset ionosphere with instance-level constraints

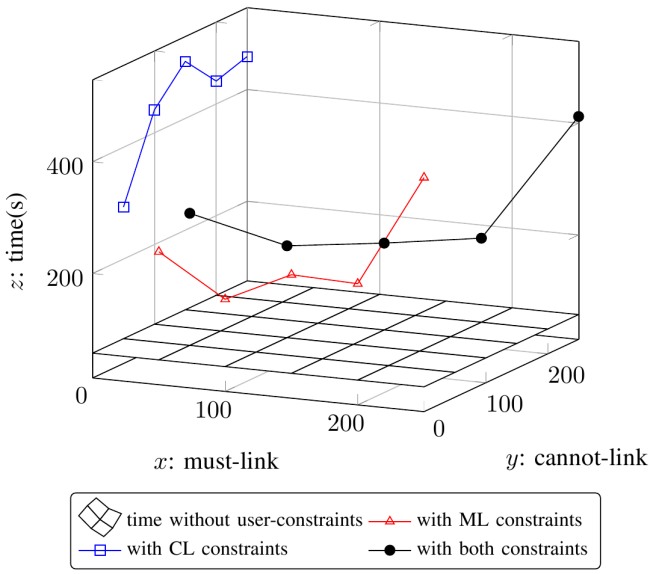

Figure 21: dataset kdd with instance-level constraints

6.3.2 Cluster-level constraints

6.3.3 Capacity constraints

The most common type of cluster-level constraints is the constraint put on the size of clusters. Here we test the minimum capacity and the maximum capacity constraint.

- dataset iris with k = 3

Min capacity	Time Total	Number of nodes	Diameter
0	0.1	305	0.94
5	0.1	305	0.94
10	0.1	305	0.94
15	0.1	305	0.94
20	0.1	305	0.94
25	0.1	305	0.94

Max capacity	Time Total	Number of nodes	Diameter
100	0.1	305	0.94
110	0.1	305	0.94
120	0.1	305	0.94
130	0.1	305	0.94
140	0.1	305	0.94
159	0.1	305	0.94

  In this dataset, clusters are separated and the size of clusters in the optimal solution is {48, 50, 52}. Therefore the capacity constraints do not affect the solution and the search at all.

- dataset kdd_synthetic_control, n=600, k=6

Min capacity	Time Total	Number of nodes	Diameter
0	57.48	16946	11.16
10	58.7	16946	11.16
20	88.1	16362	11.16
30	>30min		
40	>30min		
50	>30min		

Max capacity	Time Total	Number of nodes	Diameter
200	105.2	17190	11.16
300	92.2	16946	11.16
400	91.4	16946	11.16
500	90.7	16946	11.16

  In this dataset, the size of clusters in optimal solutions without capacity constraint are {32, 56, 136, 145, 109, 122 }. When capacity constraint is added, the search space change completely and the search space is significantly larger.

- dataset vehicle n=848, k=4

Min capacity	Time Total	Number of nodes	Diameter
0	212.9	7175	7.37
20	212.6	7175	7.37
40	211.4	7175	7.37
60	210.8	7175	7.37
80	211.1	7175	7.37
100	219.1	7175	7.37

Max capacity	Time Total	Number of nodes	Diameter
300	463	8063	8.05
400	265	7989	7.37
500	223	7175	7.37
600	230	7175	7.37
700	234	7175	7.37
800	227	7175	7.37

In this dataset, the size of clusters in optimal solutions without capacity constraints is {125, 166, 354, 201}.When we limit the maximum capacity to 300, the performance slow down because the optimal maximum diameter is now larger, the constraint propagation is less efficient and it takes more time to browse the search space.

Experiments show that capacity constraints do not help to reduce search space and search time. If the optimal solution changes if capacity constraint added, the search space is normally larger and slow down the performance.

6.3.4 Separation constraint

Minimum separation constraint require the distance between two objects of different clusters to be superior to a given threshold.

- dataset iris, n = 150, k = 3

Separation	Time total	Number of nodes	Diameter
0% max distance	0.1	305	0.94
1% max distance	0.1	291	0.94
2% max distance	0.1	271	0.94
3% max distance	0.1	201	0.94
4% max distance	0.1	163	0.94
5% max distance	0.1	91	0.94
6% max distance	0.1	49	0.94
7% max distance	0.1	44	0.94
8% max distance	0.1	17	1.8
9% max distance	0	13	1.8

Separation	Time total	Number of nodes	Diameter
0% max distance	56.6	16946	11.16
1% max distance	14.1	82	11.16
2% max distance	14.3	11	39.09
3% max distance	8.3	0	No solution
4% max distance	9.5	0	No solution
5% max distance	10.7	0	No solution
6% max distance	4.2	0	No solution
7% max distance	2.2	0	No solution
8% max distance	1.7	0	No solution
9% max distance	1.7	0	No solution

- dataset vehicle n=848, k=4

Separation	Time total	Number of nodes	Diameter
0% max distance	158.5	7175	7.37
1% max distance	193.8	3931	7.37
2% max distance	144.5	195	7.37
3% max distance	113.5	149	No solution
4% max distance	35	0	No solution
5% max distance	41.5	0	No solution
6% max distance	39.9	0	No solution
7% max distance	11	0	No solution
8% max distance	12.6	0	No solution
9% max distance	5.7	0	No solution

Experiments show that separation constraints do help to reduce search space and search time, since it adds constraints to group close objects.

7 Conclusion

We presented in this paper a constraint programming framework for constraint-based clustering. The model aims at finding a clustering which minimizes the maximum diameter of clusters. It is based on the distance between objects, thus allows to consider qualitative and quantitative data. This model allows to find the global optimum and both cluster-level and instance-level constraints can be easily added. Beside of the choice of the model, we presented some search strategies which improve the efficiency. Experiments on different databases show the interest of the approach.

For further works, we plan on addressing severals issues: (1) we continue to study techniques to improve the efficiency of the framework, for example: efficient constraint propagation for exploiting user-specified constraints; (2) the actual model allows to optimize the maximum diameter criterion, we want to extend it to take into account other criteria of the distance-based clustering problem, such that intra-class variance crite-

rion or absolute-error criterion; (3) we want to extend the framework to address other clustering tasks, for example overlapping clustering or fuzzy clustering.

References

[1] Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, Samir Loudni, and Jean-Philippe Métivier. Discovering Knowledge using a Constraint-based Language. *CoRR*, abs/1107.3407, 2011.

[2] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *SDM*, pages 94–105, 2010.

[3] L. De Raedt, T. Guns, and S. Nijssen. Constraint Programming for Data Mining and Machine Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.

[4] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In Ying Li, Liu Bing, and Sarawagi Sunita, editors, *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008,*, pages 204–212. ACM, 2008.

[5] Martin Ester, Hans P. Kriegel, Jorg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In Evangelos Simoudis, Jiawei Han, and Usama Fayyad, editors, *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

[6] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[7] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 2011. Accepted.

[8] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2 edition, January 2006.

[9] Oded Maimon and Lior Rokach, editors. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag, New York, 2005.

[10] Jean-Philippe Métivier, P.Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Clustering sous constraintes utilisant SAT. *JFPC*, 2012.

[11] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier B.V., Amsterdam, Netherlands, August 2006.

[12] Michael Steinbach, Pang-Ning Tan, and Vipin Kumar. *Introduction to data mining.* Addison-Wesley, us ed edition, May 2005.

Appendices

For the distance-based clustering problem, the four versions 1.1, 1.2, 2.1 and 2.2 are experimented. Here we test the performance of 4 versions: 1.1, 1.2, 2.1, 2.2 with 2 branching techniques: branching in I, D, G and branching in I, G, D. The time limit for each test is 10 minutes. The results are measured by second. We define:

- $Nodes$: The total number of nodes in the search tree.
- $T1$: Time for reading data sets and for the arrangement of objects.
- $T2$: Time for posting constraints.
- $T3$: Time for all of branching
- $T4$: Total time
- x: The test exceeds the time limit of 10 minutes

A Branch in order: I, D, G

- Iris

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	20	9	73	383	663	3961	5954	11994	51069
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.1	0.1	0.2	0.3	0.6	4.6	7.0	10.3	26.0
	T4	0.1	0.1	0.2	0.3	0.6	4.6	7.0	10.3	26.0
Version 1.2	Nodes	20	9	73	156	129	71	151	1222	292
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.1	0.1	0.2	0.3	0.3	0.2	0.3	1.6	0.5
	T4	0.1	0.1	0.2	0.3	0.3	0.2	0.3	1.6	0.5
Version 2.1	Nodes	17	5	64	281	557	3670	5664	11798	50867
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.0	0.0	0.0	0.1	0.3	2.9	4.5	7.0	19.1
	T4	0.0	0.0	0.0	0.1	0.3	2.9	4.5	7.0	19.1
Version 2.2	Nodes	17	6	65	154	121	64	144	1213	287
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.0	0.0	0.0	0.1	0.1	0.0	0.1	0.6	0.1
	T4	0.0	0.0	0.0	0.1	0.1	0.0	0.1	0.6	0.1

- ionosphere

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	341	340	770	649	669	348	1027	1416	1127
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	4.4	4.6	11.3	10.4	3.3	4	9.3	13.2	12.3
	T4	4.6	4.8	11.6	10.7	3.7	4.4	9.8	13.7	12.8
Version 1.2	Nodes	341	342	430	325	673	691	809	833	838
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	4.4	4.5	6.1	5.5	8	8.8	13.2	9.4	47.4
	T4	4.6	4.7	6.4	5.8	8.4	9.2	13.7	9.9	47.9
Version 2.1	Nodes	335	338	766	649	665	340	1022	1415	1119
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	0.2	1.1	2.9	0.9	1.4	1.5	4.4	7.3	6.4
	T4	0.4	1.3	3.2	1.2	1.8	1.9	4.9	7.8	6.9
Version 2.2	Nodes	335	338	424	329	671	684	808	828	832
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	0.2	1	1.2	0.9	2.4	3.5	4.4	3.3	12.6
	T4	0.4	1.2	1.5	1.2	2.8	3.9	4.9	3.8	13.1

- kdd_synthetic_control

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	8	16	3820	10700	25317	316532	x	x	x
	T1	0.4	0.5	0.7	0.8	1.0	1.0	x	x	x
	T2	0.3	0.3	0.3	0.3	0.3	0.3	x	x	x
	T3	5.4	29.9	273.7	586.3	1930.2	19742.7	x	x	x
	T4	6.1	30.7	274.7	587.4	1931.5	19744	x	x	x
Version 1.2	Nodes	8	16	15	25	111	239	25	45	66
	T1	0.4	0.5	0.7	0.8	1.0	1.0	1.2	1.4	1.5
	T2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	T3	12.5	31.3	41	51.2	52.3	72.5	38.5	64	61.3
	T4	13.2	32.1	42	52.3	53.6	73.8	40	65.7	63.1
Version 2.1	Nodes	1	14	3813	3623	17092	143037	x	x	x
	T1	0.4	0.5	0.7	0.8	1.0	1.0	x	x	x
	T2	0.2	0.2	0.2	0.2	0.2	0.2	x	x	x
	T3	0.1	4.3	89.6	67.7	529.8	3361.6	x	x	x
	T4	0.7	5	90.5	68.7	531	3362.8	x	x	x
Version 2.2	Nodes	1	14	13	17	96	226	22	39	53
	T1	0.4	0.5	0.7	0.8	1.0	1.0	1.2	1.4	1.5
	T2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	T3	0.1	4.2	3.2	1.5	3.6	4.6	2	1.7	3.5
	T4	0.7	4.9	4.1	2.5	4.8	5.8	3.4	3.3	5.2

- vehicule

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	54	33	443	1787	4632	x	x	x	x
	T1	0.6	0.9	1.2	1.2	1.4	x	x	x	x
	T2	0.5	0.6	0.6	0.6	0.6	x	x	x	x
	T3	70.1	126.1	273.2	438.6	645.8	x	x	x	x
	T4	71.2	127.6	275	440.4	647.8	x	x	x	x
Version 1.2	Nodes	54	64	24	106	140	270	38	117	506
	T1	0.5	0.7	1.0	1.2	1.5	1.6	2.0	2.0	2.2
	T2	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	T3	91.7	135.1	54.6	201.7	207.9	276.4	145.9	222.4	299.4
	T4	92.8	136.4	56.2	203.5	210	278.6	148.5	225	302.2
Version 2.1	Nodes	46	28	439	1012	3473	4698	4664	x	x
	T1	0.6	0.9	1.2	1.2	1.4	1.6	0.8	x	x
	T2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	x	x
	T3	2.6	15.2	32.4	53.4	179.4	593.1	597.4	x	x
	T4	3.6	16.5	34	55	181.2	595.1	598.6	x	x
Version 2.2	Nodes	46	59	17	98	132	254	28	108	496
	T1	0.5	0.7	1.0	1.2	1.5	1.6	2.0	2.0	2.2
	T2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	T3	4.1	17.7	7.2	5.4	16.8	34	3.2	3.7	15.5
	T4	5	18.8	8.6	7	18.7	36	5.6	6.1	18.1

- yeast

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	1472	x	x	x	x	x	x	x	x
	T1	1.6	x	x	x	x	x	x	x	x
	T2	1.3	x	x	x	x	x	x	x	x
	T3	1422.1	x	x	x	x	x	x	x	x
	T4	1425	x	x	x	x	x	x	x	x
Version 1.2	Nodes	1472	x	x	x	x	x	x	x	x
	T1	1.3	x	x	x	x	x	x	x	x
	T2	1.3	x	x	x	x	x	x	x	x
	T3	1419.4	x	x	x	x	x	x	x	x
	T4	1422	x	x	x	x	x	x	x	x
Version 2.1	Nodes	1466	1474	1441	1403	2789	5127	2604	4016	6786
	T1	1.3	2.1	2.5	3	3.6	4	5	5.2	5.8
	T2	0.6	0.9	1	1	1	1	1	1	1
	T3	1.6	18	14.5	48.4	66.8	105.3	115	107.3	136.3
	T4	3.5	21	18	52.4	71.4	110.3	121	113.5	143.1
Version 2.2	Nodes	1466	1472	1441	1397	1395	1216	2951	2184	x
	T1	1.3	2.1	2.4	3.2	4	4	4.6	5.1	x
	T2	0.8	1	1	1	1.1	1	0.9	1	x
	T3	3.4	6.6	14.5	27.1	49.7	60.5	338.5	490.8	x
	T4	5.5	9.7	17.9	31.3	54.8	65.5	344	496.9	x

B Branch in order: I, G, D

- iris

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	174	314	401	688	1011	4448	6586	12830	52164
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.1	0.1	0.2	0.4	0.7	5.4	7.6	10.7	25.4
	T4	0.1	0.1	0.2	0.4	0.7	5.4	7.6	10.7	25.4
Version 1.2	Nodes	165	305	903	625	841	1002	951	2302	1484
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0.1	0.1	0.3	0.3	0.2	0.2	0.3	1.4	0.3
	T4	0.1	0.1	0.3	0.3	0.2	0.2	0.3	1.4	0.3
Version 2.1	Nodes	35	178	315	278	683	3863	6009	12214	51543
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0	0	0.1	0.1	0.3	3.2	4.8	6.9	19.7
	T4	0	0	0.1	0.1	0.3	3.2	4.8	6.9	19.7
Version 2.2	Nodes	25	188	867	493	740	879	682	2204	1687
	T1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	T3	0	0	0.1	0.1	0.1	0.1	0.1	0.6	0.1
	T4	0	0	0.1	0.1	0.1	0.1	0.1	0.6	0.1

- ionosphere

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	1340	3310	3266	4702	5273	5013	6723	7781	8147
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	0.7	1.8	3	1.6	1.6	2.7	5.9	15.2	13.3
	T4	0.9	2	3.3	1.9	2	3.1	6.4	15.7	13.8
Version 1.2	Nodes	1280	3186	6093	6613	4077	5342	7362	6659	7918
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	2	7.2	20.6	16.2	17.8	18.5	23.1	27.9	50.5
	T4	2.2	7.4	20.9	16.5	18.2	18.9	23.6	28.4	51
Version 2.1	Nodes	1331	2981	3261	5703	4599	5940	7227	7129	7071
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	0.4	1.2	1.9	2.7	1.5	2.5	4.8	10.3	7.3
	T4	0.6	1.4	2.2	3	1.9	2.9	5.3	10.8	7.8
Version 2.2	Nodes	1270	2986	5905	6325	4388	5345	6845	5339	7815
	T1	0.1	0.1	0.2	0.2	0.3	0.3	0.4	0.4	0.4
	T2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	T3	0.4	1.4	3.2	4.2	5.7	11.9	5.6	5.7	14.3
	T4	0.6	1.6	3.5	4.5	6.1	12.3	6.1	6.2	14.8

- kdd_synthetic_control

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	612	19197	27697	40925	x	x	x	x	x
	T1	0.4	0.5	0.7	0.8	x	x	x	x	x
	T2	0.3	0.3	0.3	0.3	x	x	x	x	x
	T3	4.3	66.5	348	705.7	x	x	x	x	x
	T4	5	67.3	349	706.8	x	x	x	x	x
Version 1.2	Nodes	612	10606	13050	4838	16946	14772	7891	3071	4243
	T1	0.4	0.5	0.7	0.8	1.0	1.0	1.2	1.4	1.5
	T2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	T3	12.3	39.6	49.5	23.3	56.2	70.8	27	19	28.3
	T4	13	40.4	50.5	24.4	57.5	72.1	28.5	20.7	30.1
Version 2.1	Nodes	1	19139	7560	3954	x	x	x	x	x
	T1	0.4	0.5	0.7	0.8	x	x	x	x	x
	T2	0.2	0.2	0.2	0.2	x	x	x	x	x
	T3	0.1	50.6	105.6	82.8	x	x	x	x	x
	T4	0.7	51.3	106.5	83.8	x	x	x	x	x
Version 2.2	Nodes	1	10437	12842	4380	16686	14713	8123	2563	3851
	T1	0.4	0.5	0.7	0.8	1.0	1.0	1.2	1.4	1.5
	T2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	T3	0.1	9	9	5.3	8.7	9.8	4.3	2.1	4.4
	T4	0.7	9.7	9.9	6.3	9.9	11	5.7	3.7	6.1

- vehicle

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	854	13599	16813	16654	27568	x	x	x	x
	T1	0.6	0.9	1.2	1.2	1.4	x	x	x	x
	T2	0.5	0.6	0.6	0.6	0.6	x	x	x	x
	T3	34.4	108.7	334.8	535.2	810.2	x	x	x	x
	T4	35.5	110.2	336.6	537	812.2	x	x	x	x
Version 1.2	Nodes	854	8838	7175	2725	7749	4564	8156	8836	7216
	T1	0.5	0.7	1.0	1.2	1.5	1.6	2.0	2.0	2.2
	T2	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	T3	83.5	105.7	129.2	75.5	136.3	196.7	114.9	92.9	138.6
	T4	84.6	107	130.8	77.3	138.4	198.9	117.5	95.5	141.4
Version 2.1	Nodes	46	13516	4969	2421	6553	x	x	x	x
	T1	0.6	0.9	1.2	1.2	1.4	x	x	x	x
	T2	0.4	0.4	0.4	0.4	0.4	x	x	x	x
	T3	2.7	64.5	77.6	71.5	176.7	x	x	x	x
	T4	3.7	65.8	79.2	73.1	178.5	x	x	x	x
Version 2.2	Nodes	46	8473	6865	2083	7827	4054	7154	8435	7022
	T1	0.5	0.7	1.0	1.2	1.5	1.6	2.0	2.0	2.2
	T2	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	T3	4.1	20.8	17.2	10.3	17.2	37.5	16.3	10.7	16.6
	T4	5	21.9	18.6	11.9	19.1	39.5	18.7	13.1	19.2

- yeast

	k	2	3	4	5	6	7	8	9	10
Version 1.1	Nodes	2955	10233	x	x	x	x	x	x	x
	T1	1.6	1.8	x	x	x	x	x	x	x
	T2	1.3	1.3	x	x	x	x	x	x	x
	T3	16.3	744.4	x	x	x	x	x	x	x
	T4	19.2	747.5	x	x	x	x	x	x	x
Version 1.2	Nodes	1754	5795	x	x	x	x	x	x	x
	T1	1.3	1.9	x	x	x	x	x	x	x
	T2	1.3	1.4	x	x	x	x	x	x	x
	T3	1010.4	981.5	x	x	x	x	x	x	x
	T4	1013	984.8	x	x	x	x	x	x	x
Version 2.1	Nodes	2933	5797	4307	12659	10996	13027	10377	14481	18682
	T1	1.3	2.1	2.5	3	3.6	4	5	5.2	5.8
	T2	0.6	0.9	1	1	1	1	1	1	1
	T3	2.9	15	18.8	46.9	118.7	222.5	68.5	128.8	212.8
	T4	4.8	18	22.3	50.9	123.3	227.5	74.5	135	219.6
Version 2.2	Nodes	1737	5753	4229	12346	12685	18923	22215	22147	24118
	T1	1.3	2.1	2.4	3.2	4	4	4.6	5.1	5.7
	T2	0.8	1	1	1	1.1	1	0.9	1	1.1
	T3	5.2	13.2	17.8	53.6	137.1	161	385.5	622	602
	T4	7.3	16.3	21.2	57.8	142.2	166	391	628.1	608.8

C Solution Quality

The table blow express the results of the method FPF and our version 2.2 with the branching strategy of $I, G.D$. We define:

- D_{FPF}: Diameter found by the method FPF
- D_{OPT}: Diameter found by our model
- T_{FPF}: Time total of method FPF
- T_1: Time to find the solution optimum
- T_2: Total time of our method, to make sure that there is no better solution
- $Solution(I)$: Representatives of clusters in the solution optimum

iris

k	D_{FPF}	T_{FPF}	D_{OPT}	T_1	T_2	Solution (I)	Size of Groups
2	1.47	0.0	1.44	0.0	0.0	0, 1	82 68
3	1.18	0.0	0.94	0.0	0.0	0, 1, 2	48 50 52
4	1.18	0.0	0.87	0.1	0.1	0, 1, 2, 6	43 49 51 7
5	1.02	0.0	0.81	0.0	0.1	0, 1, 2, 4, 6	17 49 48 32 4
6	0.89	0.0	0.75	0.0	0.1	0, 1, 2, 3, 4, 5	18 35 43 15 27 12
7	0.84	0.0	0.59	0.0	0.1	0, 1, 2, 3, 4, 5, 6	14 35 30 15 21 15 20
8	0.76	0.0	0.56	0.0	0.1	0, 1, 2, 3, 4, 5, 6, 7	13 34 28 16 13 11 11 24
9	0.76	0.0	0.55	0.0	0.6	0, 1, 2, 3, 4, 5, 6, 7, 8	11 24 23 8 9 11 23 23 18
10	0.76	0.0	0.51	0.0	0.1	0, 1, 2, 3, 4, 5, 6, 7, 8, 10	8 23 30 8 12 11 14 16 19 9

ionosphere

k	D_{FPF}	T_{FPF}	D_{OPT}	T_1	T_2	Solution (I)	Size of Groups
2	4.72	0.03	4.54	0.4	0.6	0, 1	93 258
3	4.61	0.03	4.21	1.4	1.6	0, 1, 2	52 87 212
4	4.61	0.03	4.03	1.8	3.5	0, 1, 2, 3	34 48 194 75
5	4.61	0.03	3.84	3.9	4.5	0, 1, 2, 3, 4	22 26 186 75 42
6	4.15	0.03	3.78	5.7	6.1	0, 1, 2, 3, 4, 7	16 27 183 71 43 11
7	4.15	0.03	3.7	8.7	12.3	0, 1, 2, 3, 4, 5, 7	17 21 178 67 40 17 11
8	4.15	0.03	3.63	4.8	6.1	0, 1, 2, 3, 4, 5, 7, 10	16 7 149 43 24 11 10 91
9	4.03	0.03	3.57	4.6	6.2	0, 1, 2, 3, 4, 5, 6, 7, 12	4 20 168 26 31 9 77 6 10
10	4.03	0.03	3.48	6.3	14.8	0, 1, 2, 3, 4, 5, 6, 7, 8, 10	5 7 142 23 27 8 54 4 14 67

kdd_synthetic_control

k	D_{FPF}	T_{FPF}	D_{OPT}	T_1	T_2	Solution (I)	Size of Groups
2	33.42	0.1	33.42	0.1	0.8	0, 1	277 323
3	33.45	0.1	22.29	8.9	9.7	0, 1, 2	141 201 258
4	24.61	0.1	16.8	8.5	10	0, 1, 2, 3	85 122 192 201
5	16.57	0.1	13.48	5.1	6.3	0, 1, 2, 3, 4	50 81 172 155 142
6	16.52	0.1	11.16	8.2	9.9	0, 1, 2, 3, 4, 6	32 56 136 145 109 122
7	16.52	0.1	9.66	8.6	11	0, 1, 2, 3, 4, 6, 7	24 38 122 125 90 100 101
8	12.42	0.1	8.49	3.8	5.7	0, 1, 2, 3, 4, 5, 6, 7	22 25 103 110 65 82 99 94
9	8.9	0.1	7.6	1.9	3.7	0, 1, 2, 3, 4, 5, 6, 7, 8	17 21 93 98 79 75 79 87 51
10	8.55	0.1	6.73	2.2	6.1	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	15 20 86 96 74 55 52 71 39 92

vehicle

k	D_{FPF}	T_{FPF}	D_{OPT}	T_1	T_2	Solution (I)	Size of Groups
2	15.14	0.2s	15.14	3.9	5	0, 1	453 393
3	14.16	0.2s	10.16	20.6	21.9	0, 1, 2	196 212 438
4	11.1	0.2s	7.37	15.8	18.6	0, 1, 2, 3	125 166 354 201
5	7.34	0.2s	6.26	10.3	12	0, 1, 2, 3, 4	76 133 311 128 198
6	7.3	0.2s	5.29	12.8	19.6	0, 1, 2, 3, 4, 5	58 108 272 103 177 128
7	5.54	0.2s	4.5	17.9	39.5	0, 1, 2, 3, 4, 5, 8	51 107 180 89 132 95 192
8	5.36	0.2s	3.78	15.2	18.7	0, 1, 2, 3, 4, 5, 6, 7	39 76 206 74 148 127 86 90
9	4.7	0.2s	3.6	9.9	13.1	0, 1, 2, 3, 4, 5, 6, 7, 8	39 53 131 60 136 80 60 94 193
10	4.52	0.2s	3.42	9.8	19.6	0, 1, 2, 3, 4, 5, 6, 7, 8, 11	39 67 136 54 101 67 66 80 146 90

yeast

k	D_{FPF}	T_{FPF}	D_{OPT}	T_1	T_2	Solution (I)	Size of Groups
2	1.27	0.4s	1.24	4.6	7.3	0, 1	201 1283
3	1.19	0.4s	1.15	13.1	16.3	0, 1, 2	196 1179 109
4	1.03	0.4s	1.01	16.8	21.2	0, 1, 2, 3	128 784 15 557
5	1.03	0.4s	0.87	44.1	57.8	0, 1, 2, 3, 4	39 160 15 294 976
6	0.97	0.4s	0.82	122.8	142.2	0, 1, 2, 3, 4, 5	10 129 15 172 784 374
7	0.97	0.4s	0.75	145.6	166	0, 1, 2, 3, 4, 5, 6	13 116 15 172 758 282 128
8	0.93	0.4s	0.73	228.8	390.9	0, 1, 2, 3, 4, 5, 6, 8	578 78 15 5 429 243 110 26
9	0.85	0.4s	0.7	124.7	628	0, 1, 2, 3, 4, 5, 6, 7, 8	74 76 15 6 368 172 101 655 17
10	0.85	0.4s	0.67	337.1	766.2	0, 1, 2, 3, 4, 5, 6, 7, 9, 180	109 49 15 8 34 119 10 1117 8

D Instance-level constraints

Each test is called in 10 times. Int the tables below, we define:

- ML: number of must-link constraints
- CL: number of cannot-link constraints
- N_1: minimum number of nodes in the search tree in 10 times
- N_2: maximum number of nodes in the search tree in 10 times
- N_3: mean of number of nodes in the search tree in 10 times
- N_4: standard deviation of number of nodes in the search tree in 10 times
- D_1: minimum diameter in 10 times
- D_2: maximum diameter in 10 times
- D_3: mean of diameters in 10 times
- D_4: standard deviation of diameter in 10 times

- T_1: minimum total time in 10 times
- T_2: maximum total time in 10 times
- T_3: mean of total time in 10 times
- T_4: standard deviation of total time in 10 times

iris, n=150, k=3

ML	CL	N_1	N_2	N_2	N_{dev}	D_1	D_2	D_3	D_{dev}	T_1	T_2	T_3	T_{dev}
0	0			305				0.94				0.1	
10	0	168	296	232.00	64.00	0.94	1.09	1.02	0.08	0.08	0.11	0.10	0.01
20	0	154	156	154.60	0.92	1.09	1.09	1.09	0.00	0.08	0.11	0.09	0.01
30	0	143	194	183.80	20.40	1.00	1.09	1.02	0.04	0.08	0.10	0.09	0.01
40	0	226	263	233.40	14.80	1.09	1.20	1.18	0.04	0.09	0.11	0.10	0.01
50	0	108	247	191.40	68.10	1.15	1.21	1.17	0.03	0.06	0.10	0.09	0.01
0	10	282	293	286.40	5.38	0.94	0.94	0.94	0.00	0.10	0.12	0.12	0.01
0	20	158	223	203.50	29.79	1.05	1.21	1.10	0.07	0.10	0.13	0.11	0.01
0	30	158	261	209.50	51.50	1.04	1.21	1.13	0.09	0.10	0.12	0.12	0.01
0	40	227	308	245.00	31.95	0.94	1.17	1.12	0.09	0.11	0.14	0.12	0.01
0	50	318	514	405.30	92.40	0.94	0.94	0.94	0.00	0.12	0.23	0.17	0.05
10	10	285	290	287.50	2.50	0.94	0.94	0.94	0.00	0.08	0.13	0.10	0.01
20	20	165	304	206.70	63.70	1.09	1.21	1.13	0.05	0.10	0.11	0.10	0.00
30	30	209	329	315.90	35.78	1.15	1.21	1.20	0.02	0.09	0.13	0.11	0.01
40	40	117	118	117.30	0.46	1.21	1.21	1.21	0.00	0.08	0.10	0.09	0.01
50	50	107	108	107.60	0.49	1.21	1.21	1.21	0.00	0.07	0.10	0.09	0.01

ionosphere, n=351, k=2

ML	CL	N_1	N_2	N_3	N_{dev}	D_1	D_2	D_3	D_{dev}	T_1	T_2	T_3	T_{dev}
0	0			1280				4.54				2.2	
20	0	931	1225	1176.30	82.09	4.54	4.59	4.55	0.02	1.10	1.22	1.16	0.03
40	0	312	1424	1131.90	303.14	4.54	5.08	4.60	0.16	1.02	1.19	1.14	0.05
60	0	577	1340	990.20	185.71	4.54	4.62	4.56	0.03	1.07	1.17	1.14	0.03
80	0	885	1502	1085.20	184.39	4.54	4.59	4.56	0.02	1.14	1.20	1.16	0.02
100	0	252	1402	844.80	410.99	4.54	5.08	4.68	0.15	1.00	1.19	1.12	0.08
0	20	1200	1222	1203.00	6.40	4.54	4.58	4.54	0.01	2.08	2.59	2.20	0.19
0	40	1122	1447	1169.60	101.02	4.54	4.61	4.55	0.02	2.09	2.62	2.53	0.15
0	60	1039	1357	1205.30	121.18	4.54	4.63	4.56	0.04	2.12	2.62	2.47	0.18
0	80	753	1207	988.10	110.45	4.54	4.65	4.56	0.03	2.09	2.78	2.46	0.20
0	100	803	1451	980.70	190.50	4.54	4.62	4.56	0.03	2.07	2.67	2.51	0.15
20	20	1119	1424	1209.60	131.75	4.54	4.55	4.54	0.00	1.16	1.22	1.19	0.02
40	40	274	1238	904.90	331.24	4.54	4.94	4.61	0.12	1.01	1.23	1.15	0.07
60	60	235	1240	774.80	307.04	4.54	4.76	4.62	0.09	1.03	1.36	1.18	0.08
80	80	193	893	488.60	261.03	4.54	5.24	4.74	0.20	1.01	1.20	1.11	0.06
100	100	155	588	322.00	159.15	4.59	5.24	4.82	0.23	1.01	1.15	1.08	0.04

kdd_synthetic_control, n=600, k=6
In this data set, the tests is limited in 10 minutes. We count the number of times that the test exceed the time limit in the col *limit*.
The T_2 is always 600 seconds.

ML	CL	N_1	N_2	N_3	N_{dev}	D_1	D_2	D_3	D_{dev}	T_1	T_3	T_{dev}	limit
0	0			16946				11.16			57.5		
50	0	3252	83003	34257	27998	18.38	33.37	24.37	4.60	9	251	254	4
100	0	2910	109525	38357	43945	21.47	35.88	27.23	4.35	9	178	253	3
150	0	7147	79494	37173	28161	24.86	32.50	28.88	2.72	11	234	262	4
200	0	5347	70668	29530	26228	24.20	38.03	31.04	3.60	10	230	265	4
250	0	7504	107930	58752	30455	30.23	35.90	33.51	1.86	12	432	213	8
0	50	16417	141578	58791	44891	11.94	19.10	15.44	2.02	41	293	204	4
0	100	10091	207848	104864	60378	14.63	19.24	15.81	1.29	90	441	134	6
0	150	20316	236024	108831	57465	14.53	19.00	17.22	1.37	95	502	141	9
0	200	18557	286105	118489	71363	15.32	17.82	16.83	0.71	117	441	153	9
0	250	29086	169026	96456	45240	16.11	19.39	17.72	0.94	195	459	101	8
50	50	8092	78402	44968	30310	19.55	31.87	24.16	3.34	16	294	258	5
100	100	2704	93370	33743	32114	25.61	38.03	30.21	3.84	9	222	258	4
150	150	4670	97000	37203	33348	20.78	35.90	29.93	4.56	9	213	238	4
200	200	11161	33849	24310	7303	27.11	36.20	30.05	2.63	20	208	229	2
250	250	4665	54914	30705	16728	27.14	38.03	33.12	2.91	10	412	239	6