COBRAS: Interactive Clustering with Pairwise Queries

Toon Van Craenendonck, Sebastijan Dumančić, Elia Van Wolputte, and Hendrik Blockeel

KU Leuven, Department of Computer Science
firstname.lastname@kuleuven.be

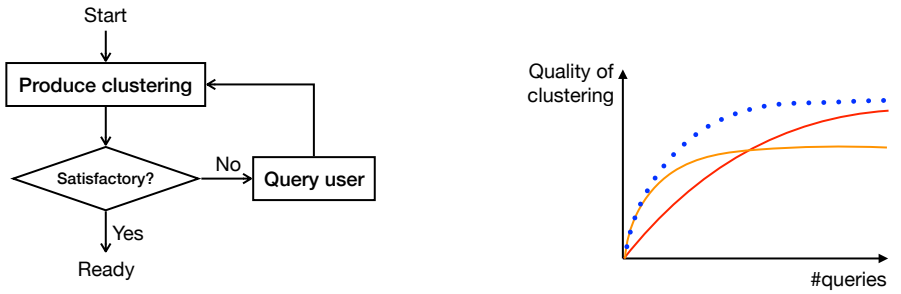
Abstract. Constraint-based clustering algorithms exploit background knowledge to construct clusterings that are aligned with the interests of a particular user. This background knowledge is often obtained by allowing the clustering system to pose pairwise queries to the user: should these two elements be in the same cluster or not? Answering yes results in a must-link constraint, no in a cannot-link. Ideally, the user should be able to answer a couple of these queries, inspect the resulting clustering, and repeat these two steps until a satisfactory result is obtained. Such an interactive clustering process requires the clustering system to satisfy three requirements: (1) it should be able to present a reasonable (intermediate) clustering to the user at *any time*, (2) it should produce good clusterings given few queries, i.e. it should be *query-efficient*, and (3) it should be *time-efficient*. We present COBRAS, an approach to clustering with pairwise constraints that satisfies these requirements. COBRAS constructs clusterings of super-instances, which are local regions in the data in which all instances are assumed to belong to the same cluster. By dynamically refining these super-instances during clustering, COBRAS is able to produce clusterings at increasingly fine-grained levels of granularity. It quickly produces good high-level clusterings, and is able to refine them to find more detailed structure as more queries are answered. In our experiments we demonstrate that COBRAS is the only method able to produce good solutions at all stages of the clustering process at fast runtimes, and hence the most suitable method for interactive clustering.

Keywords: Semi-supervised clustering · Pairwise constraints · Active clustering.

1 Introduction

Clustering is inherently subjective [4, 9]: different users often require very different clusterings of the same dataset, depending on their prior knowledge and goals. Constraint-based (or semi-supervised) clustering methods are able to deal with this subjectivity by taking a limited amount of user feedback into account. Often, this feedback is given in the form of pairwise constraints [17]. The algorithm has no direct access to the cluster labels in a target clustering, but it can perform pairwise queries to answer the question: *do instances i and j have the same cluster label in the target clustering?* A must-link constraint is obtained if the answer is yes, a cannot-link constraint otherwise.

When obtaining constraints is expensive (e.g., requires human intervention), the clustering process ideally proceeds iteratively, as summarized schematically in Figure 1(a). It is a loop where in each step the system's current estimate of the clustering is shown to



(a) interactive clustering process (b) learning curves

Fig. 1: (a) The interactive clustering process. (b) Typical learning curves with COBRA, the current state of the art. For a small number of super-instances, performance rises rapidly but stagnates at a suboptimal level (orange curve). For a higher number of super-instances, performances rises more slowly but stagnates at a higher level (red curve). The dotted line shows the learning curve that we hope to obtain with the proposed COBRAS system.

the user, and the user has the opportunity to answer several questions that will allow the system to improve the clustering, or end the process and accept the current clustering. Ideally, such a process has three properties: (1) the user can stop it at *any time* and get the best result obtained until then; (2) the number of loop executions (hence, the number of queries asked) until an acceptable result is obtained is as small as possible; (3) each loop execution is fast; e.g., a user may not want to wait more than a few seconds between queries. Summarizing this, the process must be *anytime* (in the number of queries), *query-efficient*, and *time-efficient*; we abbreviate this as AQT.

No existing constraint-based clustering system fulfills all three requirements (see next section for details). The approach closest to it is COBRA [15]. COBRA uses the concept of super-instances: sets of instances that are assumed to belong to the same cluster in the unknown target clustering. It uses constraints on the level of super-instances, rather than individual instances. This dramatically improves its query efficiency when the number of super-instances is small. However, having few super-instances increases the risk that a single super-instance contains instances from different target clusters, causing COBRA to find lower-quality clusterings. The number of super-instances N_S is a parameter of COBRA and is fixed during the clustering process. This forces the user to trade off query-efficiency with clustering quality. Figure 1(b) illustrates this: depending on N_S, COBRA quickly converges to a low-quality clustering, or slowly converges to a higher-quality clustering.

In this paper, we introduce a method for dynamically refining super-instances during clustering, based on user feedback. Extending COBRA with this method gives COBRAS (COnstraint-Based Repeated Aggregation and Splitting). The goal of this effort is to eliminate the above trade-off, and thus provide the first clustering system that meets the AQT requirements without sacrificing clustering quality; ideally its learning curve should be close to the one shown in Figure 1(b) (dotted line). An experimental evaluation confirms that COBRAS meets this goal.

2 Related work

The most common way to develop a constraint-based clustering method is to extend an existing unsupervised method. One can either adapt the clustering procedure to take the pairwise constraints into account [17, 12, 18], or use the existing procedure with a new similarity metric that is learned based on the constraints [19, 5]. Alternatively, one can also modify both the similarity metric and the clustering procedure [3, 2].

Most constraint-based clustering methods assume that a set of constraints is provided prior to running the clustering algorithm [19, 2, 3, 10]. This makes them unsuitable for anytime (in the number of constraints) clustering. Furthermore, traditional systems typically query random pairs [19, 3], which might not be the most informative ones; these are less query-efficient. Several active constraint-based clustering methods have been proposed that outperform random query selection [1, 10], but most of them still require all queries to be answered prior to clustering (query-efficient but not anytime). An exception to this is NPU [20], an active selection procedure in which the data is clustered multiple times and each resulting clustering is used to determine which pairs to query next based on the principle of uncertainty sampling. NPU is both anytime and query-efficient. However, it is not time-efficient: it requires re-clustering the entire dataset after every few constraints, which becomes prohibitively slow for large datasets.

COP-COBWEB [16] is similar to COBRAS in that it has both splitting and merging of clusters as key algorithmic steps. However, it is not anytime: it assumes that all constraints are given prior to clustering.

COBS [14] uses an approach that is very different from the above. It generates a large set of clusterings by varying the hyperparameters of several unsupervised clustering algorithms, and selects from the resulting set the clustering that satisfies the most pairwise constraints. Generating the set of clusterings, however, can be time consuming for large datasets, which reduces its suitability for anytime clustering.

COBRA [15] is a recently proposed method that is inherently active: deciding which pairs to query is part of its clustering procedure. First, COBRA uses K-means to cluster the data into super-instances. The number of super-instances, denoted as N_S, is an input parameter. Initially, each of the super-instances forms its own cluster. In the second step, COBRA repeatedly queries the pairwise relation between the closest pair of (partial) clusters between which the relation is not known yet and merges clusters if necessary, until all relations between clusters are known.

It was already mentioned in the introduction that the results of COBRA strongly depend on the number of super-instances N_S. Figure 2 illustrates this on a toy dataset. For $N_S = 10$, the initial clustering (after 0 queries) is not too bad (panel A). As queries are answered, the quality goes up, but after 14 queries it stagnates at a suboptimal level (panel D; the incorrectly clustered part is marked with a red ellipse). For $N_S = 100$, COBRA starts with a worse clustering (panel B) but ends with a better one (panel E). It takes 103 queries, however, to obtain the final clustering.

Note that the N_S parameter allows the user to trade off one disadvantage for the other, but not to remove both. The dynamic super-instance refinement procedure that we introduce with COBRAS eliminates this trade-off.

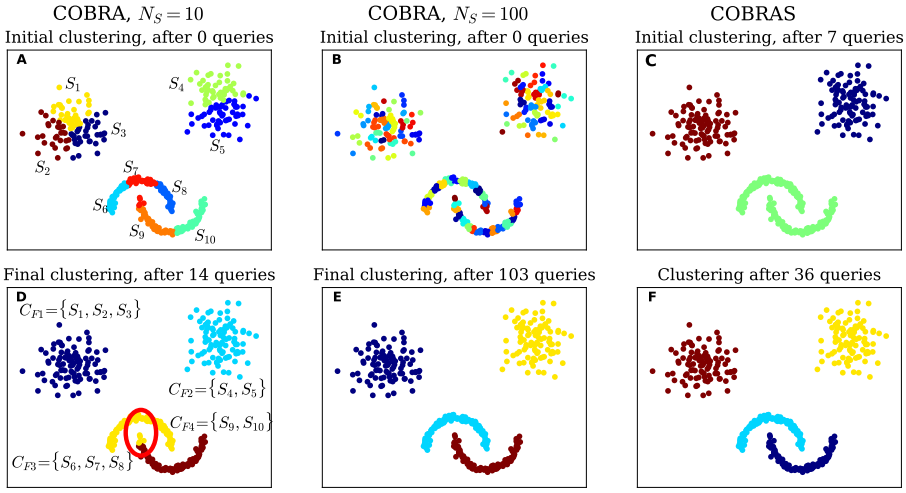


Fig. 2: A: The starting situation of COBRA with 10 super-instances (COBRA-10). Initially, each cluster consists of a single super-instance. B: The initial solution of COBRA-100, which is highly over-clustered. C: The clustering produced by COBRAS after 7 queries. D: The final result of COBRA-10. Each of the clusters is represented as a set of super-instances. The final clustering is not correct, as S_7 contains instances from two actual clusters. E: the final clustering of COBRA-100. F: after 36 queries, COBRAS produces the correct clustering.

3 COBRAS: Constraint-based Repeated Aggregation and Splitting

The key problem when running COBRA with a small N_S is that super-instances often contain instances from different clusters (e.g., S_7 in Figure 2A). COBRA cannot assign all of these instances to the correct clusters, as each super-instance is treated as a single unit.

COBRAS solves this problem by allowing super-instances to be refined. It starts with a single super-instance that contains all instances, and repeatedly refines this super-instance until a satisfactory clustering is obtained. More specifically, each iteration of COBRAS consists of two steps. First, it removes the largest super-instance from its cluster and splits it into several new super-instances. A new cluster is added for each of these new super-instances. A key challenge in this step is determining a suitable splitting level for a super-instance, i.e. the number of new super-instances that an existing one should be split in. For this, we propose a constraint-based procedure, which is detailed in 3.2.

In the second step of each iteration, COBRAS determines the relation of the newly created clusters to each other and the existing clusters by running the merging step of COBRA on the new set of clusters.

By using this procedure of refining super-instances, COBRAS uses a small number of super-instances in the beginning of the clustering process, and a larger number as

more queries are answered. This allows it to perform well for both a small and larger number of queries, as illustrated in panels C and F in Figure 2.

3.1 Algorithmic description

COBRAS is described in detail in Algorithm 1. In this algorithm a super-instance S is a set of instances, a cluster C is a set of super-instances, and a clustering $\mathcal{C}$ is a set of clusters. COBRAS starts with a single super-instance S that contains all instances, which constitutes the only cluster C (line 2). As long as the user keeps answering queries, COBRAS keeps refining the set of super-instances and the corresponding clustering (lines 3-10). In each iteration it selects the largest super-instance S_{split} (line 4), determines an appropriate splitting level for it (line 5, this is detailed in Algorithm 2 in Section 3.2), and splits it into k new super-instances by clustering its instances using K-means (line 6). We use K-means as it is faster than K-medoids, even if the medoids are computed afterwards (each K-medoid iteration is $\mathcal{O}(k(n-k)^2)$ [11] whereas each K-means iteration is $\mathcal{O}(nk)$). S_{split} is then removed from its original cluster (line 7), and a new cluster is added for each of the newly created super-instances (line 8). Finally, in the last step of the while iteration COBRA is used to determine the pairwise relations between all the clusters (new and existing). The COBRA merging step is slightly modified compared to the original one [15]: if the relation between two clusters is already known, i.e. from a query in a previous COBRAS iteration, it is not queried again. Note that one could also think of other heuristics to determine which super-instance to split instead of simply the largest one, e.g. one could split the super-instance with the highest intra-cluster dissimilarity. We have found, however, that selecting the largest super-instance is a simple and effective heuristic that is difficult to beat.

Algorithm 1 COBRAS

Input: $\mathcal{X}$: a dataset, q: a query limit
Output: $\mathcal{C}$: a clustering of D

1: $ML = \emptyset, CL = \emptyset$
2: $S = \{\mathcal{X}\}, C = \{S\}, \mathcal{C} = \{C\}$
3: **while** $|ML| + |CL| < q$ **do**
4: $\quad S_{split}, C_{origin} = \arg\max_{S \in C, C \in \mathcal{C}} |S|$
5: $\quad k, ML, CL = \texttt{determineSplitLevel}(S_{split}, ML, CL)$
6: $\quad S_{new_1}, \ldots, S_{new_k} = \texttt{K-means}(S_{split}, k)$
7: $\quad C_{origin} = C_{origin} \setminus \{S_{split}\}$
8: $\quad \mathcal{C} = \mathcal{C} \cup \{\{S_{new_1}\}, \ldots, \{S_{new_k}\}\}$
9: $\quad \mathcal{C}, ML, CL = \texttt{COBRA}(\mathcal{C}, ML, CL)$
10: **end while**
11: **return** $\mathcal{C}$

3.2 Determining the splitting level k

Different users may want different clusterings, which can require super-instances at different granularities. For example, consider clustering a set of images of 20 different

Algorithm 2 *determineSplitLevel*

Input: $\mathcal{S}$: a set of instances that is to be split
Output: k: an appropriate splitting level, ML, CL: the obtained ML and CL constraints
1: $d = 0$, $ML = \emptyset$, $CL = \emptyset$
2: **while** no must-link obtained **do**
3: $\mathcal{S}_1$, $\mathcal{S}_2$ = k-means($\mathcal{S}$,2)
4: **if** must-link(medoid($\mathcal{S}_1$), medoid($\mathcal{S}_2$)) **then**
5: add (medoid($\mathcal{S}_1$), medoid($\mathcal{S}_2$)) to ML
6: $d = \max(d, 1)$
7: **return** 2^d, ML, CL
8: **else**
9: add (medoid($\mathcal{S}_1$), medoid($\mathcal{S}_2$)) to CL
10: $\mathcal{S}$ = pick between $\mathcal{S}_1$ and $\mathcal{S}_2$ randomly
11: d++
12: **end if**
13: **end while**

people, each taking two different poses. Clustering this data based on identity will require more super-instances than clustering it based on pose. Consequently, it is crucial to take user feedback into account to determine appropriate splitting levels.

Algorithm 2 describes the procedure that COBRAS uses to determine the splitting level k for a super-instance S. The procedure tries to search for a k such that the new super-instances will be pure w.r.t. the unknown target clustering. To check the purity of S, COBRAS splits it into two new (temporary) super-instances (by running 2-means on its instances), and queries the relation between their medoids. If they must link, COBRAS assumes that the super-instance was pure, and an appropriate level of granularity has been reached. If they cannot link, the procedure is then repeated on one of the two new super-instances. This continues until a must-link constraint is obtained. If d bisections are made before an appropriate level of granularity is reached, then the super-instance as a whole must be split into 2^d smaller super-instances. Figure 3(a) illustrates this process: super-instance S_1 gets split into two super-instances which cannot link; one of these, S_{t1}, is next split into two which again cannot link; among these, S_{t3} is split into two which must link; hence, S_{t3} seems to be at the right level of granularity and S_1 is split into $2^2 = 4$, which is the number of super-instances at this level. Line 6 in Algorithm 2 makes sure that the super-instance is at least split into two, even when the first constraint is must-link. This ensures that COBRAS will continue refining super-instances as long as the user is willing to answer queries, even when the data does not provide evidence for the usefulness of a particular split.

3.3 Illustration

Figure 3 illustrates two iterations of the entire COBRAS clustering process. The splitting of S_1 into 4 smaller super-instances was already explained. These 4 super-instances are put into new clusters, and next, the standard merging process of COBRA is applied. For details about this merging process, we refer to Van Craenendonck et al. [15]. In this illustration, we assume that COBRA finds a must-link between S_4 and S_5 and

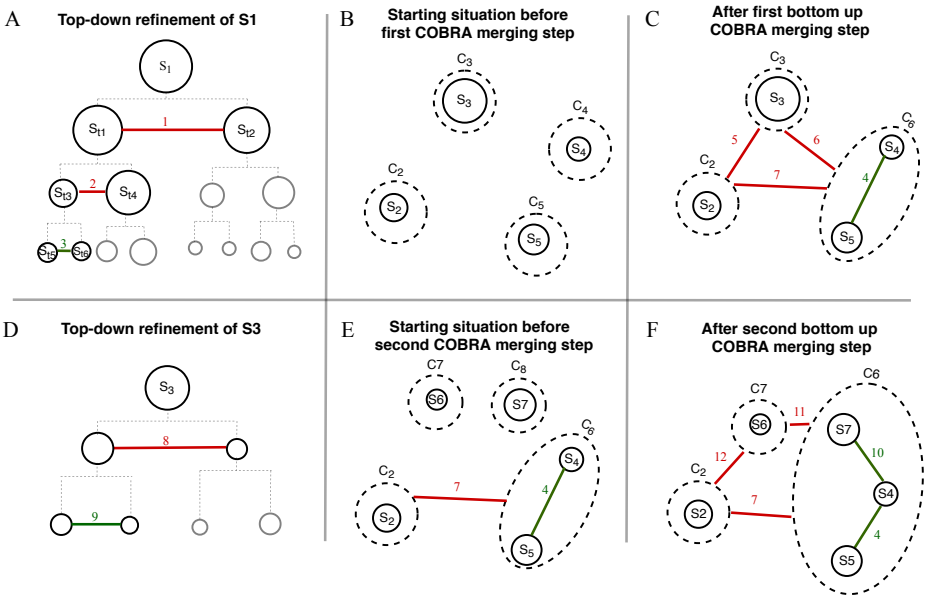


Fig. 3: (A) COBRAS decides to split the initial super-instance S_1 into 4 new ones, as discussed in section 3.2. (B) S_1 has been removed from the set of clusters (rendering it empty), and a new cluster added for each of the newly created super-instances. This is the starting situation for the first bottom-up COBRA run. (C) Using additional queries, COBRA has merged the S_4 and S_5 clusters into one, and kept the others. In the next iteration (D), COBRAS selects S_3 for refinement, and splits it into 2 new super-instances; this results in two new clusters (E). Finally (F), the merging step has clustered S_7 together with S_4 and S_5, while S_2 and S_6 remain in their own cluster.

cannot-links between the others, which results in 3 clusters. Next, super-instance S_3 is considered for splitting, and split into 2. About the resulting S_6 and S_7, COBRA finds that S_6 should remain in its own cluster, but S_7 must link with S_4 and thus the clusters $\{S_7\}$ and $\{S_4, S_5\}$ are merged. This step shows how a part of one super-instance (in this case S_7, which was originally part of S_3) can get reassigned to a more suitable cluster.

4 Experimental evaluation

In this section, we evaluate COBRAS[1] in terms of the AQT criteria (anytime, query efficiency, time efficiency). We compare it to the following state-of-the-art constraint-based clustering algorithms:

- **COBS** [14] uses constraints to select and tune an unsupervised clustering algorithm. We use the active variant in our experiments.
- **COBRA** [15] is the algorithm that is most related to COBRAS, as discussed earlier in this paper. We run it with 10, 25 and 50 super-instances.

[1] Source code for COBRAS is available at `https://dtai.cs.kuleuven.be/software/cobras/`

– **NPU** [20] is an active constraint selection framework that can be used with any non-active constraint-based clustering method. It constructs neighborhoods of points that are connected by must-link constraints, with cannot-link constraints between the different neighborhoods. It repeatedly selects the most informative instance, and queries its neighborhood membership by means of pairwise constraints. NPU is an iterative method: after neighborhood membership is determined, the data is re-clustered and the obtained clustering is used to determine the next pairwise queries. NPU can be used with any constraint-based clustering algorithm, and we use it with the following two:
  - **MPCKMeans** [3] is an extension of K-means that exploits constraints through metric learning and a modified objective. We use the implementation in the WekaUT package [2].
  - **COSC** (for Constrained Spectral Clustering) [12] is an extension of spectral clustering optimizing for a modified objective. We use the code provided by the authors [3].

COSC-NPU and MPCKMeans-NPU need to know the desired number of clusters K prior to clustering. In our experiments, the true K (as indicated by the class labels) is given to these algorithms. Note that this puts them at an advantage in the experimental comparison, as in practice K is often not known in advance.

Datasets

We use the same datasets as those used in the evaluation of COBRA [15]. These include **15 UCI datasets**: iris, wine, dermatology, hepatitis, glass, ionosphere, optdigits389, ecoli, breast-cancer-wisconsin, segmentation, column_2C, parkinsons, spambase, sonar and yeast. These were selected because of their repeated use in earlier work on constraint-based clustering (for example, [3, 20]). Optdigits389 contains digits 3, 8 and 9 of the UCI handwritten digits data [3, 10]. Duplicate instances are removed from all of these datasets, and the data is normalized between 0 and 1. Further, we use the **CMU faces** dataset, containing 624 images of 20 persons with different poses and expressions, with and without sunglasses. This dataset has four natural clustering targets: identity, pose, expression and sunglasses. A 2048-value feature vector is extracted for each of the images using the pre-trained Inception-V3 network [13]. Further, two clustering tasks are included for the **20 newsgroups** text dataset: clustering documents from 3 newsgroups on related topics (the target clusters are comp.graphics, comp.os.ms-windows and comp.windows.x, as in [1, 10]), and clustering documents from 3 newsgroups on very different topics (alt.atheism, rec.sport.baseball and sci.space, as in [1, 10]). To extract features from the text documents we apply tf-idf, followed by latent semantic indexing (as in [10]) to reduce the dimensionality to 10.

In summary, the comparison is based on 21 clustering tasks (15 UCI datasets, 4 target clusterings for the CMU faces data, and 2 subsets of the newsgroups data).

[2] http://www.cs.utexas.edu/users/ml/risc/code/

[3] http://www.ml.uni-saarland.de/code/cosc/cosc.htm

Experimental methodology

We perform 10-fold cross-validation 10 times (similar to e.g. [1] and [10]), and report averaged results. The algorithms always cluster the full dataset, but can only query the relations between pairs that are both in the training set. The quality of the resulting clustering is evaluated by computing the Adjusted Rand index (ARI, [8]), only on the instances in the test set. The ARI measures the similarity between the produced clusterings and the ground-truth indicated by the class labels. A score of 0 means that the clustering is random, 1 means that it is exactly the same as the ground-truth. The score for an algorithm for a particular dataset is given by the average ARI over the 10 repetitions of 10 fold cross-validation.

We make sure that COBRAS and COBRA do not query any test instances during clustering by only using training instances to compute the medoids of the super-instances. For NPU, pairs involving an instance from the test set are simply excluded from selection.

In each iteration of COBRAS, a super-instance is split and COBRA is run on the resulting new set of clusterings. If the user stops answering pairwise queries before the end of the COBRA run (which is simulated frequently in the experiments: we consider the intermediate clusterings after each query), COBRAS returns the clustering as it was at the beginning of the iteration. The clustering that is returned is only updated after the COBRA run, which prevents us from returning clusterings for which the merging step was not finished yet. This holds for all COBRA runs expect the first one, as in that case there is no real prior clustering at the beginning of the iteration.

COBRA is not able to handle an unlimited amount of pairwise queries: once all the relations between super-instances are known, the clustering process naturally stops. In our experiments, we assume that COBRA simply keeps returning its final clustering after this point, which allows us to compare all algorithms for the same number of pairwise queries.

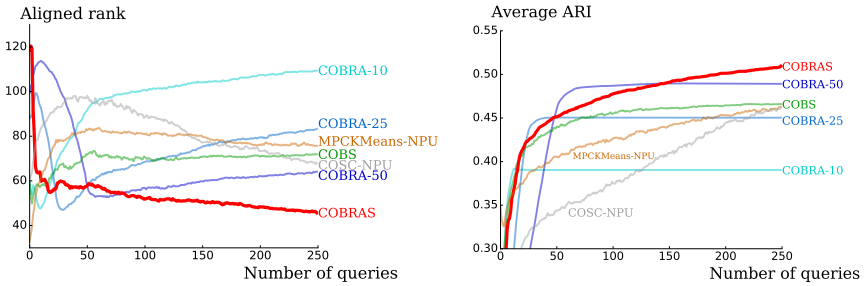
Clustering quality

Figure 4(a) shows the aligned ranks for COBRAS and all competitors over all clustering tasks[4]. In contrast to the regular rank, the aligned rank [7, 6] takes the relative differences between algorithms for individual datasets into account. The first step in computing it is to calculate for each dataset the average ARI achieved by the algorithms. Then, for each algorithm, the difference between its ARI and this average is calculated, and the resulting differences are sorted from 1 to kn (k the number of algorithms, n the number of datasets). The aligned rank for an algorithm is the average of the positions of its entries in the sorted list.

Figure 4(b) shows the average ARI of each method over all clustering tasks. This gives some indication of how substantial the differences in ARI are in practice.

Figures 4(a) and 4(b) show that, *compared over the entire range of queries*, COBRAS is clearly superior to each individual competitor. None of the competitors is

[4] For COSC-NPU we set a timeout of 24h for each run of 250 queries for spambase. Typically it only got to 40 queries after that time. We considered the last clustering produced within 24h to be the final one, and use it in the results for all remaining queries in producing the graphs.



(a) (b)

Fig. 4: (a) Aligned rank for all methods over all clustering tasks (b) Average ARIs for all methods over all clustering tasks

able to produce good results during the entire clustering process, which is crucial for interactive clustering. Some of them outperform COBRAS for a specific range of the number of queries, but those that do are outperformed by a much larger margin for other ranges. We illustrate this point by comparing COBRAS to COBRA-50 in more detail. Figure 4(a) shows that COBRA-50 outperforms COBRAS in the range of (roughly) 50-70 queries. However, for < 50 queries, COBRAS-50 performs much worse than COBRAS; the difference in average ARI in this range is much greater than in the 50-70 range. Furthermore, COBRA-50's performance stagnates around 50 queries. Thus, the anytime behavior of COBRA-50 is vastly inferior to that of COBRAS. COBRA-50 is only preferable to COBRAS when one knows the optimal number of super-instances in advance. The same holds for COBRA-10 and COBRA-25.

Table 1 shows win/loss statistics that confirm the above conclusions. It demonstrates that COBRAS outperforms its competitors in the majority of cases (18 out of 24). COBRAS significantly (Wilcoxon test, $p < 0.05$) outperforms COBRA-10, COBRA-25, COBRA-50 and COSC-NPU for at least one of the query numbers. It outperforms MPCKMeans-NPU and COBS as well, but this difference is found not to be statistically significant. It is never significantly outperformed by any other method.

Table 1: Wins and losses over the 21 clustering tasks. An asterisk indicates that the difference is significant according to the Wilcoxon test with $p < 0.05$. Between parentheses we report the average margin by which COBRAS wins or loses.

	25 queries		50 queries		100 queries		200 queries	
	win	loss	win	loss	win	loss	win	loss
COBRAS vs. COBRA-10	**11** (0.05)	10 (0.02)	**16*** (0.07)	5 (0.01)	**17*** (0.10)	4 (0.01)	**18*** (0.12)	3 (0.01)
COBRAS vs. COBRA-25	7(0.03)	**14** (0.04)	9 (0.03)	**12** (0.03)	**14** (0.04)	7 (0.01)	**17*** (0.06)	4 (0.01)
COBRAS vs. COBRA-50	**16*** (0.15)	5 (0.01)	9 (0.04)	**12** (0.04)	9 (0.02)	**12** (0.02)	**12** (0.03)	9 (0.01)
COBRAS vs. MPCKM-NPU	**11** (0.06)	10 (0.02)	**13** (0.07)	8 (0.02)	**11** (0.07)	10 (0.02)	**11** (0.06)	10 (0.02)
COBRAS vs. COSC-NPU	**14*** (0.13)	7 (0.02)	**15*** (0.13)	6 (0.02)	**13** (0.13)	8 (0.03)	9 (0.10)	**12** (0.04)
COBRAS vs. COBS	**12** (0.04)	9 (0.03)	**13** (0.04)	8 (0.03)	**12** (0.05)	9 (0.03)	10 (0.06)	**11** (0.02)

Runtime

Figure 5 shows the ratio of the run time of COBRAS to the run times of its competitors for the 21 clustering tasks after performing 100 queries. COBRA is typically the fastest algorithm. This is not surprising, as it requires only a single run of K-means, while COBRAS requires multiple K-means runs. Compared to the other competitors, COBRAS is *one to three orders of magnitude* faster for all datasets. The key difference between COBRAS and its competitors is that COBRAS only re-clusters the parts of the dataset that are being refined. In contrast, MPCKMeans-NPU and COSC-NPU require frequent re-clustering of the entire dataset.

Practically speaking, the time between consecutive queries is under a second for all datasets considered here, which is fast enough for interactive clustering.

The high runtimes of COBS are caused by the fact that it generates a large number of unsupervised clusterings prior to querying the user. Once this set of clusterings is generated, however, selecting the clusterings is fast. This means that COBS can be useful in interactive settings where the time between starting the system and answering the first query is of no concern.

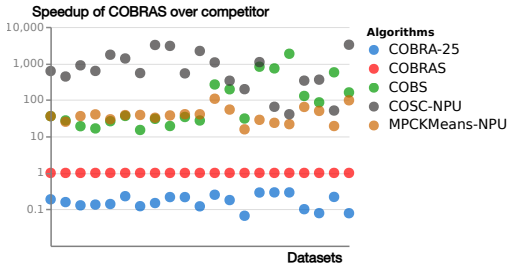


Fig. 5: Ratio of COBRAS to competitors run time for 21 clustering tasks. For COBRA we only include the run times of COBRA-25 to not clutter the graph, also the run times for COBRA-10 and COBRA-50 are typically lower than all others.

To summarize all the above: COBRA and possibly COBS can compete with COBRAS in terms of time efficiency; COBRA can compete in terms of query efficiency if its N_S parameter is chosen optimally; none of the existing methods can compete in terms of anytime behavior.

5 Conclusion

We have introduced COBRAS, a novel system for interactive semi-supervised clustering. The key innovation in COBRAS is its procedure for dynamically refining super-instances. This innovation makes it the first clustering system to excel at all three of the following crucial criteria for interactive clustering systems: anytime behavior, query efficiency, and time efficiency. This should make COBRAS the method of choice in many applications of semi-supervised clustering.

Acknowledgements

TVC is supported by the Agency for Innovation by Science and Technology in Flanders (IWT). Research supported by Research Fund KU Leuven (GOA/13/010), Research Foundation - Flanders (G079416N), and the European Research Council (Horizon 2020, grant agreement 694980, "SYNTH").

References

1. Basu, S., Banerjee, A., Mooney, R.J.: Active semi-supervision for pairwise constrained clustering. In: Proc. of SDM 2004
2. Basu, S., Bilenko, M., Mooney, R.J.: A probabilistic framework for semi-supervised clustering. In: Proc. of KDD 2004
3. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proc. of ICML 2004
4. Caruana, R., Elhawary, M., Nguyen, N.: Meta clustering. In: Proc. of ICDM 2006
5. Davis, J.V., Kulis, B., Jain, P., Sra, S., Dhillon, I.S.: Information-theoretic metric learning. In: Proc. of ICML 2007
6. 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 (2010), special Issue on Intelligent Distributed Information Systems
7. Hodges, J.L., Lehmann, E.L.: Rank methods for combination of independent experiments in analysis of variance. The Annals of Mathematical Statistics (1962)
8. Hubert, L., Arabie, P.: Comparing partitions. Journal of Classification (1985)
9. von Luxburg, U., Williamson, R.C., Guyon, I.: Clustering: Science or Art? In: Workshop on Unsupervised Learning and Transfer Learning (2014)
10. Mallapragada, P.K., Jin, R., Jain, A.K.: Active query selection for semi-supervised clustering. In: Proc. of ICPR 2008
11. Ng, R.T., Han, J.: Clarans: A method for clustering objects for spatial data mining. IEEE TKDE **14**(5), 1003–1016 (2002)
12. Rangapuram, S.S., Hein, M.: Constrained 1-spectral clustering. In: Proc. of AISTATS 2012
13. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR **abs/1512.00567** (2015), `http://arxiv.org/abs/1512.00567`
14. Van Craenendonck, T., Blockeel, H.: Constraint-based clustering selection. In: Machine Learning (2017)
15. Van Craenendonck, T., Dumančić, S., Blockeel, H.: COBRA: A Fast and Simple Method for Active Clustering with Pairwise Constraints. In: Proc. of IJCAI 2017
16. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proc. of ICML 2000
17. Wagstaff, K., Cardie, C., Rogers, S., Schroedl, S.: Constrained K-means Clustering with Background Knowledge. In: Proc. of ICML 2001
18. Wang, X., Qian, B., Davidson, I.: On constrained spectral clustering and its applications. Data Mining and Knowledge Discovery, 2014
19. Xing, E.P., Ng, A.Y., Jordan, M.I., Russell, S.: Distance metric learning, with application to clustering with side-information. In: NIPS 2003
20. Xiong, S., Azimi, J., Fern, X.Z.: Active learning of constraints for semi-supervised clustering. TKDE, 2014