


Tackling Noise in Active Semi-Supervised Clustering

Jonas Soenen ^{1,2}[0000-0002-7289-6091],
Sebastijan Dumančić^{1,2}[0000-0003-0915-8034],
Toon Van Craenendonck³[0000-0002-2175-3293], and
Hendrik Blockeel^{1,2}[0000-0003-0378-3699]

¹ KU Leuven, Department of Computer Science

`firstname.lastname@cs.kuleuven.be`

² Leuven.AI

³ VITO NV, Unit Health

Abstract. Constraint-based clustering leverages user-provided constraints to produce a clustering that matches the user’s expectation. In active constraint-based clustering, the algorithm selects the most informative constraints to query in order to produce good clusterings with as few constraints as possible. A major challenge in constraint-based clustering is handling noise: the majority of existing approaches assume that the provided constraints are correct, while that might not be the case. In this paper, we propose a method to identify and correct noisy constraints in active constraint-based clustering. Our approach reasons probabilistically about the correctness of the user’s answers and asks additional constraints to corroborate or correct the suspicious answers. We demonstrate the method’s effectiveness by incorporating it into COBRAS, a state-of-the-art method for active constraint-based clustering. Compared to COBRAS and other active-constraint-based clustering algorithms, the resulting system produces better clusterings in the presence of noise.

Keywords: Active learning, Clustering, Semi-supervised learning

1 Introduction

Despite being one of the fundamental data mining tasks, clustering is an inherently subjective problem [8, 3]: different users often expect different clusterings of the same dataset. In contrast to a supervised learning setting, there is no way to select the clustering algorithm, its hyper-parameters and a similarity metric based on data only; the user has to try different settings until an informative clustering is found.

Semi-supervised clustering relies on a limited amount of supervision to guide the clustering process towards an informative clustering. Such supervision comes in the form of *pairwise constraints*: a *must-link constraint* between two instances indicates that the instances must be in the same cluster, while a *cannot-link constraint* indicates that the instances must be in different clusters (Figure 1).

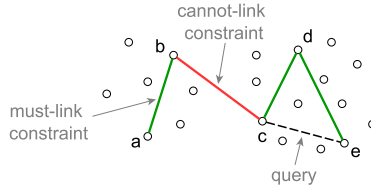


Fig. 1. In semi-supervised clustering, constraints between pairs of instances guide the clustering algorithm towards the desired solution. A must-link constraint indicates that two instances must be in the same cluster (e.g. a and b), a cannot-link constraint indicates that two instances must be in different clusters (e.g. b and c).

These pairwise constraints are often gathered in advance without knowing to which extent they are useful for the clustering process. Moreover, obtaining constraints usually requires human intervention and can be prohibitively expensive. In this case, it is beneficial to let the algorithm actively query the user: the algorithm presents the user with a specific pair of instances (a query) and the user answers with a 'must-link' or 'cannot-link' constraint. This way, an algorithm can pose the most informative queries first and produce a high quality clustering with a substantially smaller number of constraints. This setting is known as *active semi-supervised clustering*.

The prevalent assumption among existing semi-supervised clustering approaches is that all user-provided constraints are correct. However, this assumption is often violated as the user might only have a vague idea of the clustering structure of the data or might simply make mistakes while answering queries. It is thus likely that a user provides inaccurate or even contradicting constraints. These noisy constraints can have a detrimental impact on the quality of the produced clustering. This is especially true for active methods which try to minimize the number of queries by asking only the most informative ones – wrongly answering such a query will have a significant impact on the final clustering.

In this work, we tackle the problem of handling noise in *active semi-supervised clustering*. The core idea behind our approach is to *intentionally introduce redundancy in the constraint set, by means of querying constraints that would form cycles in the available set*. Whether or not the resulting cycles are consistent will then be used to reason about noise in the constraints. For instance, answering a redundant query between instances c and e in Figure 1 would make a cycle $cdec$. If the user answers this query with a cannot-link constraint, the new cycle is inconsistent: the newly provided constraint states that instances c and e should not be in the same cluster, while the available must-links ($c - b$ and $b - e$) state that they should. This is contradictory information implying the existence of a noisy constraint. On the other hand, if the user answers with a must-link constraint, the new constraint forms a consistent cycle with the must-link constraints (c, d) and (d, e) and, therefore, increases our confidence in the correctness of the constraints in this cycle. To reason about noise in a principled way, we use a probabilistic model that entails the reasoning illustrated above.

To show the effectiveness of our method, we integrate it into COBRAS [13], a state-of-the-art active semi-supervised clustering algorithm. COBRAS will select queries that are informative for clustering; our approach will complement these constraints by selecting queries that help to detect and correct the noisy constraints. We focus on COBRAS because it is one of the approaches most sensitive to noise. In our experiments, we show that with our approach COBRAS becomes significantly more robust to noise.

2 Related work

While there is a substantial amount of research on constraint-based clustering, including active learning approaches, almost none of it explicitly deals with noisy constraints.

Most existing methods for semi-supervised clustering take pairwise constraints into account by optimising a loss function that includes a penalty for each violated constraint (e.g., PCK-means [1], MPCK-means [2], LCVQE [10] and COSC [11]). Thus, the constraints are interpreted as soft constraints. This makes these methods robust to noise in the constraints, but it does not really *counter* noise. The approach reflects the viewpoint that it may not be possible to satisfy all constraints, but without distinguishing two different reasons for this: because the constraint is simply wrong (noisy) or because a good solution that satisfies it could not be found. Ideally, when a constraint is deemed likely to be noisy, the penalty for violating it should be lower. Moreover, whether the constraint is likely noisy should not be assessed based on how well the constraint fits the inductive bias of the clustering system, as the purpose of these constraints is exactly to change that bias.

A system that does actively try to reduce the effect of noisy constraints is COP-RF [16]. Before the clustering process starts, COP-RF filters out 50% of the constraints that are the least similar to all other constraints of the same type. Besides this, the approach by Yang et al. [15] formulates a maximum entropy model that can learn from noisy pairwise constraints. This model can be used to learn a kernel matrix from the noisy pairwise constraints, which is then used to cluster the dataset using spectral clustering [12].

None of the above approaches are active learners: they consider the set of constraints as fixed, rather than actively looking for new constraints. In an active learning context, dealing with noise is even more important: to minimize the number of queries (instances pairs to be labeled by the user), active learners try to eliminate as much redundancy as possible in the constraints, when in fact such redundancy is crucial in a noisy context. In the context of active constraint-based clustering, the only noise-handling approach we are aware of is the work by Mazumbar and Saha [9]. They propose a method that theoretically guarantees that the correct ground truth is recovered with high probability. However, this method requires multiple constraints per instance in the dataset, so the number of queries becomes orders of magnitude larger than for typical active constraint-

based clustering systems. This renders the method practically infeasible for large datasets.

What we propose in this paper, is a method that tries to identify noisy constraints in a way that is independent of the clustering system, and thus could in principle be combined with any of the existing constraint-based clustering methods (whether active or not, though non-active methods become active if this procedure is included). The method employs a small number of additional queries to that aim.

3 Reasoning about noisy constraints

In active semi-supervised clustering, the algorithm actively queries the user for the constraints between specific pairs of instances. The constraints obtained from the user are leveraged during clustering to produce high quality results. Usually, active semi-supervised clustering algorithms avoid asking redundant constraints. In a noiseless world, these redundant constraints do not give the algorithm any additional information. However, when there is noise, redundant constraints are crucial to reason about noisy constraints in a manner independent from clustering bias.

Our approach expects a set of constraints and will ask additional redundant queries to reason about the noise in these constraints. The end result is a corrected version of the given constraint set that is likely to be free of noisy constraints. This corrected set of constraints can then be used for clustering. Before we present the details of our approach, we first introduce the necessary terminology.

3.1 Terminology and background

We will denote a must-link constraint between instances x and y as $ml(x, y)$ and a cannot-link constraint as $cl(x, y)$. Let S be the set of instance pairs for which the user has provided a constraint. Let \mathcal{U} be the set of all constraints that are obtained from the user, such that \mathcal{U} contains $ml(x, y)$ or $cl(x, y)$ for each $(x, y) \in S$. The set \mathcal{G} stands for the set of ground truth constraints about the instance pairs in S . The constraints in \mathcal{G} are unknown; our goal is to recover these constraints from \mathcal{U} . If a new query is posed, the query itself is added to S , the user's answer is added to \mathcal{U} and \mathcal{G} will contain the corresponding ground truth constraint. For notational convenience, $\mathcal{C}(x, y)$ will denote the constraint type (ml or cl) of the constraint in the constraint set \mathcal{C} between the instances x and y . A **noisy constraint** is any user-provided constraint that differs from the corresponding ground truth constraint, i.e. $\mathcal{U}(x, y) \neq \mathcal{G}(x, y)$.

For example, suppose the user has provided the constraints in Figure 1; then $\mathcal{U} = \{ml(a, b), cl(b, c), ml(c, d), ml(d, e)\}$. However, the constraint between d and e is actually a cannot-link constraint in the ground truth constraint set, thus $\mathcal{G} = \{ml(a, b), cl(b, c), ml(c, d), cl(d, e)\}$. In this case, the user-provided constraint $ml(d, e)$ is noisy (because $\mathcal{U}(d, e) = ml$ and $\mathcal{G}(d, e) = cl$).

Given a constraint set, we can deduce additional constraints that are implied by the constraints in the given set:

$$\begin{aligned} ml(x, y) \wedge ml(y, z) &\Rightarrow ml(x, z) && \text{(must-link transitivity)} \\ ml(x, y) \wedge cl(y, z) &\Rightarrow cl(x, z) && \text{(cannot-link entailment)} \end{aligned}$$

We call a constraint set **consistent** if there is no instance pair (x, y) for which both a must-link and a cannot-link constraint are implied. In other words, a consistent constraint set does not contain any contradicting constraints.

If we represent a constraint set as a graph where nodes are instances and edges indicate the corresponding constraints (as in Figure 1), then the following holds:

Proposition 1. *A constraint set is inconsistent if and only if its graph contains a cycle with exactly one cannot-link edge. We call such a cycle an **inconsistent cycle**.*

Proof. The *if* part in the above claim is trivial. To explain the *only if* part, note that the propagation rules can only derive both ml and cl for the same instance pair (x, y) if there are at least two paths between x and y , one of which is ml-only (hence the ml derivation) and the other contains exactly one cl (two cls would break the derivation chain). These two paths form a cycle with exactly one cannot-link edge.

If the user-provided constraint set \mathcal{U} is inconsistent, there is no clustering where all constraints from \mathcal{U} are satisfied and thus there must be at least one constraint in \mathcal{U} that is noisy. Moreover, there is at least one noisy constraint in each inconsistent cycle in \mathcal{U} .

3.2 Overview

The goal of our work is to find a constraint set \mathcal{C} , derived from the user-provided constraints \mathcal{U} , such that we are reasonably confident that \mathcal{C} corresponds to ground truth \mathcal{G} . To reason about the correctness of a constraint set \mathcal{C} , we define a probabilistic model that quantifies $P(\mathcal{G} = \mathcal{C} \mid \mathcal{U})$, i.e. the probability that the ground truth is equal to a set of constraints \mathcal{C} given a set of user constraints \mathcal{U} . Our goal is to find a constraint set \mathcal{C} such that $P(\mathcal{G} = \mathcal{C} \mid \mathcal{U}) \geq \alpha$ with α a predefined threshold. In the following, we call $P(\mathcal{G} = \mathcal{C} \mid \mathcal{U})$ the **confidence in \mathcal{C}** .

When the user-provided constraint set \mathcal{U} is consistent, the most likely constraint set \mathcal{C} is equal to \mathcal{U} . When \mathcal{U} is inconsistent, there is noise in \mathcal{U} . In this case, the most likely constraint set \mathcal{C} is the consistent constraint set that differs from \mathcal{U} in as few constraints as possible.¹

However, \mathcal{C} does not automatically satisfy the confidence threshold α . There might not be enough redundancy in the constraint set to support \mathcal{C} . Therefore, we

¹there might be multiple most likely constraint sets with equal likelihood

Algorithm 1: Verification procedure

```

Function verify( $\mathcal{U}$ ):
   $\mathcal{C} \leftarrow \text{most\_likely}(\mathcal{U})$ 
  while confidence( $\mathcal{C}, \mathcal{U}$ ) <  $\alpha$  do
     $\text{new\_con} \leftarrow \text{ask\_informative\_redundant\_query}(\mathcal{U})$ 
     $\mathcal{U} \leftarrow \mathcal{U} \cup \{\text{new\_con}\}$ 
     $\mathcal{C} \leftarrow \text{most\_likely}(\mathcal{U})$ 
  return  $\mathcal{C}$ 

```

select additional redundant queries that complete cycles among the constraints in \mathcal{U} until the confidence of the most likely constraint set \mathcal{C} reaches the threshold. Adding the new constraint to \mathcal{U} changes the most-likely constraint set \mathcal{C} . The confidence of the new \mathcal{C} might be higher than that of the previous \mathcal{C} , e.g. the new constraint is consistent with the previous \mathcal{C} , or lower, e.g. a new inconsistent cycle is detected. By carefully selecting which queries we ask the user, we aim to reach the confidence threshold with as few queries as possible. Pseudocode for this procedure can be found in Algorithm 1.

In the remainder of this section, we present the probabilistic model used to calculate the confidence of a constraint set. However, because evaluating this model entails summing over all possible constraint sets, exact computation of the confidence is computationally infeasible. Therefore, we introduce a simple approximation of the actual confidence by focusing on a subset of all possible constraint sets and provide a procedure to efficiently enumerate this subset. Lastly, the way we select informative redundant queries is explained.

3.3 Defining the confidence of a constraint set

We develop a probabilistic approach where the value of all user constraints $\mathcal{U}(x, y)$ and the corresponding ground truth constraints $\mathcal{G}(x, y)$ are considered random variables. We use a Bayesian approach that computes $P(\mathcal{G} = \mathcal{C} \mid \mathcal{U})$ using the Bayes rule:

$$P(\mathcal{G} = \mathcal{C} \mid \mathcal{U}) = \frac{P(\mathcal{U} \mid \mathcal{G} = \mathcal{C})P(\mathcal{G} = \mathcal{C})}{\sum_{\mathcal{C}' \in \mathbb{C}} P(\mathcal{U} \mid \mathcal{G} = \mathcal{C}')P(\mathcal{G} = \mathcal{C}')} \quad (1)$$

Where \mathbb{C} is the set of all possible constraint sets about the pairs in S

For the prior $P(\mathcal{G} = \mathcal{C})$, we assume a uniform distribution over all *consistent* constraint sets \mathcal{C} . This implies that all inconsistent constraint sets have a probability of 0, thus we can sum over all consistent constraint sets instead of all constraint sets. As $P(\mathcal{G} = \mathcal{C})$ is equal for every consistent constraint set, it can be ignored (assuming \mathcal{C} is consistent, otherwise $P(\mathcal{G} = \mathcal{C} \mid \mathcal{U})$ is equal to 0).

We assume i.i.d. noise: there is a fixed probability ν that the user answers a query incorrectly. Consequently, for each pair (x, y) and label c , $P(\mathcal{U}(x, y) \neq c \mid \mathcal{C}(x, y) = c) = \nu$. With this assumption in place, the likelihood $P(\mathcal{U} \mid \mathcal{G} = \mathcal{C})$ can

be written in function of n , the number of constraints in \mathcal{U} , and d , the number of constraints where \mathcal{U} and \mathcal{C} disagree.

$$P(\mathcal{U} \mid \mathcal{G} = \mathcal{C}) = \nu^d (1 - \nu)^{n-d} \quad (2)$$

Note that the assumptions behind the probabilistic model are likely to be violated in practice; for instance, some pairs may be easier to correctly label than others. Despite being violated in practical cases, the model allows us to set formal foundations for tackling noise.

3.4 Approximating the confidence of a constraint set

To calculate the confidence of a certain constraint set \mathcal{C} , we have to sum over all consistent constraint sets \mathcal{C}' (see denominator of Equation 1). Naively enumerating all consistent constraint sets is not practically feasible as the number of consistent constraint sets is, in the worst case, exponential in the number of constraints. Therefore, we introduce an approximation that relies on a subset of the consistent constraint sets that contribute the most to the confidence and provide a procedure that finds these constraint sets efficiently.

High-likelihood constraint sets. Given our noise model, the contribution of a consistent constraint set \mathcal{C}' to the denominator of the confidence is equal to $\nu^d (1 - \nu)^{n-d}$ with n the total number of constraints in \mathcal{U} and d the number of pairs for which \mathcal{U} and \mathcal{C} disagree (Equation 1, 2). For small ν , terms with high d contribute little to the sum. We therefore only sum over consistent constraint sets \mathcal{C}' with $d \leq k$, for some parameter k called the **approximation order**. Higher values for k result in more accurate approximations, but increase the execution time of the algorithm considerably, as the size of the search space is exponential in k .

This approximation is accurate if it captures most of the probability mass, that is, when there is a high probability that the number of noisy constraint in \mathcal{U} is smaller than the approximation order k . From our noise assumption follows that the number of noisy constraints in \mathcal{U} follows a binomial distribution distributed with $|\mathcal{U}|$ trials and success probability ν .

$$P(\#\text{noisy constraint in } \mathcal{U} \leq k) = \sum_{d=0}^k \binom{|\mathcal{U}|}{d} \nu^d (1 - \nu)^{|\mathcal{U}|-d} \quad (3)$$

If k is fixed, $P(\#\text{noisy constraints in } \mathcal{U} \leq k)$ increases when the size of \mathcal{U} decreases or the noise probability ν decreases. Thus our approximation is most accurate for small constraint sets and low noise probabilities. If the approximation order k is too small relative to $|\mathcal{U}|$ and ν , the approximation overestimates the confidence and our approach will ask less redundant constraints than actually necessary to reach the desired confidence level.

To make our approximation more accurate for bigger constraint sets we employ two tricks: we use a dynamically growing *effective approximation order* and verify the user-provided constraints in *batches*.

When using a *static* approximation order k , the approximation becomes less accurate as the number of identified noisy constraints increases. After identifying k noisy constraints, all consistent constraint sets with more than k noisy constraints are not considered in the approximation of the confidence. This results in a highly overestimated confidence and causes the procedure to never identify more than k noisy constraints.

In order to resolve this, we use a *dynamically growing effective approximation order* k' instead of the static approximation order k . The effective approximation order k' is determined as the sum of the number of noisy constraints in the current most likely constraint set and the approximation order k . The effective approximation order thus grows each time a noisy constraint is detected in the user-provided constraints \mathcal{U} . This ensures that the approximation can always reason about k additional noisy constraints on top of the noisy constraints that have already been identified.

Additionally, the user-provided constraints can be verified in batches to enhance the accuracy of the approximation; instead of verifying all user-provided constraints \mathcal{U} in one single go, \mathcal{U} is divided in several smaller batches. One after the other, each of these batches is added to a constraint set \mathcal{U}' . After adding a batch to \mathcal{U}' , the full set is verified. Because the preexisting constraints in \mathcal{U}' have been verified in the previous iteration, it is unlikely that these preexisting constraints contain undetected noisy constraints. The number of noisy constraints in the newly added batch \mathcal{B} follows the same distribution as in Equation 3 but proportional to the $|\mathcal{B}|$ instead of $|\mathcal{U}|$, with $|\mathcal{B}| < |\mathcal{U}|$. Therefore, the accuracy of the approximation will mostly depend on the size of the newly added batch.

For several active semi-supervised clustering approaches it is actually natural to verify the constraints in batches. NPU [14] and COBRAS [13] produce a new intermediate clustering every couple of queries. Therefore, it makes sense to verify the constraints right before each intermediate clustering is produced as to make sure that none of these intermediate clusterings is based on a noisy constraint.

Enumerating consistent constraint sets. Algorithm 2 outlines a simple yet efficient recursive procedure to enumerate all consistent constraint sets up to the effective approximation order k' . These consistent constraints sets are used to approximate the confidence of a constraint set, determine the most likely constraint set and select redundant queries. In each recursive call, one constraint in a constraint set \mathcal{A} is flipped until it reaches the desired effective approximation order. If the constraint set \mathcal{A} is consistent, each of the constraints in \mathcal{A} is considered as a candidate to be flipped next. If the constraint set \mathcal{A} contains an inconsistent cycle, the only way this constraint set can become consistent is to flip one of the constraints involved in the inconsistent cycle. Therefore, only the constraints involved in the inconsistent cycle have to be considered as candidates to be flipped next. Every consistent constraint set that is encountered during this procedure is stored. To ensure that every consistent constraint set is only encountered once, a set of constraints *flipped* is maintained that keeps track

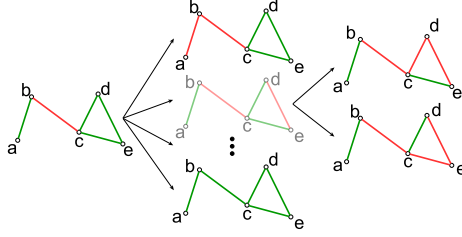


Fig. 2. An example of an incomplete search tree to find consistent constraint sets. The starting constraint set is consistent, hence every constraint is a candidate for flipping. The second constraint set at depth 1 is inconsistent, only the constraints between instance pairs (c, d) and (c, e) need to be considered as candidates ((d, e) was flipped in a previous recursive call).

Algorithm 2: Procedure to enumerate consistent assignments

```

Function solve( $\mathcal{A}$ ,  $flipped$ ):
  if  $unsat\_user\_constraints(\mathcal{A}) > k'$  then
    return  $\emptyset$ 
   $solutions \leftarrow \emptyset$ 
  if  $\mathcal{A}$  is consistent then
     $solutions \leftarrow solutions \cup \{\mathcal{A}\}$ 
     $candidates \leftarrow \mathcal{U} \setminus flipped$ 
  else
     $cycle \leftarrow find\_inconsistent\_cycle(\mathcal{A})$ 
     $candidates \leftarrow cycle \setminus flipped$ 
  for  $con$  in  $candidates$  do
     $flipped \leftarrow flipped \cup \{con\}$ 
     $\mathcal{A}' \leftarrow \mathcal{A} \setminus con \cup \{con.flip()\}$ 
     $solutions \leftarrow solutions \cup solve(\mathcal{A}', flipped)$ 
  return  $solutions$ 

```

of all the constraints that have already been flipped and should not be flipped again. Figure 2 shows part of the search tree explored by Algorithm 2 for an example scenario.

Finding inconsistent cycles. To be able to use Algorithm 2, we need to detect when a constraint set is inconsistent and, if so, find an inconsistent cycle in the constraint set.

To check if a constraint set is inconsistent, we use a procedure similar to the feasibility test with must-link and cannot-link constraints of Davidson and Ravi [4]. The procedure maintains a set of must-link components, groups of instances that are interconnected by must-link constraints, and checks whether there are two instances within the same must-link component that are connected

by a cannot-link constraint. If such a pair of instances is found, this cannot-link forms an inconsistent cycle with some must-links from this must-link component and thus the constraint set is inconsistent. To reconstruct the inconsistent cycle, we start from the two instances connected by the cannot-link and search for a path between them that only contains must-link constraints.

3.5 Selecting redundant queries

Now that we have all pieces to calculate the confidence of the most likely constraint set \mathcal{C} , we still have to select an informative query that will help us to reach the confidence threshold.

We distinguish two different scenarios in which a redundant query needs to be selected.

1. If there is only one most likely constraint set \mathcal{C} , we aim to increase the confidence of this constraint set. Therefore, we query the pair (x, y) for which the total likelihood of consistent constraint sets that imply a different constraint value for (x, y) than \mathcal{C} is maximal. As before, we only use the consistent constraint sets up to the current effective approximation order k' .
2. If there are multiple most likely constraint sets, we query a pair that will reduce the amount of most likely constraint sets. As such, we aim to query a pair (x, y) for which the amount of most likely constraint sets that imply must-link is approximately equal to the amount of most likely constraint sets that imply cannot-link. The answer to this query will thus agree with approximately half of these constraint sets and disagree with the other half. Adding the answer of this query to \mathcal{U} causes the confidence of the agreeing constraint sets to increase and the confidence of the disagreeing constraint sets to decrease. This gives rise to a binary-search-like procedure which is repeated until only one most likely constraint set remains.

It is possible that, at some point, no interesting queries are left (e.g. all possible instance pair within a single must-link component have been queried) and the confidence of the most likely constraint set is still lower than α . In this case, the most likely constraint set (or one of the most likely constraint sets if there are multiple) is assumed to be correct.

4 Integration in COBRAS

COBRAS [13] is a state-of-the-art active semi-supervised clustering algorithm. Its strengths are that it is time-efficient, query-efficient, and that it provides intermediate results. Its query-efficiency, however, naturally makes it more sensitive to noise. Moreover, COBRAS has no noise handling mechanism, as the algorithm aims to satisfy all of the user-provided constraints. Therefore, it is a good candidate for testing the effectiveness of our noise correcting approach.

COBRAS is an iterative algorithm where each iteration consists of a splitting phase and a merging phase. In its splitting phase, COBRAS identifies so-called

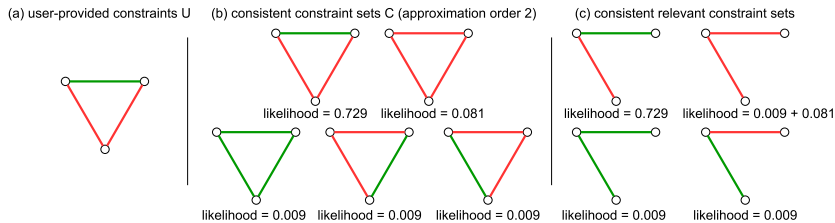


Fig. 3. Illustration of how the consistent constraints sets are marginalized over the irrelevant constraints (dotted lines) to only retain the relevant constraints (full lines). (a) shows the given user-provided constraints (b) shows all consistent constraint sets and their likelihood derived from the user-provided constraints with approximation order 2 (c) shows the marginalized constraint sets with their likelihood.

super-instances, which are groups of instances that are assumed to belong to the same cluster. During splitting, a small amount of queries are used to estimate the number of super-instances that are needed. The new super-instances (groups of instances) are then merged into clusters by querying constraints between different super-instances. Two super-instances are merged into the same cluster if the constraint between them is a must-link; two super-instances are not merged together if the constraint between them is a cannot-link. It is after this phase that our noise-correction approach is deployed.

After each merging phase, our approach tries to recover the correct constraints \mathcal{C} from the user-provided constraints \mathcal{U} . If no noisy constraints are detected (i.e. $\mathcal{C} = \mathcal{U}$), COBRAS can safely continue with the next iteration. Otherwise, the current clustering is based on at least one noisy constraint and has to be corrected. Therefore, the merging phase of COBRAS is repeated with the corrected constraint set \mathcal{C} instead of \mathcal{U} . We refer to the thus obtained noise-robust version of COBRAS as nCOBRAS.

Reasoning on a subset of relevant constraints In COBRAS, only a subset of all the constraints provided by the user is used to construct the next intermediate clustering (i.e. only the constraints between the current super-instances are used). Thus, it makes sense to only verify the constraints from this relevant subset and not waste queries on obtaining the correct values of the irrelevant constraints. However, the irrelevant constraints might still complete cycles with relevant constraints and thus might still provide useful information. Therefore, to calculate the confidence of a subset of relevant constraints \mathcal{C}_{rel} , we reason on the full set of constraints and marginalize over the irrelevant constraints \mathcal{C}_{irrel} .

$$P(\mathcal{G}_{rel} = \mathcal{C}_{rel} | \mathcal{U}) = \sum_{\mathcal{C}_{irrel} \in \mathcal{C}_{irrel}} P(\mathcal{G} = \mathcal{C}_{rel} \cup \mathcal{C}_{irrel} | \mathcal{U}) \quad (4)$$

Where \mathcal{C}_{irrel} is the set of all possible constraint sets involving the same pairs as the irrelevant constraints

To ensure that the redundant query selection procedure only selects queries that are informative to verify the relevant constraints, all consistent constraint sets enumerated by Algorithm 2 are marginalized over the irrelevant constraints. The active query selection procedure is then executed on the marginalized constraint sets that only contain the relevant constraints (Figure 3).

5 Experiments

We compare the clustering performance of several active semi-supervised clustering approaches in the presence of varying amounts of noise. We are especially interested to investigate **(Q1)** whether nCOBRAS is indeed more noise robust than COBRAS and **(Q2)** how nCOBRAS performs compared to existing active constraint-based clustering approaches.

5.1 Algorithms

We compare nCOBRAS² to the following active semi-supervised clustering algorithms:

- COBRAS [13], the original COBRAS algorithm with no noise handling mechanism. We use the code provided by the authors.³
- NPU [14] is an active query selection scheme that can be used with any semi-supervised clustering algorithm, we use it with:
 - MPCK-means [2], a constrained modification of k-means that uses a modified objective function and allows constraint violation. We use the implementation in the wekaUT package.⁴
 - COSC [11], a constrained extension of spectral clustering that optimizes a modified objective function. We use the fast implementation of the code provided by the authors.^{5,6}

NPU with MPCK-means (NPU_MPCK) and NPU with COSC (NPU_Cosc) require the desired number of clusters K before clustering. In our experiments the true value of K (as indicated by the class labels) is given to these algorithms. This gives NPU-MPCK and NPU-Cosc an advantage over COBRAS and nCOBRAS, which do not require the number of clusters.

nCOBRAS needs three hyperparameters to be set: an estimated noise probability ν , the confidence threshold α and the approximation order k . In our experiments, we provide nCOBRAS with the true noise probability for ν , except when there is no noise: there, we set ν to 0.05 instead of 0, so that we can get an idea of the cost of noise-handling when it is in fact unnecessary. The value of the confidence threshold α is set to 0.95 and an approximation order of 3 is used.

²https://github.com/magicalJohn/noise_robust_cobras

³<https://dtai.cs.kuleuven.be/software/cobras/>

⁴<https://www.cs.utexas.edu/users/ml/risc/code/>

⁵<https://www.ml.uni-saarland.de/code/cosc/cosc.htm>

⁶In some cases the code provided by the authors fails to produce a clustering; if this happens, we use the clustering from the previous NPU iteration instead

5.2 Experimental methodology

We perform 3 times 10-fold cross-validation and report the average clustering quality. The algorithms cluster the full dataset, but only query constraints between instances that are both in the training set. The clustering quality is determined by calculating the adjusted rand index (ARI) [7] on the instances in the test set. The ARI measures the similarity between the produced clustering and the ground truth clustering. A random clustering has an expected ARI of 0, while a clustering identical to the ground truth clustering has an ARI of 1.

To guarantee that COBRAS and nCOBRAS do not query any test instances during clustering, we ensure that all super-instance representatives are in the training set. For NPU, the selection of the most informative instance x^* is modified to exclude points not in the training set.

We also report the average aligned rank (AAR) [6, 5], which ranks the overall performance of each of the algorithms. The lower the AAR the better the algorithm performs comparatively to the other algorithms. To calculate the average aligned rank, for each dataset d and each algorithm a , compute the difference between the average ARI of algorithm a on dataset d and the average ARI achieved by all algorithms on dataset d . The resulting differences are ranked from high to low. The AAR for an algorithm is the average of the positions of its entries in the sorted list.

The noisy constraints are generated according to the noise assumption presented in Section 3: for an experiment with $x\%$ of noise, each of the queries has a probability of $x\%$ that the query is answered incorrectly.

5.3 Datasets

For our experiments, we use 21 clustering tasks defined over 17 datasets as in Van Craenendonck et al. [13]. These datasets include 15 UCI classification datasets: iris, wine, dermatology, hepatitis, glass, ionosphere, optdigits389, ecoli, breast-cancer-wisconsin, segmentation, column 2C, parkinsons, spambase, sonar and yeast. For these datasets, the class labels indicate the target clustering. Additionally, we consider 4 clusterings of the CMU faces dataset and cluster 2 different subsets of the 20 newsgroup text dataset. For details about the preprocessing and the clustering tasks, we refer to Van Craenendonck et al.[13].

5.4 Results

We ran each of the algorithms for 200 queries with varying amounts of noise (0%, 5% and 10%). For each noise value, we calculate the average ARI and the average aligned rank of each algorithm over all clustering tasks. The results are shown in Figure 4 and Figure 5.

(Q1) The results (Figure 4) confirm that nCOBRAS is less sensitive to noise than COBRAS. The average clustering quality of COBRAS significantly decreases when the amount of noise increases: with 10 % of noise, its ARI decreases

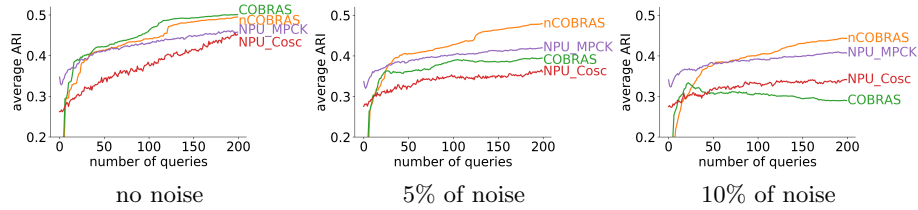


Fig. 4. Average ARI comparison of the different algorithms in the presence of varying amounts of noise (higher is better).

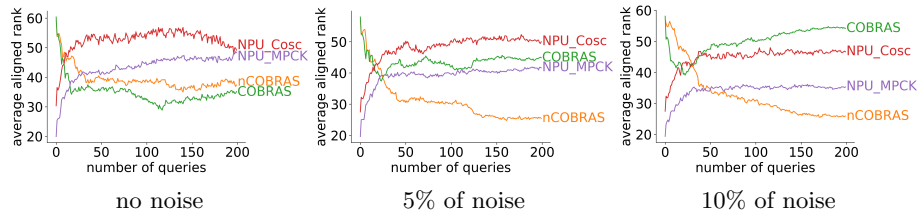


Fig. 5. Average aligned rank of the different algorithms in the presence of varying amounts of noise (lower is better).

by half. In contrast, the average clustering quality of nCOBRAS is relatively stable over all amounts of noise. Noise causes nCOBRAS’ improvement to slightly slow down: with more noise, nCOBRAS has to ask more redundant queries to verify the constraints. Moreover, with no noise nCOBRAS’ performance is very close to that of COBRAS which does not reason about the correctness of the constraints.

(Q2) The results also indicate that nCOBRAS outperforms both competitors, NPU-Cosc and NPU-MPCK, for all levels of noise. NPU-MPCK outperforms nCOBRAS for small amounts of constraints; this is due to the combined unsupervised and semi-supervised objectives of MPCK-means which allows it to get to a better initial clustering than nCOBRAS. Both NPU-MPCK and NPU-Cosc are less sensitive to noise than COBRAS.

5.5 Evaluation of noise detection

To show that our approach actually finds and corrects the majority of the noisy constraints, we calculate the average precision and average recall achieved by our noise detection algorithm when correcting a relevant constraint set with 5% of noise. For the majority of the datasets, the average recall is higher than 93% and the average precision is higher than 95%. This means our approach identifies almost all noisy constraints in the relevant constraints, and almost all constraints flagged by our approach are indeed noisy. For datasets where the average batch size is high, the precision and recall are typically lower; in this case our approximation fails to produce a reasonable estimate of the confidence. For details on the experimental set-up and full results, we refer to the appendix.

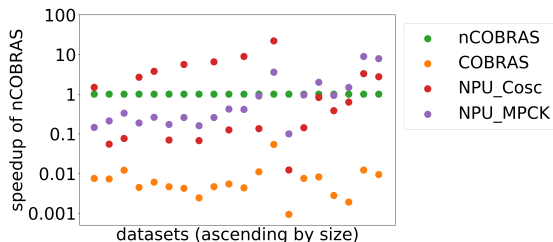


Fig. 6. Ratio of competitor to nCOBRAS average runtime for each of the 21 clustering tasks (sorted ascending by size) when there is 5% noise and the algorithms are run for 200 queries.

5.6 Runtime

Figure 6 shows the ratio of the average run time of each competitor to the average runtime of nCOBRAS for each of the 21 clustering tasks with 5% of noise for 200 queries. COBRAS is in both cases by far the fastest algorithm. nCOBRAS’ noise identification procedure makes it substantially slower than COBRAS, but it is still about as fast as the other systems.

6 Conclusion

In this paper, we have presented a novel approach to handling noise in active semi-supervised clustering. Our method reasons probabilistically about the correctness of constraints and can correct noisy constraints with the help of additional redundant queries. We integrated our method into COBRAS and have shown that this makes COBRAS significantly more robust to noise. Under noisy conditions, the noise-robust variant of COBRAS produces better clusterings compared to two other active constraint-based clustering approaches.

Acknowledgments

This research received funding from the Flemish Government under the “Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen” programme. SD is supported by the Research Foundation-Flanders (FWO)

References

1. Basu, S., Banerjee, A., Mooney, R.J.: Active semi-supervision for pairwise constrained clustering. In: Proceedings of the 2004 SIAM international conference on data mining, pp. 333–344 (2004)
2. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Twenty-first international conference on Machine learning - ICML ’04 (2004)

3. Caruana, R., Elhawary, M., Nguyen, N., Smith, C.: Meta Clustering. In: Sixth International Conference on Data Mining (ICDM'06). pp. 107–118 (2006)
4. Davidson, I., Ravi, S.S.: Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. In: Jorge, A.M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (eds.) Knowledge Discovery in Databases: PKDD 2005. pp. 59–70 (2005)
5. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences* **180**(10), 2044 – 2064 (2010)
6. Hodges, J., Lehmann, E.L., et al.: Rank methods for combination of independent experiments in analysis of variance. *The Annals of Mathematical Statistics* **33**(2), 482–497 (1962)
7. Hubert, L., Arabie, P.: Comparing partitions. *Journal of Classification* **2**, 193–218 (1985)
8. von Luxburg, U., Williamson, R.C., Guyon, I.: Clustering: Science or Art? In: Proceedings of ICML Workshop on Unsupervised and Transfer Learning. Proceedings of Machine Learning Research, vol. 27, pp. 65–79 (2012)
9. Mazumdar, A., Saha, B.: Clustering with noisy queries. In: Advances in Neural Information Processing Systems. pp. 5788–5799 (2017)
10. Pelleg, D., Baras, D.: K-Means with Large and Noisy Constraint Sets. In: Machine Learning: ECML 2007. pp. 674–682 (2007)
11. Rangapuram, S.S., Hein, M.: Constrained 1-Spectral Clustering. In: AISTATS. vol. 30, p. 90 (2012)
12. Shi, J., Malik, J.: Normalized cuts and image segmentation. *Departmental Papers (CIS)* p. 107 (2000)
13. Van Craenendonck, T., Dumančić, S., Wolputte, E.V., Blockeel, H.: COBRAS: Interactive Clustering with Pairwise Queries. In: Proceedings of the 17th International Symposium on Intelligent Data Analysis. Springer (2018)
14. Xiong, S., Azimi, J., Fern, X.Z.: Active learning of constraints for semi-supervised clustering. *IEEE Transactions on Knowledge and Data Engineering* **26**(1), 43–54 (2013)
15. Yang, T., Jin, R., K. Jain, A.: Learning from Noisy Side Information by Generalized Maximum Entropy Model. In: ICML 2010 - Proceedings, 27th International Conference on Machine Learning. pp. 1199–1206 (2010)
16. Zhu, X., Loy, C.C., Gong, S.: Constrained Clustering With Imperfect Oracles. *IEEE Transactions on Neural Networks and Learning Systems* **27**(6), 1345–1357 (2016)