Incremental Constrained Clustering by Minimal Weighted Modification

Aymeric Beauchamp, Thi-Bich-Hanh Dao, Samir Loudni, Christel Vrain

Incremental Constrained Clustering by Minimal Weighted Modification

Aymeric Beauchamp
University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Thi-Bich-Hanh Dao
University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Samir Loudni
TASC (LS2N-CNRS), IMT Atlantique, France
GREYC, University of Caen Normandy, France

Christel Vrain
University of Orléans, INSA Centre Val de Loire, LIFO EA 4022, France

Abstract

Clustering is a well-known task in Data Mining that aims at grouping data instances according to their similarity. It is an exploratory and unsupervised task whose results depend on many parameters, often requiring the expert to iterate several times before satisfaction. Constrained clustering has been introduced for better modeling the expectations of the expert. Nevertheless constrained clustering is not yet sufficient since it usually requires the constraints to be given before the clustering process. In this paper we address a more general problem that aims at modeling the exploratory clustering process, through a sequence of clustering modifications where expert constraints are added on the fly. We present an incremental constrained clustering framework integrating active query strategies and a Constraint Programming model to fit the expert expectations while preserving the stability of the partition, so that the expert can understand the process and apprehend its impact. Our model supports instance and group-level constraints, which can be relaxed. Experiments on reference datasets and a case study related to the analysis of satellite image time series show the relevance of our framework.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Computing methodologies → Semi-supervised learning settings

Keywords and phrases Incremental constrained clustering, Constrained optimization problem, User feedback

Digital Object Identifier 10.4230/LIPIcs.CP.2023.38

Related Version *Extended Version*: `hal-04158825`

Supplementary Material *Software (Source Code)*: `https://github.com/aymericb213/IAC`

Funding This work was supported by the French national research project HERELLES under grant agreement ANR-20-CE23-0022.

Acknowledgements The authors want to thank the anonymous reviewers for their comments and suggestions which helped to improve this paper.

1 Introduction

Clustering is a popular task in Data Mining in which data instances (i.e. points of a dataset) are grouped into distinct clusters according to their similarity. Over time, many strategies have been explored to compute data partitions, each of them having their own strengths and biases. Constrained clustering [16] aims to find relevant clusters by stating some desired properties in the form of constraints, thus alleviating the aforementioned biases. The most

commonly used constraints for clustering state that two points must be clustered together (*must-link*) or apart (*cannot-link*) [32]. Other constraints exist as for instance limiting the size or the diameter of the clusters [1]. They are usually given by a subject matter expert (SME) that possesses domain knowledge on the data; they model his/her knowledge and expectations about the result by expert constraints.

In practice, it may be difficult for an expert to express constraints solely on the basis of the data. It is usually easier to him/her to provide feedback on the current partition to refine it. This leads to a *human-in-the-loop* clustering process, where new constraints given or validated by the expert are incrementally integrated, modifying the result until user satisfaction. However, this raises several non trivial questions such as how to elicit the constraints, how to integrate them, or how to further exploit them. By incorporating human feedback and domain knowledge, the resulting clusters are more likely to align with the expert expectations while making them more intuitive and interpretable. This incremental setting mimics the natural step-by-step progression of an exploratory task such as clustering, where the user needs to iterate several times before satisfaction. In such a process, the result at each step should (1) be computed fast enough, (2) exploit the expert constraints efficiently (3) be similar to the result of the previous step, in order not to disturb the expert.

A naive way to integrate new expert constraints is to restart a constrained clustering algorithm. However, this presents at least two weaknesses: it starts from scratch without ever considering intermediate results and the new constraints can lead to a partition very different from the one previously shown to the expert. To cope with this problem, we propose a first generic framework that allows for truly interactive and iterative constrained clustering that fulfills the conditions mentioned above. Our main contributions are :

- an incremental constrained clustering framework designed for human interaction, combining active constraint selection and clustering modification;
- a new constraint programming model for minimal clustering modification, which ensures the stability of the partition.

The paper is organized as follows. We review in Section 2 related work on incremental constrained clustering and minimal clustering modification and present our method in Section 3. Experiments on reference and satellite image time series datasets are presented in Section 4 and perspectives are discussed in Section 5.

2 Related Work

The first works on using constraints in clustering are extensions of classic algorithms for handling must-link/cannot-link constraints [33, 10, 35]. They either search for a solution satisfying all the constraints [33] or a compromise between constraint satisfaction and clustering quality [6]. Thereafter methods allowing to integrate more general constraints, using declarative frameworks such as SAT [12], ILP [29] or CP [9] have been proposed. When new constraints are given by the user, all these methods require to restart from scratch, without any guarantee that the new partition will be similar to the previous one. Therefore they are not suited for an incremental setting.

There is a growing body of works related to incremental constrained clustering. In [7], the idea of gathering feedback from an existing result rather than expecting the user to provide insightful constraints by themselves is demonstrated. Later, the authors of [11] studied the problem of adding or removing a constraint from a constraint set satisfied by a partition. They described conditions under which the problem is easy to solve and an algorithm working under these conditions. In [26], a cluster refinement framework uses subclustering to find

representatives to present to the user before learning an embedding using the feedback. More recently, [21] proposed an incremental variant of a collaborative constrained clustering system that integrates constraint satisfaction in its objective function. Among declarative approaches, an ILP model for minimal clustering modification (MCM) is proposed in [20]. It constrains cluster diameters with the objective of removing undesirable properties from a partition. Another ILP model [28] computes a membership score for each point to each cluster and optimizes a criterion based on this score. It can modify the assignment of points to clusters, even if they are not involved in constraints. However those models are restrictive since they cannot handle conflicting constraints, which make the problem unsolvable.

Our approach seeks to preserve the general cluster structure of an existing partition while satisfying new constraints. To the best of our knowledge, this is the first approach explicitly tackling both quality and stability. It is based on CP, therefore it can integrate different kinds of constraints that can be either soft or hard, with some control over constraint relaxation and the ability to handle conflicting constraints. We use subclustering in a similar way to [26], albeit for a different purpose as it allows generalizing the changes decided by our CP model as well as finding cluster representatives used in our objective function.

3 Incremental and Active Clustering Framework

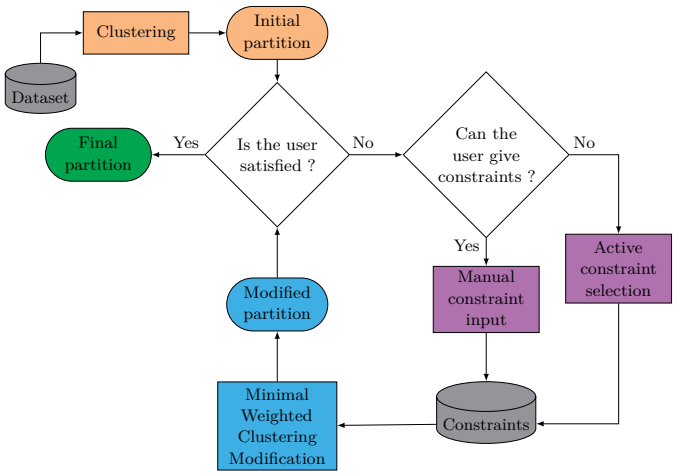


Figure 1 Schematic view of the incremental clustering cycle.

In this section, we describe our proposed incremental and active clustering (IAC) framework. Figure 1 gives a general overview of IAC. The incremental constrained clustering loop starts from an initial partition computed by any clustering algorithm. This partition is then shown to the user to collect his/her general feedback (satisfied or not). If he/she is not satisfied, he/she can modify this partition in two ways: by manually providing a set of constraints and/or by inferring it through an active constraint selection method. He/she can also set the proportion of constraints to satisfy as well as the scope of modifications. The clustering modification step updates the current partition according to these new constraints, optionally generalizing modifications to unconstrained data instances. The output is a new partition satisfying the constraints while preserving its stability, i.e., similar to the previous one. This process is repeated until the user is satisfied by the resulting partition. It is noteworthy that our framework is generic as any active constraint selection method and any modification algorithm could be used as long as the constraints generated match the

constraints handled in modification. We formulate the problem of clustering modification in a declarative way and present a CP model. This has the benefit of being able to integrate several types of constraints for the user feedback.

3.1 Minimal Weighted Clustering Modification

We consider the minimal weighted clustering modification (MWCM) problem: given a partition $\mathcal{P}$ of N data instances (numbered from 1 to N) into a number K of clusters, and a set of user constraints $\mathcal{C}$, the objective is to find a new partition $\mathcal{P}'$ such that the constraints are satisfied while minimizing some function f modeling the difference between $\mathcal{P}$ and $\mathcal{P}'$. For solving this problem, Algorithm 1 shows the different steps that are detailed below.

Algorithm 1 Minimal Weighted Clustering Modification

Input: Dataset $\mathcal{X}$, partition $\mathcal{P}$, constraints $\mathcal{C}$, anchor generation rate α, super-instance rate β, constraint satisfaction rate δ

Output: modified partition $\mathcal{P}'$

1: $anchors \leftarrow$ ComputeRepresentatives$(\mathcal{X}, \mathcal{P}, \alpha)$ ▷ See section 3.1.1
2: $X \leftarrow$ ComputeCOPInstances$(\mathcal{X}, \mathcal{P}, \mathcal{C}, \beta)$ ▷ Instances used in COP (Section 3.1.2)
3: $\mathcal{D} \leftarrow$ DistanceMatrix$(X, anchors)$ ▷ See section 3.1.1
4: $p \leftarrow$ GetConstrainedPartition$(X, \mathcal{P})$ ▷ Cluster membership of the constrained instances
5: $mods \leftarrow$ SolveModel$(\mathcal{D}, p, \mathcal{C}, \delta)$ ▷ Solves the COP in Section 3.2
6: **return** ApplyModifications$(mods, \mathcal{P})$ ▷ Updates $\mathcal{P}$ and **generalizes** modifications

3.1.1 Objective function and anchors

A straightforward candidate for f is to count the number of instances that have changed their cluster membership between $\mathcal{P}$ and $\mathcal{P}'$ [20]:

$$\arg\min \sum_{i=1}^{N} \mathbb{I}(\mathcal{P}[i] \neq \mathcal{P}'[i]) \tag{1}$$

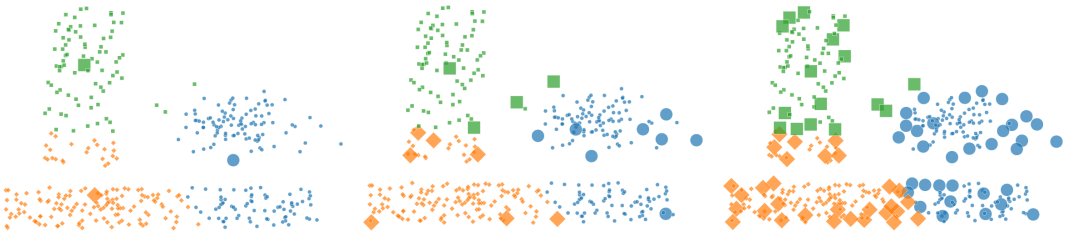
where $\mathcal{P}[i]$ denotes the number $c \in [1, K]$[1] of the cluster containing instance $i \in [1, N]$ and $\mathbb{I}$ the indicator function that returns 1 if the expression given as argument is true, and 0 otherwise. The main drawback of Equation (1) is that it does not take into account the structure of the clusters. For example, it is reasonable to consider that putting two instances in a nearby cluster is more akin to the idea of minimal modification than putting one instance into a faraway cluster. Therefore we propose an alternate objective function that integrates a distance-based weighting of the modifications:

$$\arg\min \sum_{i=1}^{N} \mathbb{I}(\mathcal{P}[i] \neq \mathcal{P}'[i])\ \mathcal{D}[i, \mathcal{P}'[i]] \tag{2}$$

where $\mathcal{D}$ is a distance matrix of dimensions $N \times K$ such that $\mathcal{D}[i, c]$ is the distance of instance i to cluster c. This objective function measures the changes between $\mathcal{P}$ and $\mathcal{P}'$ and keeps the cumulative distances resulting from these changes small. This objective function is integrated into the model for MWCM that will be presented in Section 3.2.

[1] We use the notation $[1, K]$ for the set $\{1, .., K\}$.

A simple way to compute $\mathcal{D}[i, c]$ is to set $\mathcal{D}[i, c] = d(i, \mu_c)$, where μ_c is the representative (medoid or centroid) of c and d is a distance measure, typically the Euclidean distance. This has however a known limitation: the modifications made by the model implicitly treat all clusters as spherical. It can be counterproductive if the user seeks more complex shaped clusters. To overcome this, we use anchors [17] such that each cluster will be represented by a subset of its instances, in order to better represent its structure. Anchors are computed by dividing each cluster c of the input partition into smaller sub-clusters using Single-Link hierarchical clustering. For each of these sub-clusters, an anchor is defined as the instance minimizing the sum of its distances with the other instances of the sub-cluster. Using anchors, $\mathcal{D}[i, c]$ represents the distance of instance i to its closest anchor belonging to cluster c. A parameter α defining the proportion of instances per cluster (in percentage) that will become anchors is used. For example, a rate of 0% means that we only use medoids, while at a rate of 100%, all instances are anchors (cf. Figure 2).



(a) $\alpha = 0\%$ (medoids) **(b)** $\alpha = 5\%$ **(c)** $\alpha = 20\%$

Figure 2 Anchor positions for different values of α computed from the partition generated by Kmeans on `lsun` dataset. The anchors are represented bigger than normal instances.

3.1.2 Generalizing constraints with super-instances

In a real use case, we assume that the expert will only react on a small number of instances per iteration. As a result, the clustering modification could become unnoticeable when the dataset size is large compared to the number of constraints, which is a fairly common case. Hence, exploiting expert feedback to *generalize* the modifications to relevant unconstrained instances is an important issue, while asking a reasonable number of queries. This issue has been highlighted in [34], and is especially relevant in our incremental setting. Furthermore, this generalization must be controlled to ensure the expert can grasp the scope of potential modifications. We assume that, in most cases, the expert will want to modify a zone around the selected instances and not only the instances themselves. Bearing this intuition in mind, making use of nearest neighbors or a proximity radius seem adequate, but these methods lack predictability: if an instance is within the radius of two constrained points who were reassigned to different clusters, determining how to resolve the generalization is not obvious.

We propose to use super-instances, i.e. virtual instances grouping several real data points, to generalize the modifications. Note that *generalization* does not mean that new constraints are generated, rather that the super-instances are passed directly to the model instead of the data instances (see Appendix C). Thus any modification in the cluster membership of a super-instance amounts to changing the membership of every real data instance that compose it. The generalization scope is controlled as follows: *the less a cluster is divided, the stronger the impact of a modification.* Thus the scope depends on the number of super-instances, determined by a rate β proportional to the cluster size. As such, setting β to e.g. 10% means that each cluster will be split into a number of super-instances equal to 10% of its

size, with 1 being equivalent to not generalizing at all. Super-instances are determined by sub-clustering each cluster of the current partition into small groups, each group representing a super-instance. We empirically found that *complete-link* hierarchical clustering is adequate, despite its memory usage which makes it unsuitable on large datasets. Other alternatives include the density-based OPTICS as in [26], or the Furthest Point First (FPF) algorithm [15]. However, user constraints defined on instances need to be transferred to super-instances. This may raise potential conflicts. To avoid this pitfall, we ensure that every super-instance contains no more than one constrained data point. If this is not the case, we split the super-instance using the constrained instances as centers of the new split super-instances. An illustrative case is given in Appendix A.

3.2 Constraint Optimization Problem Formulation

Taking advantage of declarative approaches, we formulate the problem of finding a similar partition satisfying user constraints as a Constraint Optimization Problem (COP). In the following, we use the term *instance* to denote a data instance or a super-instance if it is used.
Variables and Objective Function. Only instances that are subject to the constraints will be concerned by the COP. Function GetConstrainedPartition in Algorithm 1 produces the subset X containing the constrained instances. For each instance in $i \in X$, we define a variable G_i with the domain $[1, K]$, where $G_i = c$ means instance i is assigned to cluster c in the new partition $\mathcal{P}'$. Using Eq. (2), the objective function is:

$$\arg\min \sum_{i \in X} \mathbb{I}(G_i \neq \mathcal{P}[i])\ \mathcal{D}[i, G_i] \tag{3}$$

User constraints. Several instance-level and group-level constraints can be expressed in our model, as below. Must-link (ML)/cannot-link (CL) constraints on two instances i, j stating that the instances must/cannot be in the same cluster, can be expressed by $G_i = G_j$ for ML and $G_i \neq G_j$ for CL. We also compute the transitive closure on ML/CL constraints [25], which derives supplementary constraints according to three rules : (i) if $ML(a, b)$ and $ML(b, c)$, then $ML(a, c)$; (ii) similarly $ML(a, b)$ and $CL(b, c)$ imply $CL(a, c)$; (iii) in a binary clustering case ($K = 2$), $CL(a, b)$ and $CL(b, c)$ imply $ML(a, c)$.
Triplet constraint (a, p, n) [23] states that a reference instance a is more similar to instance p than to instance n. Instance p is therefore called positive instance and n negative. This constraint can be expressed using an implication constraint as follows:

$$G_a = G_n \implies G_a = G_p \tag{4}$$

Span-limited constraints [27] restricting the span of a set of instances $S \subseteq X$. A *specific* span-limited constraint states that the instances of S must be assigned only to clusters from a given subset $C \subseteq [1, K]$. It can be expressed using the *count* global constraint[2]:

$$count(c, [G_i \mid i \in S], =, 0) \quad \forall\ c \notin C \tag{5}$$

A *generic* span-limited constraint specifies that the instances of S must be assigned to at most a number γ of clusters. It can be expressed using the constraint *atmost_nvalue*[3] [5]:

$$atmost\_nvalue(\gamma, [G_i \mid i \in S]) \tag{6}$$

[2] https://sofdem.github.io/gccat/gccat/Ccount.html
[3] https://sofdem.github.io/gccat/gccat/Catmost_nvalue.html

More thematic expert feedback can also be integrated as implication constraints. In our model, they are of the form $P \implies Q$, where P and Q are conjunctions of simpler constraints such as ML/CL.

Cluster creation. Our model allows $\mathcal{P}'$ to have more clusters than $\mathcal{P}$, by defining the domain of G_i as $[1, K']$, with $K' > K$. This enables assigning an instance to a new cluster if some constraints are unsatisfied. For example, if $K = 2$ and the expert states three constraints $G_i \neq G_j, G_j \neq G_l$ and $G_l \neq G_i$, then it is necessary to create a new cluster. Given its conditions of apparition, it will typically be very small. In order to prevent the model from creating clusters because it would be optimal to do so w.r.t. the objective function, we set the distance $\mathcal{D}[i, k']$ to a value greater than all other distances for $K < k' \leq K'$.

Relaxing constraints. During the incremental process, the expert could make mistakes when trying to improve the partition, which would result in adding conflicting constraints, thus leading to an over-constrained CP model. We reify the user constraints to gain control over constraint satisfaction. Each constraint $c \in \mathcal{C}$ is associated with a Boolean variable S_c such that $S_c = 1$ iff c is satisfied. The satisfaction rate δ sets a lower/upper bound or the exact value of the number of constraints the model must satisfy:

$$\sum_{c \in \mathcal{C}} \mathcal{S}_c \gtreqless \delta \cdot |\mathcal{C}| \tag{7}$$

Constraint relaxation can both solve problems with conflicting constraints and ignore - or warn the user about - constraints that would modify instances far from their new cluster. Relaxation however increases runtime due to the additional Boolean variables. As is, our framework automatically detects ML/CL conflicts and reduces the satisfaction rate accordingly.

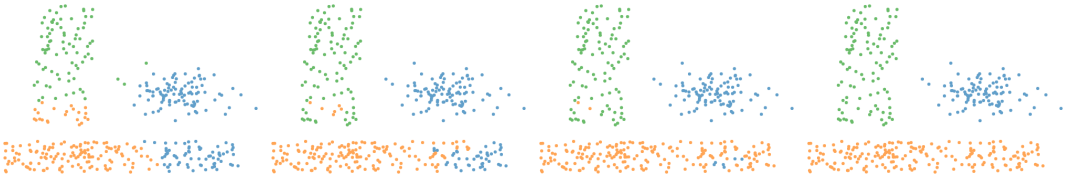
Managing the constraint store. Algorithm 1 solves this COP with the user constraints collected at each iteration. The incremental setting raises the issue of managing the constraints between iterations. It is possible to store every constraint received since the beginning of the process to ensure that all expert feedbacks are respected. In the experiments, we choose instead to treat the constraint set given at each iteration independently. In this way, if the expert adds a constraint in conflict with another one given previously, we consider that the user is simply rescinding some of his/her feedback. The expert can also mark some constraints as mandatory so that they are kept satisfied throughout the process. Appropriately managing the constraint store is a potential future research lead.

3.3 Active Constraint Selection

Obtaining constraints manually can be costly. This motivates active constraint selection methods [2, 25, 37], which select the most informative constraints to query. To evaluate the interest of exploiting an active constraint selection approach within our framework, we use NPU [37], a neighborhood-based sampling strategy. Neighborhoods $\mathcal{N}$ are groups of instances whose cluster assignment is certain, they represent the underlying clusters. NPU iteratively builds the neighborhoods by selecting the most informative instance x^* and querying its relation with respect to existing neighborhoods. The informativeness of an instance x is defined by the ratio $H(\mathcal{N}|x)/\mathbb{E}[q(x)]$, where $H(\mathcal{N}|x)$ is the entropy measure of the uncertainty to assign x to a neighborhood in $\mathcal{N}$, and $\mathbb{E}[q(x)]$ the expected number of queries needed to discover its neighborhood.

Exploiting this informativeness, we adapt the NPU framework to the incremental setting. The neighborhoods $\mathcal{N}$, which are initially empty, are constructed and kept throughout the iterations. For each informative instance x^*, queries are put on the membership relation between x^* and each neighborhood $N \in \mathcal{N}$. Once the user answers favorably, a ML constraint

is created with the queried neighborhood, otherwise a CL is created. If no ML is achieved, a new neighborhood is created for x^* (see the detailed algorithm in Appendix B). In relation to constraint management between iterations, it must be pointed out that the preservation of the neighborhoods between iterations prevents selecting constraints conflicting with those selected in earlier iterations. Indeed, an instance stored in a neighborhood is never picked again by NPU, and is only used to generate constraints with an instance that has not been presented to the user before. The only factor that could cause previous constraints to be involuntarily relaxed is a high generalization scope. We found no such occurences in our experiments with the values we tested for β (see Section 4.2.1). Fig. 3 illustrates the framework walkthrough on a toy example, with noticeable separation between non-spherical clusters that KMeans is unable to recover.



(a) $t = 0, ARI = 0.44$ **(b)** $t = 4,\ ARI = 0.66$ **(c)** $t = 7,\ ARI = 0.94$ **(d)** $t = 10,\ ARI = 1$

Figure 3 Illustration of IAC with ($\alpha = 20\%, \beta = 30\%$) over 10 iterations on `lsun` dataset, starting from a KMeans partition (Fig. 3a). Subsequent figures show the evolution of the partition after t iterations of IAC with NPU. Adjusted Rand Index with ground truth is reported on each figure.

4 Experiments

In this section, the experiments aim to answer the following research questions (RQ):

1. What effect do IAC parameters (α and β) have on clustering results? (Section 4.2.1)
2. How does our CP model scale with the number and type of constraints? (Section 4.2.2)
3. How does the constraint relaxation of IAC compare with other methods that use soft constraints? (Section 4.2.3)
4. How effective is IAC in an active constraint selection context? (Section 4.2.4)
5. What is the performance of our framework in terms of clustering quality, partition similarity and runtime when compared to state of the art methods? (Section 4.2.4)
6. How effective is our framework on a real use case with human feedback? (Section 4.3)

4.1 Experimental Methodology

Evaluation measures. For all experiments we use datasets for which a *ground-truth* labeling is known. The produced partition is then compared to the known partition using an external measure. A high value of the measure indicates a good partition, meaning that the clustering algorithm has successfully identified the already known structure. We consider three measures: Adjusted Rand Index (ARI) [19], Adjusted Mutual Information (AMI) [31] and Folkes-Mallows Index (FMI) [13]. ARI measure is defined by:

$$ARI(\mathcal{P}, \mathcal{P}') = \frac{2(ab - cd)}{(a+d)(d+b) + (a+c)(c+b)} \tag{8}$$

where a (resp. b) is the number of instances pairs clustered together (resp. apart) in $\mathcal{P}$ and $\mathcal{P}'$, and c (resp. d) is the number of pairs clustered together (resp. apart) in $\mathcal{P}$ and apart

(resp. together) in $\mathcal{P}'$. AMI is a variation of Normalized Mutual Information corrected for chance:

$$AMI(\mathcal{P}, \mathcal{P}') = \frac{MI(\mathcal{P}, \mathcal{P}') - E(MI(\mathcal{P}, \mathcal{P}'))}{\max(H(\mathcal{P}), H(\mathcal{P}')) - E(MI(\mathcal{P}, \mathcal{P}'))} \tag{9}$$

where MI is the measure of mutual information, H the entropy of a partition, and E the expected mutual information if the partitions are random. Finally FMI is the geometric mean of precision and recall, using one partition as a reference for the other:

$$FMI(\mathcal{P}, \mathcal{P}') = \sqrt{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}} \tag{10}$$

We also measure the runtime of the modification step (reclustering or MWCM) since it is crucial to take it into account in the incremental setting. We set a timeout after 1 hour of modification time.

Experimental protocol. We have implemented our CP model in Python 3.11 using the `CPMpy` library [18], interfacing with CP-SAT solver from `or-tools`[4]. We chose this library because it allows to easily use landmark ML libraries such as `scikit-learn` together with CP. Code for reproducing the experiments is available at the repository given at the summary of this paper. The implementation of `NPU` and all clustering algorithms we considered for comparison (COPKMeans, PCKMeans and MPCKMeans) are from the `active-semi-supervised-clustering`[5] library. All experiments were run on a computer with two 48-core Intel Xeon processors at 4 GHz and 64 GB of RAM running Ubuntu 20.04.

For each dataset, we first generate an initial partition with KMeans [24] with K set to the true number of clusters. Queries correspond to pairwise constraints. We emulate user feedback using the ground truth labeling of the data as an oracle i.e. a must-link constraint is added if the selected pair of instances belong to the same class, and a cannot-link otherwise. For **RQs 1**, **4** and **5**, we perform 10 iterations of selection-modification loop. At each iteration, we use the current partition to select a batch of 10 queries with `NPU`, get feedback from the oracle, and apply either a constrained clustering algorithm to the full dataset or our CP model for cluster modification. In order to smooth out the random effects occurring in the partition initialisation and in constraint selection, we repeat each experiment 90 times.

To evaluate the overall performance over the 11 successive partitions (including the initial partition) obtained for each run, we compute for each metric the area under the budget curve (AUBC) [38]) for different fixed budgets of queries to ask the user. Given the budget curve, the AUBC is calculated by the trapezoid method, and the higher value reflects better performance of the evaluated method under varying budgets. For each metric, we compute two types of AUBC: $AUBC_{quality}$ when comparing the successive partitions to the ground truth partition, and $AUBC_{similarity}$ when comparing two consecutive intermediate partitions. Since $AUBC_{similarity}$ values are defined over the interval $[0, 0.9]$, we perform a min-max normalization so that all metrics are defined over the $[0, 1]$ range. To statistically compare the performance of different algorithms and/or configurations of the same algorithm for different parameter settings on multiple data sets, we resort to Bayesian pairwise comparison [4] using `baycomp`[6] library. The principle is to use Bayes' rule to update a prior statistical distribution representing the null hypothesis (both compared algorithms have the same performance),

[4] `https://developers.google.com/optimization`
[5] `https://github.com/datamole-ai/active-semi-supervised-clustering`
[6] `https://baycomp.readthedocs.io/en/latest/index.html`

UCI				FCPS			
Name	(N, A, K)	Name	(N, A, K)	Name	(N, A, K)	Name	(N, A, K)
`iris`	(150, 4, 3)	`ionosphere`	(351, 34, 2)	`lsun`	(400, 2, 3)	`chainlink`	(1000, 3, 2)
`wine`	(178, 13, 3)	`yeast`	(1484, 8, 10)	`target`	(770, 2, 6)	`wingnut`	(1016, 2, 2)
`sonar`	(208, 60, 2)	`statlog`	(2310, 19, 7)	`atom`	(800, 3, 2)	`engytime`	(4096, 2, 2)
`glass`	(214, 9, 6)	`Letters`	(20000, 16, 26)				
`ecoli`	(336, 7, 8)	`MNIST`	(70000, 784, 10)				

Table 1 Dataset Characteristics, with N the number of instances, A the number of features and K the number of clusters or classes.

with a likelihood function modeling the experimental observations. We then get a posterior distribution, reflecting how the prior belief has changed, taking the observations into account. Using the Markov chain Monte Carlo method, the posterior is sampled $50,000$ times to estimate the probability of one algorithm being better than the other as well as the probability of being in the region of practical equivalence (or *rope*). In practice, querying the posterior distribution allows to simulate repeating the whole experimental process and to quantify the likelihood of our results. We choose to fix to 1% the difference of performance between the methods as the rope.

4.2 UCI and FCPS Datasets

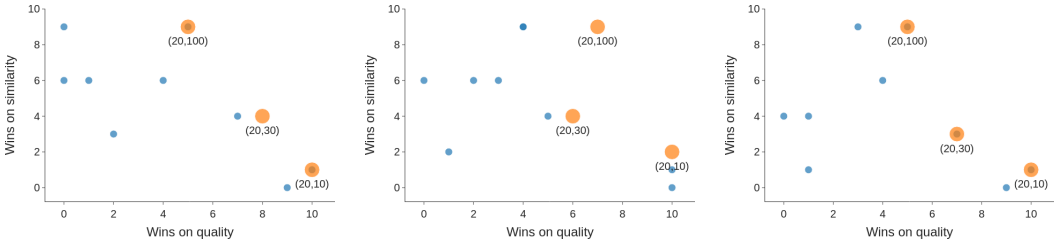
In this section, we report experimental results on ten real-world datasets from the UCI repository[7] and on six synthetic datasets from the FCPS [30] suite designed to address specific challenges to the clustering algorithms such as lack of linear separability, classes defined by data density rather than data spacing, no cluster structure at all, etc. A summary of the basic characteristics is given in Table 1. We used the versions of datasets available under the library `clustering-benchmarks`[8] [14].

4.2.1 Parameter Settings of IAC

To answer **RQ1**, we evaluate the effects of different parameter settings of the clustering modification step of our IAC framework: the anchor generation rate $\alpha \in \{0\%, 5\%, 20\%\}$ and the super-instance generation rate $\beta \in \{10\%, 30\%, 50\%, 100\%\}$. This makes a total of 12 configurations of parameter combinations to evaluate. Recall that $\alpha = 0\%$ means that we only compute the cluster medoids, while $\beta = 100\%$ means no generalization by super-instances is performed. For each configuration and each metric, we perform Bayesian pairwise comparison according to $AUBC_{quality}$ and $AUBC_{similarity}$ values for each of the three metrics ARI, AMI and FMI over all the datasets and count the number of wins. More precisely, given a pairwise comparison between two configurations $conf_1$ and $conf_2$, using a Bayesian hierarchical model [8], we get three probabilities: the probability that $conf_1$ has higher scores than $conf_2$, the probability that differences are within the region of practical equivalence (rope), or that $conf_2$ has higher scores. If $(p_{conf_1} > p_{conf_2} + p_{rope})$, then we count this comparison as winning for $conf_1$.

The table of wins is available in Appendix D. Configurations using anchors (i.e. $\alpha \neq 0$) ensure the highest $AUBC_{quality}$ values in almost all configurations. Additionally, the best

[7] `https://archive.ics.uci.edu/ml/index.php`
[8] `https://clustering-benchmarks.gagolewski.com/weave/suite-v1.html`

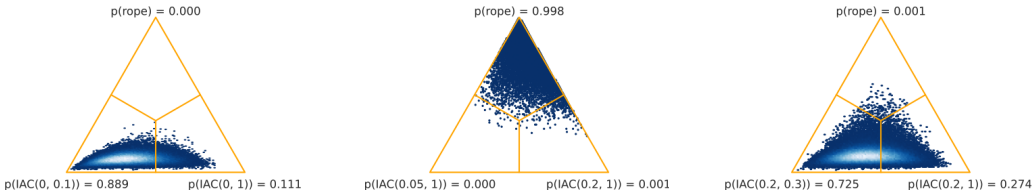


(a) ARI **(b)** AMI **(c)** FMI

Figure 4 Pairwise plot comparing the number of wins for each configuration (α, β) using a Bayesian hierarchical model, according to $AUBC_{quality}$ (horizontal) or $AUBC_{similarity}$ (vertical). Best values are close to the top left, with configurations on the Pareto front in orange.

values are obtained with $\alpha = 20\%$. For a fixed value of α, $AUBC_{quality}$ values increase significantly with the decrease of β, 10% being the best value. This result suggests that a large scope of generalization does not strongly impact the modification of the clustering and thus its quality. $AUBC_{similarity}$ values have the opposite behavior: $\beta = 10\%$ is the worst setting for similarity. Note that the impact of α seems negligible compared to β. It slightly improves similarity when β is low, while its contribution seems negligible in the absence of generalization. Generalizing modifications understandably degrades similarity, albeit not dramatically. A Pareto front made of three configurations on every metric (see Fig. 4) emerges from the results: one best in quality $(20\%, 10\%)$, another best in similarity $(20\%, 100\%)$, and a compromise setup $(20\%, 30\%)$.

Figure 5 plots the posterior distribution of pairwise comparison of configurations in a simplex. The distribution is shown as a triangle with regions corresponding to different samples of the distribution, e.g. Figure 5a compares the posterior distribution obtained with $\alpha = 0\%$ (medoids only) and $\beta = 10\%$ with the one obtained without generalization. This indicates that there is a 88.9% probability that using medoids combined with SI for this value of β is better than using medoids without SI.

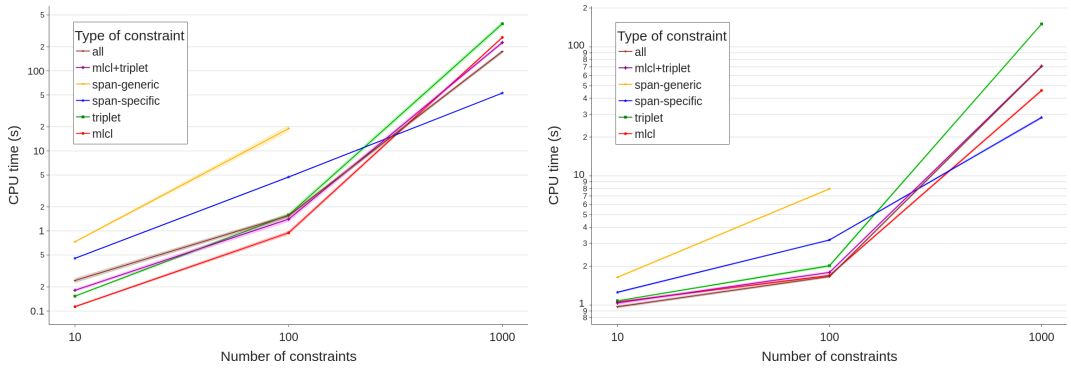


(a) $(0, 10\%)$ vs. $(0, 100\%)$ **(b)** $(5\%, 100\%)$ vs. $(20\%, 100\%)$ **(c)** $(20\%, 30\%)$ vs. $(20\%, 100\%)$

Figure 5 Simplex view of Bayesian comparison of two configurations w.r.t $AUBC_{quality}$ of ARI. Each sample is plotted according to probabilities p_{conf_1} (left), p_{rope} (top) and p_{conf_2} (right).

4.2.2 Impact of Number and Types of User Constraints on Runtime

In this section, we answer **RQ2** by studying two points: the computational efficiency of our CP model and the expressiveness of our approach (see Sect. 3.2). For these experiments, we consider 3 sizes of constraint set (10, 100, 1000). For each test case, we randomly generate sets of four types of constraints (pairwise, triplet, span-limited specific and generic). We compute an initial KMeans partition and run our CP model only once for each set of constraints and we report the average CPU times over 90 runs. For pairwise constraints, we disable the



(a) `letters` **(b)** `MNIST`

Figure 6 Evolution of running time of our CP model for the two largest datasets when varying the number and type of constraints, with 95% confidence interval. CPU times are in log-scale.

computation of transitive closure to keep the number of constraints unchanged. Span-limited constraints are created by randomly choosing 10 instances and finding the ground truth set of clusters - or number of clusters, in the generic case - to which the group belongs.

Runtime analysis. Figure 6 shows the results we obtained for `letters` and `MNIST` datasets with $\alpha = 0\%$ and $\beta = 100\%$. Our CP model can process 10 ML/CL constraints in less than 0.05 seconds, while for triplet and specific span-limited constraints the runtime reaches 0.035 and 0.15 seconds respectively. This seems very reasonable in an incremental context. For the `yeast` dataset, with 100 pairwise constraints, it takes 0.35 seconds, whereas for triplet and span-limited constraints the runtime increases up to 1.36 seconds. With 1000 constraints, the runtime is more than 44s for specific span-limited, 67s for ML/CL, and over 180s for triplet; generic span-limited constraints took up too much memory to finish. However, in practice, the number of constraints expected from the user is in the tens rather than the hundreds or thousands. Triplet and span-limited constraints show a substantial increase in runtime compared to ML/CL constraints.

Mixed constraint types. One of the advantages of our approach is its ability to easily combine different types of constraints without the need to create a specialized algorithm. To demonstrate this ability, we compared two composite settings:

- **mlcl+triplet**: generate pairwise and triplet constraints in equal proportions, e.g. 50 ML/CL and 50 triplet constraints for the 100 constraints case.
- **all**: similar to **mlcl+triplet**, except a pairwise (resp. triplet) constraint is replaced by a specific (resp. generic) span-limited constraint.

As far as we are aware, such problems cannot be solved by any existing techniques. Problems with ML/CL and triplet constraints take much less time to solve than those involving the three types of constraints, the latter being much more time-consuming. Surprisingly, with 1000 constraints, the ML/CL+Triplet combination takes less time compared to the case where only triplet constraints are involved. All these results underline the relevance of carefully selecting a small number of constraints to guarantee a good compromise between efficiency and quality of the final clustering.

4.2.3 Relaxing Constraints

We address the research question **RQ3** by comparing IAC with existing constrained clustering methods for constraint relaxation. We selected two methods: *Pairwise Constrained K-*

	Test case with conflicts				Test case with $\delta = 94\%$			
	Quality	Similarity	Time	n_r	Quality	Similarity	Time	n_r
IAC+Anchors	0.576	**0.075**	5.262	49.7	0.309	**0.177**	3.051	60.1
IAC+Anchors+SI	**0.760**	0.024	4.868	49.83	0.393	0.108	2.972	60.1
PCK-Means	0.081	0.051	**3.639**	149.3	0.375	0.045	**2.834**	66.5
MPCK-Means	0.078	0.017	29.27	159.9	**0.406**	0.019	26.98	61.2

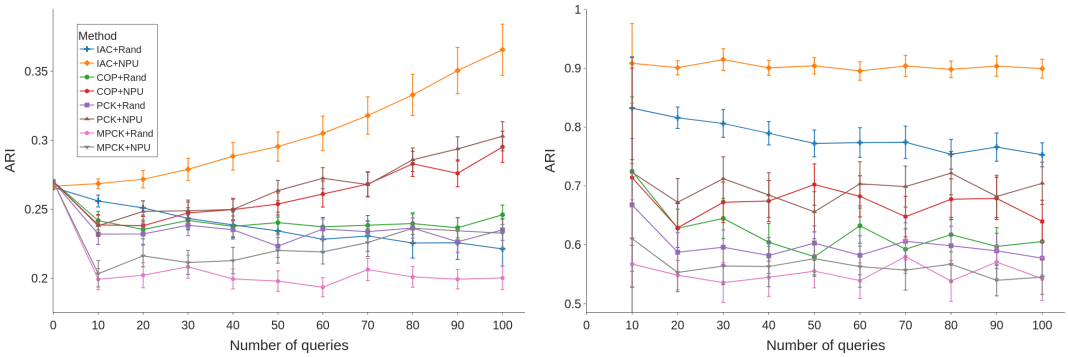
Table 2 Comparative study for clustering with pairwise constraints relaxation for the `mk2` dataset. Metrics are ARI with ground truth (Quality), ARI with unconstrained KMeans (Similarity), runtime and number of constraints relaxed in the solution (n_r).

Means (PCK-Means) [3] allows the violation of ML and CL constraints, and *Metric PCK-Means* (MPCK-Means) [6] combines PCK-Means with distance-metric learning [36]. As the alternatives only support pairwise constraints, we compare performance for problems that involve only ML/CL constraints. To that end, we consider two settings: in the first one, we generate 950 constraints at random based on the ground truth and add 50 conflicting constraints so that some constraints must be relaxed to solve the problem; in the second, we generate 1000 constraints at random without any explicit conflict constraint and set the satisfaction rate δ of IAC to the mean satisfaction rate of PCK-Means and MPCK-Means. We run two variants of IAC: IAC+Anchors ($\alpha = 20\%$, $\beta = 100\%$) and IAC+Anchors+SI ($\alpha = 20\%$, $\beta = 30\%$). Table 2 shows the performance of the different approaches for the `mk2` dataset, measured by ARI, runtime and number of constraints relaxed. Each value in the table is the average of 90 runs with different sets of constraints. As previously, for each run, we compute an initial KMeans partition and run our CP model only once for each set of constraints. Results show that in the presence of conflicts, our method violates fewer constraints than the other methods. Furthermore, the runtimes of IAC are comparable to those of PCK-Means. In contrast, MPCK-Means is significantly more expensive. In terms of clustering accuracy (measured by ARI), our approach clearly outperforms the compared to the alternatives. When imposing a satisfaction rate to the model, PCK-Means and MPCK-Means achieve comparable or better quality than IAC, but similarity stays low. However, IAC achieves again better performance in number of constraints relaxed.

4.2.4 Comparing IAC with alternatives in the incremental setting

In this section, we address research questions **RQ4** and **RQ5**, and conduct experiments to evaluate the interest and performance of our IAC framework for active constraint section context. NPU can be used in combination with any semi-supervised clustering algorithm, we use the same ones as in the previous section, including COPK-Means algorithm [33]. This leads to several combinations, and for each combination we perform 10 iterations of selection-modification loop. For each iteration, we use the current partition to select a batch of 10 queries with NPU and at random, get feedback from the user, and perform either a reclustering or MCM. In this experiment, queries correspond to pairwise constraints. In light of the results of Section 4.2.1, we choose the compromise configuration between quality and similarity ($\alpha = 20\%, \beta = 30\%$) for this experiment.

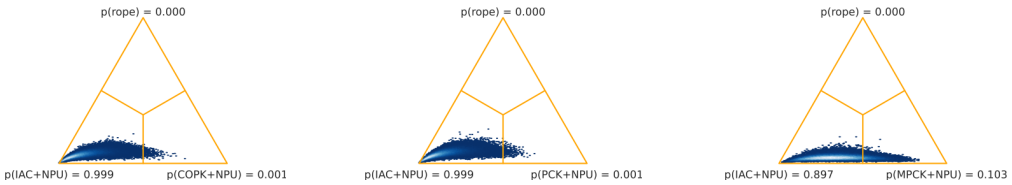
Clustering quality comparison. Figure 7 shows the evolution of the ARI scores over 10 iterations of incremental and active clustering modification for the `glass` dataset. IAC+NPU produces increasingly better clusterings as more iterations are given (cf. Fig. 7a), while keeping a high similarity throughout the iterations (cf. Fig. 7b). None of the competitors



(a) Comparison to the ground truth. **(b)** Comparison to the previous partition.

Figure 7 ARI scores of the partition at each iteration of selection-modification, compared to ground truth (left) or to the previous partition (right), for the `glass` dataset. All ARI scores are the mean and 95% confidence interval over 90 runs. Higher is better.

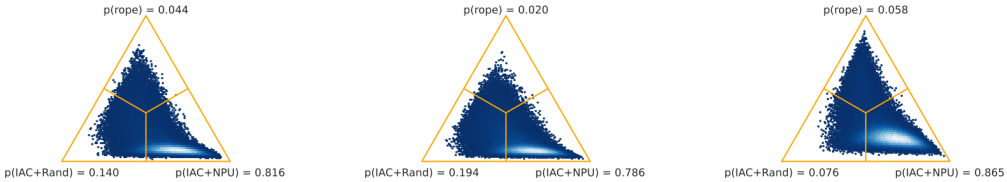
produces a clustering with a high ARI. Interestingly, both PCK-Means and COPK-Means with NPU are able to find good clusterings while MPCK-Means is not, even after a relatively large number of iterations. However, the behaviour of the competitors are more chaotic in terms of similarity. We observe similar results for the other datasets (due to lack of space, all other results are available via our link in the summary). These results also show that methods with a random selection of constraints produce typically worse results. We validate these observations by Bayesian comparison w.r.t. $AUBC_{quality}$ and $AUBC_{similarity}$ values for each metric. Fig. 8 show that IAC has a high probability to perform better than the alternatives on ARI ; we have similar results for AMI and FMI. Interestingly, using NPU leads to better similarity (see Fig. 9). This suggests that the use of an active constraint selection strategy brings another advantage besides improving the quality of the clustering. However, in an online context, runtime is particularly important as it requires user interaction and selecting the next query can be very costly, superseding the time taken for modification. For the biggest datasets (`Letters` and `MNIST`), only methods using random selection finish before timeout. We can conclude from these results that our model for minimal clustering modification is effectively a better way for active incremental constrained clustering compared to the naive approach to incrementality.



(a) IAC vs COPK-Means **(b)** IAC vs. PCK-Means **(c)** IAC vs. MPCK-Means

Figure 8 Bayesian comparisons with IAC using NPU w.r.t. $AUBC_{quality}$ values for ARI.

Runtime comparison. In Fig. 10, the runtime of modification (reclustering or MWCM) is shown for each dataset. The runtime of our model is comparable to those of the competitors, although it seems to have better scaling on the largest datasets. Empirically, IAC is an order of magnitude faster on `yeast`, `statlog` and `letters` datasets that have a large number of clusters. It is also noteworthy that MPCK-Means is much slower than other methods due to metric learning, yet this increase does not translate into better quality or similarity than



(a) ARI **(b)** AMI **(c)** FMI

Figure 9 Bayesian comparison of IAC with or without NPU w.r.t $AUBC_{similarity}$ for all metrics.

IAC. The limiting factor for the modification step of IAC is the computation of anchors and super-instances, whose scaling is worse than solving the COP in itself.

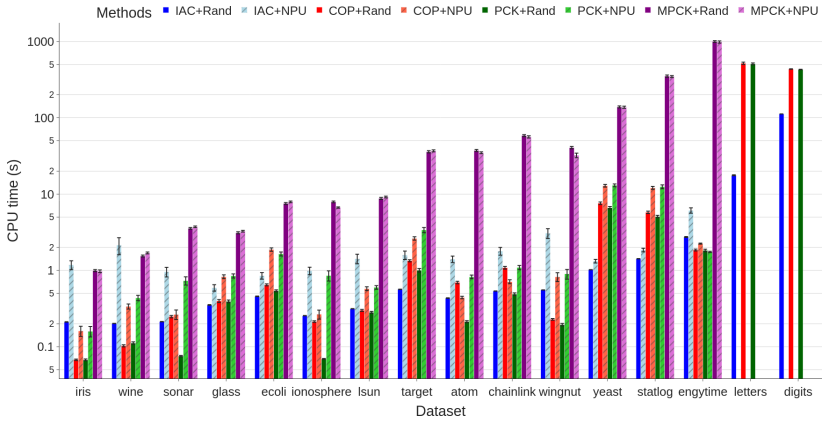


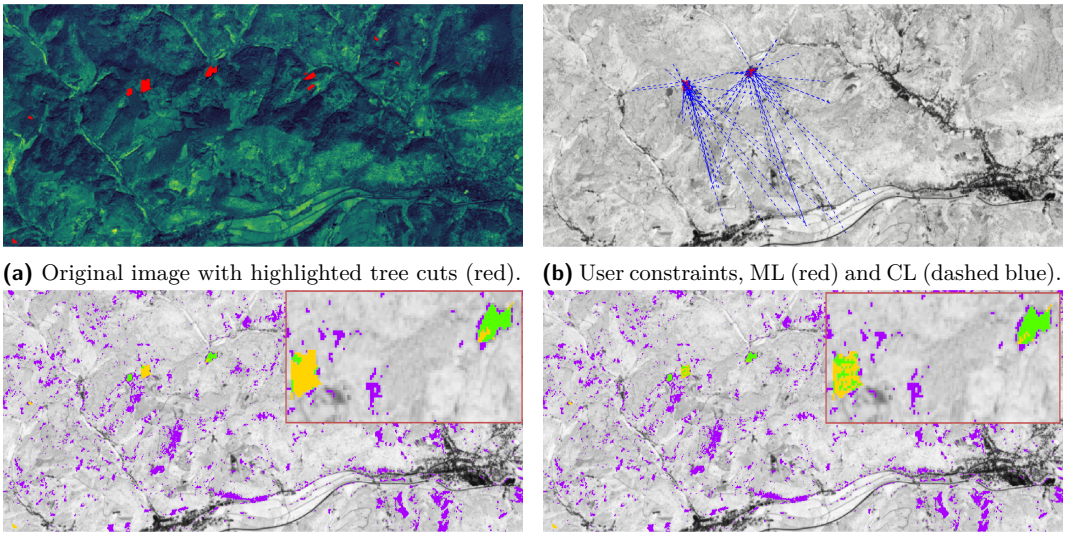
Figure 10 Comparing the runtime of the different methods in seconds (log scale). Methods using NPU are hatched.

4.3 Tree Cut Data

Introducing the data. Our case study for research question **RQ6** concerns the analysis of satellite image time series (SITS) composed of 11 images of dimensions 724×337 of a zone of the Vosges mountains in eastern France, taken irregularly on the span of 3 years from 2016 to 2018. Each pixel is associated to a series of NDVI (*Normalized Difference Vegetation Index*) values denoting the level of vegetation at each timestamp. At our disposition are labels that separate the SITS into three classes: vegetation, artificial structures, and tree cut zones. This last class has several properties: it was precisely labeled by domain experts, whereas the two other classes are more approximately defined. It is also a very small class as shown in Fig. 11a, containing only 639 instances (less than 0.3% of the data), which makes it hard to detect with unsupervised learning. Image-wise, 10 zones have been identified as places where trees have been cut within the time interval. Lastly, the evolution of these zones (a sharp decrease of NDVI value followed by a slow return to normal) is similar to that of field harvesting or grassland mowing, which complicates the problem further. In these conditions, expert intervention is paramount.

Problem definition. A set of 179 ML/CL constraints has been collected in [21] from domain experts, focused on the two largest tree cut zones (204 and 147 instances, i.e. more than half of tree cuts) as shown in Fig.11b. We define the problem as recovering these areas with binary constrained clustering. Following [22], we clustered the dataset with K-Means

and $K = 15$, only retaining the cluster covering the areas the most as the "positive" cluster of our problem. In Fig. 11c, this cluster is displayed in colors, while the "negative" cluster is composed of all pixels not colored. We then used this binary partition as input of IAC, and selected the unsatisfied user constraints in the partition to improve it. In our experiments, 79 constraints were unsatisfied. IAC was set to iterate until all constraints are satisfied. We set β to 100% as the large dataset size means that super-instance computation takes hours to complete, which is not compatible with a real life setting.



(a) Original image with highlighted tree cuts (red).

(b) User constraints, ML (red) and CL (dashed blue).

(c) Initial partition (with inset).

(d) Modified partition (with inset).

Figure 11 Some views of the use case ; pictures (c) and (d) show the "positive" cluster, before and after modification. Highlighted therein are the true positives (green), false negatives (yellow), and false positives (purple). Best viewed in colors.

Results. The modified clear cut cluster is displayed in Fig. 11d. The recovering of tree cut can be observed as the green zone of true positives enlarges in this figure : the left area progressed from 37 to 92 true positives, covering almost half the area. The right area gained 10 true positives. The modification was made within 22 seconds.

5 Conclusion

We have developed IAC, a framework for clustering modification that can be used in an incremental setting where an expert iteratively adds constraints, either manually or using an active method. A CP model for minimal weighted clustering modification ensures that the general cluster structure is preserved to maintain some continuity between iterations, as shown in experiments on reference datasets and on a real use case. It can also efficiently exploit an active query strategy to converge faster, and handle contradictory constraints that the user may give as input. The runtime of IAC is dependent of the constraint selection step, which requires further experiments with more active methods and/or to develop a new one suited for the incremental setting. It would also be interesting to explore the use of multiple CP models for modification, such as [20] for minimal modification with cluster-level constraints. Lastly, there remain open questions about the potential reuse of relaxed constraints at a later iteration : What constraints to choose? When to propose them to the user? The conception of a strategy answering these interrogations is worth considering.

References

1 Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discov.*, 13(3):365–395, 2006.

2 Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, pages 333–344. SIAM, 2004. doi:10.1137/1.9781611972740.31.

3 Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. Active Semi-Supervision for Pairwise Constrained Clustering. In *ICDM*, pages 333–344, 2004.

4 Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis, 2017.

5 Christian Bessière, Emmanuel Hébrard, Brahim Hnich, Zeynep Kiziltan, and Toby Walsh. Filtering Algorithms for the NValue Constraint. *Constraints*, 11(4):271–293, December 2006. doi:10.1007/s10601-006-9001-9.

6 Mikhail Bilenko, Sugato Basu, and Raymond J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, pages 11–18, 2004.

7 David Cohn, Rich Caruana, and Andrew Mccallum. Semi-Supervised Clustering with User Feedback. 4, 2001. doi:10.1201/9781584889977.ch2.

8 Giorgio Corani, Alessio Benavoli, Janez Demšar, Francesca Mangili, and Marco Zaffalon. Statistical comparison of classifiers through Bayesian hierarchical modelling. *Machine Learning*, 106(11):1817–1837, November 2017. doi:10.1007/s10994-017-5641-9.

9 Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017.

10 Ian Davidson and S. S. Ravi. Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results. In *Knowledge Discovery in Databases: PKDD 2005*, Lecture Notes in Computer Science, pages 59–70, 2005.

11 Ian Davidson, S. S. Ravi, and Martin Ester. Efficient incremental constrained clustering. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 240–249, 2007.

12 Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 94–105, 2010.

13 E. B. Fowlkes and C. L. Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983.

14 Marek Gagolewski. A Framework for Benchmarking Clustering Algorithms, 2022.

15 Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

16 Germán González-Almagro, Daniel Peralta, Eli De Poorter, José-Ramón Cano, and Salvador García. Semi-Supervised Constrained Clustering: An In-Depth Overview, Ranked Taxonomy and Future Research Directions, 2023.

17 Mathieu Guilbert, Christel Vrain, Thi-Bich-Hanh Dao, and Marcilio C. P. de Souto. Anchored Constrained Clustering Ensemble. In *International Joint Conference on Neural Networks*, IJCNN, 2022.

18 Tias Guns. Increasing modeling language convenience with a universal n-dimensional array, cppy as python-embedded example. In *Modref*, volume 19, 2019.

19 Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

20 Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain, and Ian Davidson. A framework for minimal clustering modification via constraint programming. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pages 1389–1395, 2017.

21 Baptiste Lafabregue, Pierre Gançarski, Jonathan Weber, and Germain Forestier. Incremental constrained clustering with application to remote sensing images time series. 2022.

22 Thomas Lampert, Baptiste Lafabregue, Thi-Bich-Hanh Dao, Nicolas Serrette, Christel Vrain, and Pierre Gancarski. Constrained Distance-Based Clustering for Satellite Image Time-Series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(11):4606–4621, 2019.

23 Eric Yi Liu, Zhaojun Zhang, and Wei Wang. Clustering with relative constraints. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 947–955, New York, NY, USA, August 2011. Association for Computing Machinery. doi:10.1145/2020408.2020564.

24 James MacQueen. Some Methods For Classification And Analysis Of Multivariate Observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability,*, volume 1, pages 281–297, 1967.

25 Pavan Kumar Mallapragada, Rong Jin, and Anil K. Jain. Active query selection for semi-supervised clustering. In *19th International Conference on Pattern Recognition*, pages 1–4, 2008.

26 Logan Adam Mitchell. *INCREMENT - Interactive Cluster Refinement*. PhD thesis, 2016.

27 Nguyen-Viet-Dung Nghiem, Christel Vrain, and Thi-Bich-Hanh Dao. Knowledge integration in deep clustering. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2022, Proceedings, Part I*, volume 13713 of *Lecture Notes in Computer Science*, pages 174–190. Springer, 2022. doi:10.1007/978-3-031-26387-3_11.

28 Nguyen-Viet-Dung Nghiem, Christel Vrain, Thi-Bich-Hanh Dao, and Ian Davidson. Constrained Clustering via Post-processing. In *Discovery Science*, Lecture Notes in Computer Science, pages 53–67, 2020.

29 Abdelkader Ouali, Samir Loudni, Yahia Lebbah, Patrice Boizumault, Albrecht Zimmermann, and Lakhdar Loukil. Efficiently finding conceptual clustering models with integer linear programming. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 647–654, 2016.

30 Alfred Ultsch and Jörn Lötsch. The Fundamental Clustering and Projection Suite (FCPS): A Dataset Collection to Test the Performance of Clustering and Data Projection Algorithms. *Data*, 5(1):13, 2020.

31 Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? 2009.

32 Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, page 1103–1110, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

33 Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 577–584, 2001.

34 Kiri L. Wagstaff. Value, Cost, and Sharing: Open Issues in Constrained Clustering. In *Knowledge Discovery in Inductive Databases*, pages 1–10, 2007.

35 Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 563–572, 2010.

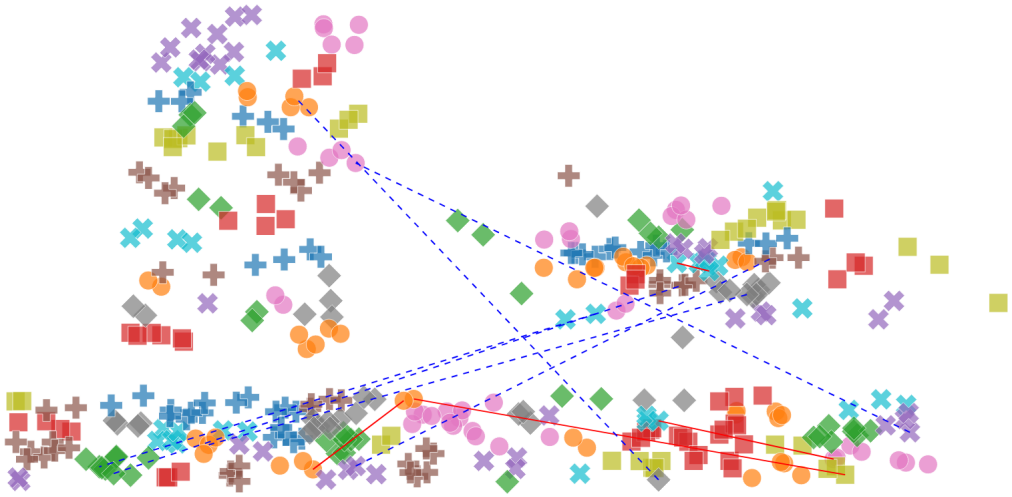
36 Eric Xing, Michael Jordan, Stuart J Russell, and Andrew Ng. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, volume 15. MIT Press, 2002.

37 Sicheng Xiong, Javad Azimi, and Xiaoli Z. Fern. Active Learning of Constraints for Semi-Supervised Clustering. *IEEE Trans. on Knowledge and Data Engineering*, 26(1):43–54, 2014.

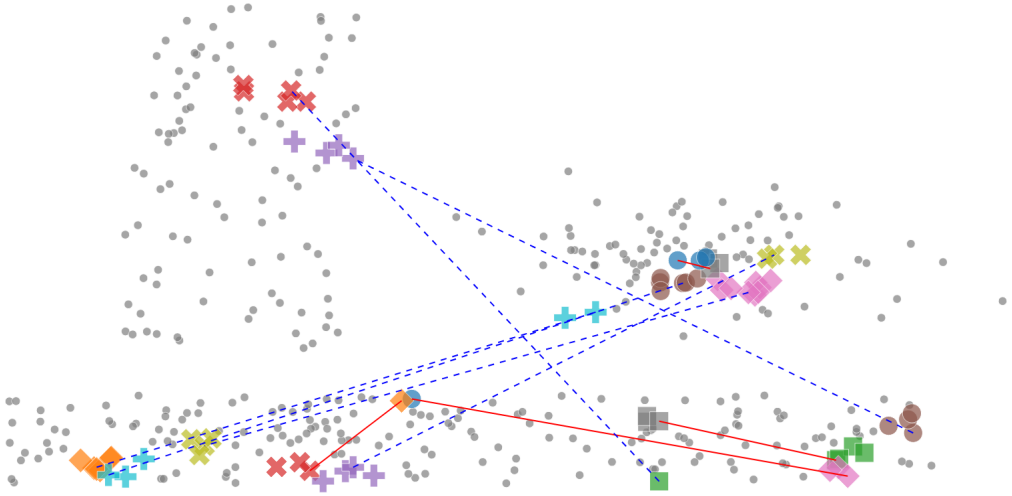
38 Xueying Zhan, Huan Liu, Qing Li, and Antoni B. Chan. A Comparative Survey: Benchmarking for Pool-based Active Learning. volume 5, pages 4679–4686, August 2021. ISSN: 1045-0823. URL: https://www.ijcai.org/proceedings/2021/634, doi:10.24963/ijcai.2021/634.

A Super-instances generation

This is an example case of generating super-instances and splitting to prevent emergent conflicts. In Fig. 12a, the clusters of the partition in Fig. 2 are divided into super-instances. However, some super-instances contain multiple constrained instances, e.g. the green one in the bottom left, which could lead to conflicts. In Fig. 12b, these super-instances have been splitted.



(a) Result of generating super-instances through *complete-link* hierarchical clustering on every cluster.



(b) Final super-instances after splitting. Greyed instances are unconstrained and not used in the CSP.

Figure 12 Exemple preprocessing for super-instance generation on `lsun` dataset. Instances sharing a color are represented by the same super-instance. ML constraints are in red, CL constraints in dashed blue.

B Incremental NPU

This is the variant of NPU we use in the selection step of IAC. In Algorithm 2, the main modifications are the removal of the reclustering step of the original NPU, and the output of the set of constraints obtained from the queries for the modification step. Algorithm 3 is unchanged and is shown to give a complete view of the algorithm.

Algorithm 2 Incremental NPU

Input : Dataset $\mathcal{D}$, partition $\mathcal{P}$, oracle
Output : constraint set $\mathcal{C}$

$\mathcal{C} \leftarrow \emptyset$; $l \leftarrow 1$; $\mathcal{N} \leftarrow N_1 \mid N_1 = \{random(\mathcal{D})\}$
$x^* \leftarrow MostInformative(\mathcal{D}, \mathcal{P}, \mathcal{N})$
for each $N_i \in \mathcal{N}$ in decreasing order of $P(x^* \in N_i)$ **do**
    Query x^* against any $x_i \in N_i$ to the oracle
    if (x^*, x_i, ML) **then**
        $\mathcal{C} \leftarrow (x^*, x_i, ML)$
        $N_i = N_i \cup x^*$
        break
    else
        $\mathcal{C} \leftarrow (x^*, x_i, CL)$
if no ML is returned **then**
    $l++$; $N_l = x^*$; $\mathcal{N} \leftarrow \mathcal{N} \cup N_l$
return $\mathcal{C}$

Algorithm 3 MostInformative

Input : Dataset $\mathcal{D}$, partition $\mathcal{P}$, set of neighborhoods $\mathcal{N}$
Output : most informative data point x^*

Learn a random forest classifier using $\mathcal{P}$ as labels
Compute the similarity matrix M s.t. $M[i,j]$ is the number of leaves where i and j are together normalized by the number of trees of the RF
for each $x \in \mathcal{U} = \mathcal{D} \setminus \mathcal{N}$ **do**
    for $i = 1$ to l **do**
        $$p(x \in N_i) = \frac{\frac{1}{|N_i|} \sum_{x_j \in N_i} M(x, x_j)}{\sum_{p=1}^{l} \frac{1}{|N_p|} \sum_{x_j \in N_p} M(x, x_j)}$$
    $H(\mathcal{N}|x) = -\sum_{i=1}^{l} p(x \in N_i) \log_2 p(x \in N_i)$
    $E(x) = \sum_{i=1}^{l} i * p(x \in N_i)$
return $\arg\max_{x \in \mathcal{U}} \frac{H(\mathcal{N}|x)}{E(x)}$

C Modifications and generalization

For each modified instance (or super-instance), we store its initial cluster membership and its new cluster membership. This allows the framework to keep a history of modifications and to easily retrieve the partition at any given iteration. Considering generalization, we also keep track of the composition of each constrained super-instance. When the COP produces a solution, Algorithm 4 transmits the modifications from the super-instance to the real data.

Algorithm 4 ApplyModifications

Input : dataset $\mathcal{X}$, super-instances S, modifications $\mathcal{M}$, partition $\mathcal{P}$
Output : modified partition $\mathcal{P}'$

$\mathcal{P}' \leftarrow \mathcal{P}$
for each $sp \in S$ **do**
    $points \leftarrow \{x \in \mathcal{X} \mid x \in sp\}$
    for each $p \in points$ **do**
        Update the membership of p in $\mathcal{P}'$ with the corresponding value in $\mathcal{M}$
return $\mathcal{P}'$

D Bayesian pairwise comparison of IAC configurations

	$AUBC_{quality}$			$AUBC_{similarity}$		
(α, β)	ARI	AMI	FMI	ARI	AMI	FMI
$(0, 10)$	9 (1)	10 (1)	9 (1)	0	0	0
$(0, 30)$	2 (0)	1 (0)	1 (0)	3 (1)	2 (0)	1 (1)
$(0, 50)$	0	0	0	6 (3)	6 (2)	4 (2)
$(0, 100)$	0	4 (1)	3	9 (9)	9 (9)	9 (9)
$(5, 10)$	10 (3)	10 (3)	10 (3)	1 (0)	1 (0)	1 (0)
$(5, 30)$	7 (3)	5 (3)	7 (3)	4 (2)	4 (1)	3 (1)
$(5, 50)$	1 (0)	2 (0)	1 (0)	6 (4)	6 (3)	4 (2)
$(5, 100)$	5 (2)	4 (0)	5 (1)	9 (9)	9 (9)	9 (9)
$(20, 10)$	10 (7)	10 (6)	10 (6)	1 (1)	2 (1)	1 (0)
$(20, 30)$	8 (3)	6 (3)	7 (3)	4 (2)	4 (2)	3 (1)
$(20, 50)$	4 (1)	3 (1)	4 (1)	6 (5)	6 (4)	6 (4)
$(20, 100)$	5 (2)	7 (3)	5 (1)	9 (9)	9 (9)	9 (9)

Table 3 Number of wins for each configuration (α, β) using a Bayesian hierarchical model. Values in parentheses indicate the number of cases where the probability that a configuration has a higher score is greater than 95%. Values of α and β are in percentage (%).