Mach Learn
DOI 10.1007/s10994-013-5350-y

A flexible cluster-oriented alternative clustering algorithm for choosing from the Pareto front of solutions

Duy Tin Truong · Roberto Battiti

Received: 25 May 2012 / Accepted: 3 April 2013

Abstract Supervised alternative clustering is the problem of finding a set of clusterings which are of high quality *and* different from a given *negative* clustering. The task is therefore a clear multi-objective optimization problem. Optimizing two conflicting objectives at the same time requires dealing with trade-offs. Most approaches in the literature optimize these objectives sequentially (one objective after another one) or indirectly (by some heuristic combination of the objectives). Solving a multi-objective optimization problem in these ways can result in solutions which are dominated, and not Pareto-optimal. We develop a direct algorithm, called **COGNAC**, which fully acknowledges the multiple objectives, optimizes them directly and simultaneously, and produces solutions approximating the Pareto front. **COGNAC** performs the recombination operator at the *cluster level* instead of at the object level, as in the traditional genetic algorithms. It can accept arbitrary clustering quality and dissimilarity objectives and provides solutions dominating those obtained by other state-of-the-art algorithms. Based on **COGNAC**, we propose another algorithm called **SGAC** for the sequential generation of alternative clusterings where each newly found alternative clustering is guaranteed to be different from all previous ones. The experimental results on widely used benchmarks demonstrate the advantages of our approach.

Keywords Alternative clustering · Multi-objective optimization · Cluster-oriented recombination · Genetic algorithms

1 Introduction

Given a dataset, traditional clustering algorithms often only provide a single set of clusters, a single view of the dataset. On complex tasks, many different ways of clustering exist,

Editors: Emmanuel Müller, Ira Assent, Stephan Günnemann, Thomas Seidl, and Jennifer Dy.

D.T. Truong (✉) · R. Battiti
Department of Information Engineering and Computer Science, University of Trento, Trento, Italy
e-mail: truong@disi.unitn.it

R. Battiti
e-mail: battiti@disi.unitn.it

therefore a natural requirement is to ask for *alternative* clusterings, to have complementary views. Clustering flowers depending on colors and aesthetic characteristics can be suitable for a florist, but not for a scientist, who usually prefers different organizations.

Recently, many techniques have been developed for solving the *alternative clustering* problem. They can be split into two groups: unsupervised or supervised. In unsupervised alternative clustering, the algorithm automatically generates a set of alternative clusterings of high quality and different from each other (Jain et al. 2008; Dang and Bailey 2010; Dasgupta and Ng 2010; Niu et al. 2010; Günnemann et al. 2012; De Bie 2011). Unsupervised alternative clustering is useful if users do not know what they want and need some initial options. In other cases, users already know some trivial or *negative* clusterings, and they ask for different and potentially more informative clusterings. These algorithms are called supervised because the user is directing the alternative clustering by explicitly labeling some clusterings as undesired, or *negative*.

This paper focuses on *supervised alternative clustering*, the problem of finding new clusterings of good quality *and* as different as possible from a given negative clustering. Supervised alternative clustering is a multi-objective optimization problem with two objectives of clustering quality and dissimilarity, and the goal is to find a representative set of Pareto-optimal solutions. A Pareto-optimal solution is a solution such that there is no solution which improves at least one objective without worsening the other objectives. The Pareto front is the set of all Pareto-optimal solutions in the objective space. Most approaches in the literature only optimize the two objectives *sequentially* (optimizing one objective first and then optimizing the other one) (Davidson and Qi 2008; Qi and Davidson 2009) or *indirectly* by some heuristics (Bae and Bailey 2006; Cui et al. 2007). Other methods combine the two objectives into a single one and then optimize this single objective (Gondek and Hofmann 2003; Vinh and Epps 2010).

Solving a multi-objective optimization problem in the above ways can result in solutions which are not Pareto-optimal, or in a single or a very limited number of solutions on the Pareto front. The user flexibility is thus limited because the tradeoff between the different objectives is decided *a priori*, before knowing the possible range of solutions. The tradeoff can be decided in a better way *a posteriori*, by generating a large set of representative solutions along the Pareto front and having the user pick the favorite one among them. More practical approaches are *interactive* and incorporate *machine learning*: some initial information is given but "intelligent optimization" schemes collect user feedback about the initial solutions and direct the software to explore the most interesting areas of the objective space (Battiti et al. 2008; Battiti and Passerini 2010).

Some approaches are developed for specific clustering algorithms, and are therefore limited in their applications, or require setting parameters which influence the preference between clustering quality and dissimilarity. Parameter tuning by the final user is a difficult task since some dominating solutions can be lost because of improper settings.

In addition, most current alternative clustering algorithms can only accept one negative clustering $\overline{\mathbf{C}}$ and generate an alternative clustering $\mathbf{C}_1$ different from $\overline{\mathbf{C}}$. Therefore, one cannot generate a second alternative clustering $\mathbf{C}_2$ by simply rerunning the algorithm with $\mathbf{C}_1$ as the negative clustering. The second alternative clustering $\mathbf{C}_2$ will be different from $\mathbf{C}_1$ but often very similar to $\overline{\mathbf{C}}$, because $\overline{\mathbf{C}}$ is not considered when computing $\mathbf{C}_2$. In order to generate a sequence of alternative clusterings, where each one is different from the other ones, a more complex algorithm which can accept a *set* of negative clusterings is required.

To deal with the above issues, we propose an explicit multi-objective algorithm, called **C**luster-**O**riented **G**e**N**etic algorithm for **A**lternative **C**lusterings (**COGNAC**), with the following advantages:

- Optimizing directly and simultaneously the predefined objectives (clustering quality and dissimilarity).
- Generating a sequence of alternative clusterings where each newly generated clustering is guaranteed to be different from previous alternative clusterings.

In Truong and Battiti (2012), we introduced the basic version of **COGNAC** which is limited to the case where the number of clusters in both negative clusterings and alternative clusterings are the same. In this paper, we extend **COGNAC** flexibility to handle cases where the number of clusters can be different and propose a new algorithm called **SGAC** for the sequential generation of alternative clusterings. We also propose techniques for analyzing the Pareto front returned by **COGNAC** to help users select the preferred solutions. Moreover, we present detailed experiments with a thorough analysis on the Pareto solutions and compare them with the *ground truth* alternative solutions.

The rest of this paper is organized as follows. We first discuss the related work in Sect. 2. Then, we formally define the alternative clustering problem in Sect. 3.1. In Sect. 3.2, we summarize the non-dominated sorting framework of Deb et al. (2002) and then detail our algorithm **COGNAC**. We then propose the new algorithm **SGAC** in Sect. 3.3. Some techniques for analyzing and visualizing the Pareto front returned by our algorithms are presented in Sect. 3.4. We describe the test datasets in Sect. 4.1 and the experiments to compare the performance of our algorithm with that of other state-of-the-art proposals on the first alternative clustering in Sect. 4.2. Then, we illustrate the ability of our algorithm for generating a sequence of different alternative clusterings in Sect. 4.3. Finally, we analyze the parameter sensitivity of our method in Sect. 4.4.

2 Related work

Among the first algorithms in supervised alternative clustering is Conditional Information Bottleneck (**CIB**) (Gondek and Hofmann 2003), based on the information bottleneck (IB) method (Tishby et al. 2000). Their approach in modelling the clustering problem is similar to that of data compression. Given two variables X representing objects and Y representing features, and a negative clustering Z, the **CIB** algorithm finds an alternative clustering C which is different from Z but still good in quality by maximizing the mutual information $I(C;Y|Z)$ between C and Y given Z under the constraint that the mutual information (or information rate) $I(C;X)$ between C and X is less than a threshold R. However, this approach requires an explicit joint distribution between the objects and the features which can be very difficult to estimate.

Bae and Bailey (2006) show that the clusterings found by **CIB** are not of high quality if compared with those found by their **COALA** algorithm, which extends the agglomerative hierarchical clustering algorithm. Let d_1 be the smallest distance between two arbitrary clusters and d_2 be the smallest distance between two clusters where merging them does not violate the constraints (generated by the negative clustering). If the ratio $\frac{d_1}{d_2}$ is less than a threshold, then two nearest clusters are merged to preserve the clustering quality. Otherwise, two clusters with the distance of d_2 are merged to find dissimilar clusterings. A drawback is that it only considers *cannot-link* constraints, hence useful information which can be obtained through *must-link* constraints is lost. In addition, the application scope of the method is limited because it was developed particularly for the agglomerative clustering algorithms.

To overcome the scope limitation, Davidson and Qi (2008) propose a method, called **AFDT** which transforms the dataset into a different space where the negative clustering is difficult to be detected and then uses an arbitrary clustering algorithm to partition the transformed dataset. However, transforming the dataset into a different space can destroy the characteristics of the dataset. Qi and Davidson (2009) fix this problem by finding a transformation which minimizes the Kullback-Leibler divergence between the probability distribution of the dataset in the original space and the transformation space, under the constraint that the negative clustering should not be detected. This approach requires specifying user preference at the clustering quality and the clustering dissimilarity. In this paper, we will refer this approach as **AFDT2**.

Alternative clustering can also be discovered by two orthogonalization algorithms proposed in Cui et al. (2007). The algorithms first project the dataset into an orthogonal subspace and then apply an arbitrary clustering algorithm on the transformed dataset. However, when the objects of the dataset are in low-dimensional spaces, the orthogonal subspace may not exist (Davidson and Qi 2008). In addition, a requirement of Cui et al.'s algorithms which is the number of clusters must be smaller than the number of dimensions in the original dataset may not be satisfied in many practical cases. In fact, Davidson and Qi (2008) show that **AFDT** outperforms Cui et al.'s algorithms.

The above algorithms can only accept one negative clustering. Recently, Nguyen et al. (Vinh and Epps 2010) propose **MinCEntropy**$^{++}$, which can accept a set of $N_{\overline{C}}$ negative clusterings. **MinCEntropy**$^{++}$ finds an alternative clustering $\mathbf{C}_*$ by maximizing the weighted sum:

$$N_{\overline{C}}\mathbb{I}(\mathbf{C};\mathbf{X}) - \lambda \sum_{i=1}^{N_{\overline{C}}} \mathbb{I}(\mathbf{C};\overline{\mathbf{C}}_i)$$

where $\mathbb{I}(\mathbf{C};\mathbf{X})$ is the mutual information between a clustering $\mathbf{C}$ and the dataset $\mathbf{X}$; $\mathbb{I}(\mathbf{C};\overline{\mathbf{C}}_i)$ is the mutual information between a clustering $\mathbf{C}$ and a negative clustering $\overline{\mathbf{C}}_i$; λ is the parameter trading-off the importance between clustering quality and dissimilarity.

Dasgupta and Ng (2010) propose an unsupervised algorithm for generating a set of alternative clusterings. In this paper, we will refer this algorithm as **Alter-Spect** because it is based on the spectral clustering algorithm. **Alter-Spect** first forms the Laplacian matrix L of the dataset and then computes the second through $(m+1)$-th eigenvectors of L where m is the number of alternative clusterings that users want to generate. Then, it runs **K-Means** on these eigenvectors to produce m different alternative clusterings. The main intuition of **Alter-Spect** and other subspace clustering approaches (Niu et al. 2010; Günnemann et al. 2012; De Bie 2011) is that different alternative clusterings can exist in different subspaces. In contrast, our algorithm considers all features and optimizes two objectives in parallel. In detail, Nui et al. suggest an algorithm which learns low-dimensional subspaces for different views by optimizing a fixed single objective which is the combination of two alternative clustering objectives (Niu et al. 2010), i.e., the quality of all clustering is as high as possible, and the redundancy between them is as low as possible. Günnemann et al. also propose a Bayesian framework for determining different views in subspaces (Günnemann et al. 2012). The authors generalize the dataset by using multiple mixture models where each mixture of Beta distributions presents a specific view. As these mixtures can compete against each other in the data generation process, their framework can handle overlapping views and subspace clusters. Instead of generating alternative clusterings as in other approaches, De Bie (2011) describes an algorithm which generates a sequence of different clusters. In each iteration, the algorithm searches for the next cluster surprising the user most, given the previous clusters. The user surprise is inversely proportional to the probability that the new cluster appears under a predefined data distribution. However, as this framework only focuses on the surprise aspect of the next cluster but does not consider its quality, a poor quality but highly-surprising cluster can be returned.

For optimizing directly the clustering objectives, Handl and Knowles (2007) propose an evolutionary algorithm for multi-objective clustering, called **MOCK**, which uses the graph-based representation to encode the clustering solutions. Each clustering solution $\mathbf{C}_t$ of N data objects $\{\mathbf{x}_i\}_{i=1}^N$ is represented as a N-dimensional vector and the i-th element $\mathbf{C}_t(i)$ stores the index of the object $\mathbf{x}_j$ to which the i-th object $\mathbf{x}_i$ is connected. All data objects in the same connected components are extracted to form clusters. The clustering solution returned by **MOCK** can have an arbitrarily large number of clusters because the number of connected components can be varied from 1 to N. The number of clusters in alternative clustering is often fixed to make it easier to compare different clusterings. Actually, this is necessary when comparing two clusterings on quality objectives like Vector Quantization Error (VQE) of **K-Means** (Lloyd 1982), because the value of VQE decreases when the number of clusters increases. However, fixing the number of clusters decreases **MOCK** performance in a radical way, because many clustering solutions become invalid and are discarded when applying the standard procedures of initialization, recombination, and mutation. Therefore, it is difficult to extend **MOCK** for the alternative clustering problem. This is the reason why in this paper, we will only compare our algorithm against **COALA**, **AFDT**, **AFDT2**, **MinCEntropy**$^{++}$ and **Alter-Spect**.

3 A cluster-oriented genetic algorithm for alternative clustering

In this section, we first formally define the problem of alternative clustering in Sect. 3.1 and then detail our algorithm **COGNAC** to address such problem in Sect. 3.2. Then, based on **COGNAC**, we propose another algorithm (**SGAC**) for generating a set of different alternative clusterings in Sect. 3.3. Some techniques for analyzing and visualizing the Pareto front returned by our algorithm are also presented in Sect. 3.4.

3.1 The alternative clustering problem

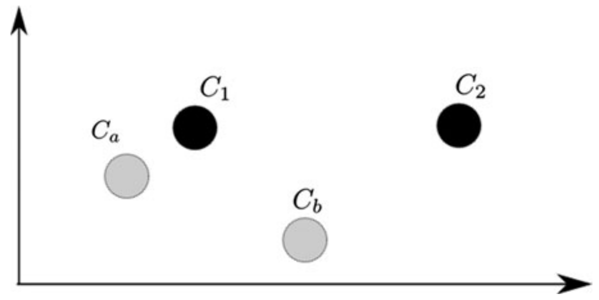
Given a dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$ of N objects, the traditional clustering problem is to partition this dataset into K disjoint clusters such that the clustering quality is as high as possible. Let $\mathbf{C}$ be a clustering solution where $\mathbf{C}(i)$ is the index of the cluster that $\mathbf{x}_i$ belongs to, $\mathbb{D}(\mathbf{C}, \overline{\mathbf{C}})$ be the dissimilarity between two clusterings $\mathbf{C}$ and $\overline{\mathbf{C}}$, and $\mathbb{Q}(\mathbf{C})$ be the inner quality of a clustering $\mathbf{C}$. We defer the definition of $\mathbb{D}(\mathbf{C}, \overline{\mathbf{C}})$ and $\mathbb{Q}(\mathbf{C})$ to Sect. 3.2 where we also present other components of our algorithm and define in this section the dissimilarity between a clustering and a set of clusterings, the overall quality of a clustering and the dominance relation between two clusterings.

Definition 1 (Dissimilarity) The *dissimilarity* between a clustering $\mathbf{C}$ and a set $\overline{\mathbf{S}}$ of clusterings is the minimum dissimilarity between $\mathbf{C}$ and all clusterings $\overline{\mathbf{C}} \in \overline{\mathbf{S}}$:

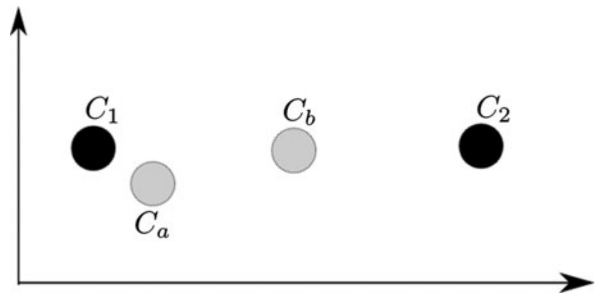
$$\mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) = \min_{\overline{\mathbf{C}} \in \overline{\mathbf{S}}} \mathbb{D}(\mathbf{C}, \overline{\mathbf{C}}) \tag{1}$$

In Fig. 1a and Fig. 1b, we illustrate the benefit of using minimum dissimilarity over maximum dissimilarity and average dissimilarity in defining the dissimilarity between a clustering and a clustering set. Assume that the clustering set is $\overline{\mathbf{S}} = \{\mathbf{C}_1, \mathbf{C}_2\}$ and we want to select a clustering between two clusterings $\mathbf{C}_a$ and $\mathbf{C}_b$ based on the dissimilarity. As it can be seen in Fig. 1a, although $\mathbf{C}_a$ is very similar (or can be equal to) $\mathbf{C}_1$ but its maximum dissimilarity to the clustering set $\overline{\mathbf{S}}$ (which is the dissimilarity between $\mathbf{C}_a$ and $\mathbf{C}_2$) is greater

Fig. 1 The benefit of using minimum dissimilarity over maximum and average dissimilarity



(a) Maximum dissimilarity prefers solution $\mathbf{C_a}$ while minimum dissimilarity prefers solution $\mathbf{C_b}$.



(b) Average dissimilarity prefers solution $\mathbf{C_a}$ while minimum dissimilarity prefers solution $\mathbf{C_b}$.

than the maximum dissimilarity of $\mathbf{C}_b$ to $\overline{\mathbf{S}}$ (which is the dissimilarity between $\mathbf{C}_b$ and $\mathbf{C}_2$). Therefore, based on the maximum dissimilarity, $\mathbf{C}_a$ will be selected. However, from human interpretation, $\mathbf{C}_b$ is more different from the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C}_a$. Similarly, in Fig. 1b, $\mathbf{C}_a$ has a higher average dissimilarity to the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C}_b$ and $\mathbf{C}_a$ will be selected if the average dissimilarity is used. However, $\mathbf{C}_b$ is clearly more different from the clustering set $\overline{\mathbf{S}}$ than $\mathbf{C}_a$.

Definition 2 (Overall Clustering Quality) The *overall quality* of a clustering $\mathbf{C}$ is characterized by the following bi-objective function $\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}})$:

$$\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}}) = \big[\mathbb{Q}(\mathbf{C}), \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}})\big] \tag{2}$$

$$\text{where} \begin{cases} \overline{\mathbf{S}} & \text{is a given negative clustering set.} \\ \mathbb{Q}(\mathbf{C}) & \text{is the quality of a clustering } \mathbf{C}. \\ \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) & \text{is the dissimilarity between } \mathbf{C} \text{ and } \overline{\mathbf{S}}. \end{cases}$$

Definition 3 (Clustering Dominance) Given a set $\overline{\mathbf{S}}$ of negative clusterings, a clustering $\mathbf{C}$ *dominates* another clustering $\mathbf{C}'$ w.r.t $\overline{\mathbf{S}}$, written $\mathbf{C} \succ_{\overline{\mathbf{S}}} \mathbf{C}'$ iff one quality objective of $\mathbf{C}$ is better than or equal to that of $\mathbf{C}'$ and the other objective of $\mathbf{C}$ is strictly better than that of $\mathbf{C}'$. Formally, $\mathbf{C} \succ_{\overline{\mathbf{S}}} \mathbf{C}'$ if and only if the following conditions hold:

$$\begin{aligned}(\mathbf{C} \neq \mathbf{C}') \wedge \big[&\big(\mathbb{Q}(\mathbf{C}) > \mathbb{Q}(\mathbf{C}') \wedge \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) \geq \mathbb{D}(\mathbf{C}', \overline{\mathbf{S}})\big) \\ &\vee \big(\mathbb{Q}(\mathbf{C}) \geq \mathbb{Q}(\mathbf{C}') \wedge \mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) > \mathbb{D}(\mathbf{C}', \overline{\mathbf{S}})\big)\big]\end{aligned} \tag{3}$$

Finally, the alternative clustering problem is defined as follows:

Definition 4 (Alternative Clustering) Given a set $\overline{\mathbf{S}}$ of negative clusterings, *alternative clustering* is the problem of finding a representative set of clusterings $\mathbf{C}$ along the Pareto front defined by the bi-objective function $\mathbb{F}(\mathbf{C}, \overline{\mathbf{S}})$.

3.2 A cluster-oriented algorithm for alternative clustering

In multi-objective optimization problems, usually there are efficient optimal solutions which cannot be compared. Therefore, one of the main goals of multi-objective optimization is to approximate the Pareto front. Evolutionary algorithms (EAs) possess several characteristics that are well suitable for multi-objective optimization. EAs approximate the Pareto front in a single run by maintaining a solution set or population. In each iteration, this solution set $\mathbf{Q}$ is modified by two basic steps: selection and variation. The selection step chooses only well-adapted candidates from $\mathbf{Q}$ to form a set $\mathbf{P}$ of parent solutions. Then, in the variation step, the parent candidates in $\mathbf{P}$ are used to produce the next generation through recombination and mutation. The two steps are repeated until a number of generations is reached. We adapt one of the most popular EAs, **NSGAII** (Deb et al. 2002), for the alternative clustering problem defined in Sect. 3.1.

Applying EAs to the clustering problem is not straightforward because the traditional recombination and mutation operators of EAs are not very suitable for the clustering problem. The first reason is that they often cannot produce offspring solutions with good characteristics inherited from their parents (Hruschka et al. 2009). Besides, in the case of fixed number of clusters, these operators can also produce invalid solutions. Therefore, when applying **NSGAII**, we replace these operators by two new operators called *Cluster-Oriented Recombination* and *Neighbor-Oriented Mutation* which can produce a valid offspring with good properties inherited from its parents. We defer the discussion of the deficiencies of the traditional genetic operators in Sects. 3.2.4 and 3.2.5, where we also describe our new genetic operators in detail. In the next sections, we summarize the **NSGAII** mechanism and present our modifications for the alternative clustering problem.

*3.2.1 Fast nondominated sorting algorithm (**NSGAII**)*

The pseudo code of **NSGAII** is shown in Algorithm 1. Let P be the fixed size of populations generated in the algorithm. The algorithm first creates an initial parent population $\mathbf{P}_0$ and then produces an offspring population $\mathbf{Q}_0$ from $\mathbf{P}_0$ by the usual binary tournament selection, recombination, and mutation operators. The binary tournament selection procedure picks randomly two solutions and returns the non-dominated one. The set of non-dominated solutions $\mathbf{P}_{\text{best}}$ is initialized as the set of non-dominated solutions in $\mathbf{P}_0 \cup \mathbf{Q}_0$. Then for each generation t, the procedure *FastNonDominatedSort*($\mathbf{R}_t$) classifies all solutions in the combined population $\mathbf{R}_t = \mathbf{P}_t \cup \mathbf{Q}_t$ into different dominance fronts (sorted in the ascending order of dominance depth where the first front is the set of non-dominated solutions). The pseudo code of the *FastNonDominatedSort* procedure is presented in Algorithm 2. We denote n_p the number of solutions that dominate a solution $\mathbf{p} \in \mathbf{P}$ and $\mathbf{S}_p$ the set of solutions that $\mathbf{p}$ dominates. The solutions $\mathbf{p}$ with $n_p = 0$ are placed in the first non-dominated front. Then, for each solution $\mathbf{p}$ with $n_p = 0$, we visit each member $\mathbf{q}$ of its dominated set $\mathbf{S}_p$ and reduce its domination count by one. When doing so, if $n_q = 0$ then we put $\mathbf{q}$ in a separate list $\mathbf{Q}$. These members will belong the second nondominated front. The above process is repeated until all solutions are classified. The complexity of this procedure is $O(P^2 \Lambda)$ where P is the

Algorithm 1: **NSGAII**

```
Input  : The number of generations $T$, the objective functions $f_i$
Output: The approximate Pareto front $\mathbf{P_{best}}$
begin
    Initialize the parent population $\mathbf{P_0}$.
    Create the population $\mathbf{Q_0}$ from $\mathbf{P_0}$.
    $\mathbf{P_{best}}$ = non-dominated solutions in $\mathbf{P_0} \cup \mathbf{Q_0}$.
    for $t = 1$ to $T$ do
        // Selection phase
        $\mathbf{R_t} = \mathbf{P}_t \cup \mathbf{Q}_t$
        $\mathbb{F} = FastNonDominatedSort(\mathbf{R_t})$
        $\mathbf{P_{t+1}} = \emptyset$
        $i = 1$
        while $|\mathbf{P_{t+1}}| + |\mathbb{F}_\mathbf{i}| \leq P$ do
            $\mathbf{P_{t+1}} = \mathbf{P_{t+1}} \cup \mathbb{F}_\mathbf{i}$
            $i = i + 1$
        end
        if $|\mathbf{P_{t+1}}| < P$ then
            Sort $\mathbb{F}_\mathbf{i}$ in the descending order of crowding distances.
            $\mathbf{P_{t+1}} = \mathbf{P_{t+1}} \cup \mathbb{F}_\mathbf{i}[1 : (P - |\mathbf{P_{t+1}}|)]$
        end
        // Variation phase
        Create the population $\mathbf{Q_{t+1}}$ from $\mathbf{P_{t+1}}$.
        Update $\mathbf{P_{best}}$ with non-dominated solutions in $\mathbf{Q_{t+1}}$.
    end
    return $\mathbf{P_{best}}$
end
```

population size, Λ is the number of objectives. In the case of alternative clustering with two objectives, the complexity of the *FastNonDominatedSort* procedure is $O(P^2)$. Because the combined population includes all solutions of the previous parent population $\mathbf{P}_t$ and the offspring population $\mathbf{Q}_t$, the non-dominated solutions found in previous generations are always kept in the next generations.

The algorithm sequentially adds the solutions of the first fronts $\mathbb{F}_i$ to the next parent population $\mathbf{P}_{t+1}$ if after adding $\mathbb{F}_i$, the size of $\mathbf{P}_{t+1}$ is still less than or equal to P. Otherwise, the remaining vacancies of $\mathbf{P}_{t+1}$ are filled by $P - |\mathbf{P}_{t+1}|$ solutions of $\mathbb{F}_i$ with the largest crowding distances. The crowding distance of a solution in a population is an estimate of the solution density around that solution. The crowding distance of a solution $\mathbf{p}$ is measured as the sum of the normalized distances of two solutions on the left and right side of that solution along each objective. As illustrated in Fig. 2, the crowding distance of the solution $\mathbf{p}$ is the sum of side lengths of the cuboid (shown with a dashed box). The larger the crowding distance of a solution is, the less the solution density surrounding that solution is. Therefore, adding the largest crowding distance points encourages the diversity of the next parent population $\mathbf{P}_{t+1}$. The parent population $\mathbf{P}_{t+1}$ is now used to create a new offspring population $\mathbf{Q}_{t+1}$ by the regular evolutionary operators like binary tournament selection, recombination, mutation. In order to create a new solution $\mathbf{p}$, **NSGAII** selects two parents $\mathbf{p}_1$ and $\mathbf{p}_2$ by the binary tournament selection and then applies the recombination operator on $\mathbf{p}_1$ and $\mathbf{p}_2$ to produce $\mathbf{p}$. With probability of α, a mutation operator can be applied on $\mathbf{p}$ to

Algorithm 2: FastNondominatedSort

```
Input  : A solution set $\mathbf{P}$
Output: Classified fronts $\mathbb{F}_\mathbf{i}$ of $\mathbf{P}$
begin
    for $\mathbf{p} \in \mathbf{P}$ do
        $\mathbf{S}_p = \emptyset$
        $n_p = 0$
        for $\mathbf{q} \in \mathbf{P}$ do
            if $\mathbf{p} \succ \mathbf{q}$ then
                $\mathbf{S}_p = \mathbf{S}_p \cup \{\mathbf{q}\}$
            else
                if $\mathbf{q} \succ \mathbf{p}$ then
                    $n_p = n_p + 1$
                end
            end
        end
        if $n_p = 0$ then
            $p_{rank} = 1$
            $\mathbb{F}_\mathbf{1} = \mathbb{F}_\mathbf{1} \cup \{\mathbf{p}\}$
        end
    end
    $i = 1$
    while $\mathbb{F}_\mathbf{i} \neq \emptyset$ do
        $\mathbf{Q} = \emptyset$
        for $\mathbf{p} \in \mathbb{F}_\mathbf{i}$ do
            for $\mathbf{q} \in \mathbf{S}_p$ do
                $n_q = n_q - 1$
                if $n_q = 0$ then
                    $q_{rank} = i + 1$
                    $\mathbf{Q} = \mathbf{Q} \cup \{\mathbf{q}\}$
                end
            end
        end
        $i = i + 1$
        $\mathbb{F}_\mathbf{i} = \mathbf{Q}$
    end
end
```

increase the perturbation level. Then, the set $\mathbf{P}_{best}$ of non-dominated solutions obtained so far is updated by the non-dominated solutions in $\mathbf{Q}_{t+1}$. The whole process is repeated for the next generations. The complexity of generating a new offspring population $\mathbf{Q}_{t+1}$ from its parent population $\mathbf{P}_{t+1}$ is $O(P\Omega)$ where Ω is the complexity of computing Λ objectives. In alternative clustering problem, we have two objectives, therefore, the total complexity of **NSGAII** is $O(T(P^2 + P\Omega))$ where T is the number of generations.

In each generation, the number of non-dominated solutions is bounded by the population size P. Therefore, when running **NSGAII** with T generations, the size of the result set $\mathbf{P}_{best}$ is bounded by PT. However, when the populations are moved towards the true Pareto

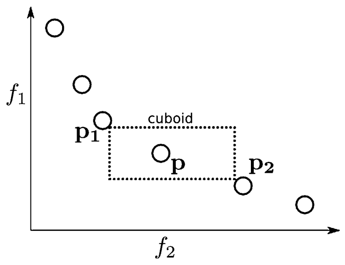


Fig. 2 Crowding distance

front, the solutions at generation t mostly dominate the solutions of previous generation $t-1$. Therefore, in practice, the size of $\mathbf{P}_{best}$ is around P and much smaller than PT. Only when **NSGAII** has converged at generation $t < T$, but it is still running for other $T-t$ generations, then the size of $\mathbf{P}_{best}$ can grow up larger than P because of very similar non-dominated solutions produced by **NSGAII** after converging.

The application of **NSGAII** to the alternative clustering problem requires the following choices:

- Two objective functions.
- A genetic encoding of clusterings.
- Recombination and mutation operators to generate a new offspring population from a parent population.
- An effective initialization scheme.

In the next sections, we present the above components.

3.2.2 Objective functions

We consider the Vector Quantization Error (VQE)—normally used in **K-Means** (Lloyd 1982)—for measuring the clustering quality, because the base clustering algorithm used in **AFDT** and **AFDT2** is also **K-Means**. This objective has been shown to be very robust for noisy datasets. The VQE of a clustering solution $\mathbf{C}_t$ is the sum of the square distances from each data object $\mathbf{x}_i$ to the centroid of the cluster $\mathbf{C}_t^k$ where $\mathbf{x}_i$ belongs to. The VQE function is:

$$VQE(\mathbf{C}_t) = \sum_{\mathbf{C}_t^k \in \mathbf{C}_t} \sum_{\mathbf{x}_i \in \mathbf{C}_t^k} \left\| \mathbf{x}_i - \boldsymbol{\mu}_t^k \right\|^2 \tag{4}$$

where $\mathbf{C}_t^k$ is a cluster in the clustering solution $\mathbf{C}_t$, $\boldsymbol{\mu}_t^k$ is the centroid of $\mathbf{C}_t^k$, and $\|\mathbf{x}_i - \boldsymbol{\mu}_t^k\|^2$ is the squared Euclidean distance between an item and its centroid. However, in the text datasets, the cosine distance is more suitable than the Euclidean distance. Therefore, when measuring the performance on the text datasets, we replace the Euclidean distance by the cosine distance. The Cosine VQE is:

$$CosineVQE(\mathbf{C}_t) = \sum_{\mathbf{C}_t^k \in \mathbf{C}_t} \sum_{\mathbf{x}_i \in \mathbf{C}_t^k} \left(1 - cosine\left(\mathbf{x}_i, \boldsymbol{\mu}_t^k\right)\right) \tag{5}$$

where $cosine(\mathbf{x}_i, \boldsymbol{\mu}_t^k)$ is the cosine similarity between $\mathbf{x}_i$ and $\boldsymbol{\mu}_t^k$. The smaller the VQE is, the better the quality of a clustering is. The cost of computing VQE for a clustering $\mathbf{C}_t$ is $O(ND)$ where N is the dataset size and D is the number of dimensions of data objects.

To measure the similarity between two clusterings, we use the popular Adjusted Rand Index (ARI) (Hubert and Arabie 1985). ARI is a normalized version of the Rand Index (RI) proposed by Rand (1971). The Rand Index $RI(\mathbf{C}_1, \mathbf{C}_2)$ between two clusterings $\mathbf{C}_1$ and $\mathbf{C}_2$ is simply defined as $\frac{n_{11}+n_{00}}{n_{11}+n_{10}+n_{01}+n_{00}}$ where n_{11} is the number of object pairs that are in the same cluster in both two clusterings; n_{00} is the number of pairs that are in different clusters in both clusterings; n_{10} is the number of pairs that are assigned in the same cluster by the clustering $\mathbf{C}_1$ and in different clusters by the clustering $\mathbf{C}_2$; n_{01} is the number of pairs that are assigned in different clusters by the clustering $\mathbf{C}_1$ and in the same cluster by the clustering $\mathbf{C}_2$. A problem with RI is that the expected value for two random clusterings is not constant. Hubert and Arabie (1985) fix this issue by introducing a normalized version of RI, called ARI. The ARI between two solutions $\mathbf{C}_1$ and $\mathbf{C}_2$ is defined as follows:

$$\begin{aligned} ARI(\mathbf{C}_1, \mathbf{C}_2) &= \frac{Index - ExpectedIndex}{MaxIndex - ExpectedIndex} \\ ARI(\mathbf{C}_1, \mathbf{C}_2) &= \frac{\sum_{ij}\binom{n_{ij}}{2} - [\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}]/\binom{n}{2}}{\frac{1}{2}[\sum_i \binom{n_{i.}}{2} + \sum_j \binom{n_{.j}}{2}] - [\sum_i \binom{n_{i.}}{2}\sum_j \binom{n_{.j}}{2}]/\binom{n}{2}} \end{aligned} \tag{6}$$

where n_{ij} is the number of common data objects of two clusters $\mathbf{X}_i$ and $\mathbf{X}_j$ produced by the clustering solutions $\mathbf{C}_1$ and $\mathbf{C}_2$, and $n_{i.} = \sum_j n_{ij}$, $n_{.j} = \sum_i n_{ij}$. The maximum value of ARI is 1 when two clusterings are identical. The value of ARI is around zero, or even a negative value, when two clusterings are very different. As we prefer different alternative clusterings, the smaller the ARI is, the better the dissimilarity between two clusterings is. In other words, we minimize the maximum similarity between the alternative clustering and the negative clustering set. Therefore, we define the similarity between an alternative clustering and a negative clustering set (similarly to the dissimilarity definition in Eq. (1)) as the maximum similarity between that alternative clustering and the clusterings in the negative clustering set. The complexity of computing ARI between two clusterings is $O(N)$ where N is the dataset size. Therefore, the total complexity of **COGNAC** when optimizing VQE and ARI is $O(T(P^2 + PND))$ where T is the number of generations and P is the population size. In other words, fixing the number of generations and the population size, the complexity of **COGNAC** increases *linearly* with the dataset size N and the number of dimensions D of data objects.

3.2.3 Genetic encoding of clusterings

We use the cluster-index based representation to encode clustering solutions. In detail, a clustering solution $\mathbf{C}_t$ of N data objects $\{\mathbf{x}_i\}_{i=1}^N$ is a N-dimensional vector where $\mathbf{C}_t(i)$ is the index of the cluster where the data object $\mathbf{x}_i$ belongs to. The index of each cluster is in the range of 1 to K with K is the fixed number of clusters. For example, with a dataset of 10 objects, and the number of clusters is 3, the clustering solution $\mathbf{C}_t = [1113331122]$ represents there clusters: $\mathbf{X}_1 = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_7, \mathbf{x}_8\}$, $\mathbf{X}_2 = \{\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6\}$, $\mathbf{X}_3 = \{\mathbf{x}_9, \mathbf{x}_{10}\}$.

3.2.4 Cluster-oriented recombination operator

Although the cluster-index encoding is popular in the literature, its main disadvantage is that the traditional recombination operators often do not produce offspring solutions which inherit good properties from their parents. The first problem is that the traditional recombination operators are performed on the *object level* whereas the clustering meaning is considered on the *cluster level*. In other words, the clusters are the smallest units containing information about the quality of the clustering solution to which they belong (Falkenauer 1994). Another drawback is that one clustering can be represented by many chromosomes, e.g. two chromosomes [123144] and [314322] represents the same solution of four clusters $\mathbf{C}^1 = \{\mathbf{x}_1, \mathbf{x}_4\}, \mathbf{C}^2 = \{\mathbf{x}_2\}, \mathbf{C}^3 = \{\mathbf{x}_3\}, \mathbf{C}^4 = \{\mathbf{x}_5, \mathbf{x}_6\}$. Therefore, performing recombination operators without a correct matching of clusters can return invalid solutions as in the following example:

$$\begin{array}{c} [1|\overline{23}|144] \\ [\overline{3}|14|\overline{322}] \\ \hline [3|23|322] \end{array}$$

The offspring [3|23|322] not only has an invalid number of clusters but also is very different from its parents. In this case, the offspring should be identical to their parents because they represent the same clustering solution.

To solve the above problems, we propose a *cluster-oriented* recombination operator where recombination is performed on clusters rather than on separate objects. The idea of performing recombination on clusters was first proposed by Falkenauer (1994) for the bin packing problem. However, their recombination operator cannot be applied to the clustering problem as it assumes special characteristics of the bin packing problem. In addition, their recombination does not perform a matching before merging clusters of two parents, therefore invalid solutions can still be returned.

The pseudo-code of our cluster-oriented recombination operator is presented in Algorithm 3. We first find a perfect matching $\mathbf{M}$ between clusters of two parents such that the number of common objects between them is largest. In this paper, we use the perfect matching algorithm proposed by Munkres (1957). The complexity of the matching algorithm is $O(K^4)$ (or $O(K^3)$ if optimized) where K is the number of clusters. Often, K is very small compared to the dataset size N, therefore the overhead of computing a perfect matching is relatively small.

Then, we perform uniform crossover on clusters as follows. First, we select a set $\mathbf{I}$ of $K/2$ random positions in $\{1, \dots, K\}$ to copy clusters $\mathbf{C}^i_{p_1}$ (where $i \in \mathbf{I}$) of the first parent $\mathbf{C}_{p_1}$ to the offspring $\mathbf{C}_o$. Let $\mathbf{U}$ be the set of all unassigned objects. Then, for each remaining position $i \in \{1, \dots, K\} \setminus \mathbf{I}$, we compute the set $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U}$ of unassigned objects in the cluster $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ of the second parent $\mathbf{C}_{p_2}$. If all objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ are assigned, it means that $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ is strictly included in some cluster of the first parent. Therefore, we move all objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$ to cluster i to avoid empty clusters. Otherwise, we simply assign the unassigned objects in $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U}$ to cluster i. After merging clusters from two parents, there are still unassigned (or orphan) objects. These orphan objects will be assigned to the clusters of one randomly chosen parent. We assign the orphan objects to the clusters of only one parent to preserve good characteristics from that parent.

An example of a dataset with 12 objects is in Fig. 3a. The number of clusters is 3. The clusters of two parents are as in Fig. 3b, 3c. The perfect matching $\mathbf{M}$ will match: $\mathbf{C}^1_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(1)=3}_{p_2}$, $\mathbf{C}^2_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(2)=1}_{p_2}$, $\mathbf{C}^3_{p_1} \rightarrow \mathbf{C}^{\mathbf{M}(3)=2}_{p_2}$ as in Fig. 3d. Assume that $\mathbf{I} = \{1\}$. We copy cluster $\mathbf{C}^1_{p_1}$ from $\mathbf{C}_{p_1}$, and move the unassigned objects in two clusters $\mathbf{C}^{\mathbf{M}(2)=1}_{p_2}$, $\mathbf{C}^{\mathbf{M}(3)=2}_{p_2}$ from $\mathbf{C}_{p_2}$ to the offspring, as in Fig. 3e. Then, we assign the orphan object 5 to the cluster $\mathbf{C}^2_o$ as in the first parent to obtain the offspring as in Fig. 3f. As it can be seen, the offspring inherits all good properties from its parents and converges to a correct clustering solution.

3.2.5 Neighbor-oriented mutation operator

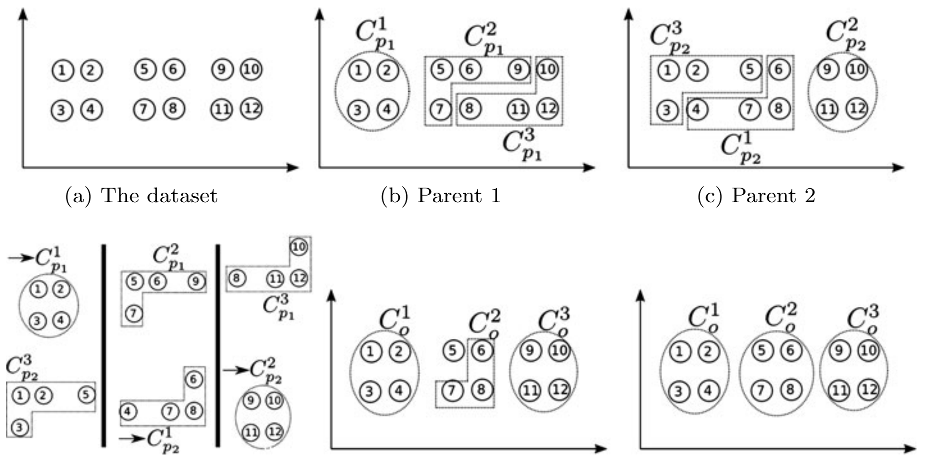
In the traditional mutation operators, usually some objects are selected and moved randomly to different clusters. However, moving an object $\mathbf{x}_i$ to a *random* cluster $\mathbf{C}^k$ can radically

Algorithm 3: Cluster-Oriented Recombination Operator

```
Input  : Two parent solutions $\mathbf{C}_{p_1}, \mathbf{C}_{p_2}$, the number of clusters $K$, the dataset $\mathbf{X}$.
Output: An offspring solution $\mathbf{C}_o$.
begin
    Initialize $\mathbf{C}_o$: $\forall i \in \{1, .., N\} : \mathbf{C}_o(i) = -1$.
    Let $\mathbf{C}^i_{p_j}$ be the i-th cluster of parent $\mathbf{C}_{p_j}$.
    Find a maximum perfect matching $\mathbf{M}$ where $\mathbf{C}^i_{p_1}$ is matched to $\mathbf{C}^{\mathbf{M}(i)}_{p_2}$.
    Let $\mathbf{I}$ be the set of $K/2$ indices selected randomly from $\{1, .., K\}$.
    // Perform uniform crossover on clusters.
    Copy clusters $\mathbf{C}^i_{p_1}$ where $i \in \mathbf{I}$ to the offspring: $\forall \mathbf{x}_t \in \mathbf{C}^i_{p_1} : \mathbf{C}_o(t) = i$.
    for $i \in \{1, .., K\} \setminus \mathbf{I}$ do
        $\mathbf{U} = \{\mathbf{x}_j : \mathbf{x}_j \in \mathbf{X} \wedge \mathbf{C}_o(j) = -1\}$.
        if $\mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U} = \emptyset$ then
            $\forall \mathbf{x}_t \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} : \mathbf{C}_o(t) = i$.
        else
            $\forall \mathbf{x}_t \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} \cap \mathbf{U} : \mathbf{C}_o(t) = i$.
        end
    end
    // Assign orphan objects.
    if rand() % 2 = 0 then
        for $i \in \{1, .., K\} \setminus \mathbf{I}$ do
            $\forall \mathbf{x}_j . \mathbf{x}_j \in \mathbf{C}^i_{p_1} \wedge \mathbf{C}_o(j) = -1 : \mathbf{C}_o(j) = i$.
        end
    else
        for $i \in \mathbf{I}$ do
            $\forall \mathbf{x}_j . \mathbf{x}_j \in \mathbf{C}^{\mathbf{M}(i)}_{p_2} \wedge \mathbf{C}_o(j) = -1 : \mathbf{C}_o(j) = i$.
        end
    end
    return $\mathbf{C}_o$
end
```

decrease the clustering quality when $\mathbf{x}_i$ is too far from $\mathbf{C}^k$. Also, if only few objects are moved to new clusters, the resulting perturbation can be too small for escaping local minima. But if many objects are moved to random clusters, the quality of the offspring can be strongly degraded. Therefore, determining the set of items to move is a difficult task. As it was the case for recombination, the traditional mutation operators can also produce invalid solutions when moving an object in a singleton cluster to a new cluster without checking the validity. To solve these problems, we replace the traditional mutation operators with a new operator called the *Neighbor-Oriented Mutation* operator.

The pseudo-code of the new mutation operator is presented in Algorithm 4. In detail, with the probability of ρ, each data object $\mathbf{x}_i$ in a cluster with size greater than 1 is moved to the cluster of one of its γ nearest neighbors $\mathbf{x}_j$. In other words, a proportion ρ of the data objects will be selected randomly and moved to the clusters of one of their nearest neighbors. Besides, we do not move objects of singleton clusters, therefore the validity of the offspring solution is guaranteed. Moving an item in this manner avoids assigning it to a very far cluster, therefore the search space is reduced significantly, but the clustering



(a) The dataset (b) Parent 1 (c) Parent 2

(d) Matching (e) The offspring after merging clusters. (f) The offspring after adding orphan objects.

Fig. 3 Cluster-oriented recombination operator example

solution is still of good quality. Besides, this operator can also produce arbitrarily-shaped clusters by linking near objects, e.g., a long cluster can be formed as a chain of near objects. Note that moving an object to one of the clusters of its nearest neighbors is different from moving that object to the nearest cluster: the first strategy allows arbitrary-shape clusters whereas the second one favors spherical clusters. To reduce the computational cost, the nearest neighbors of all objects will be computed only once before calling the neighbor-oriented mutation operator. For high-dimensional datasets, the distance between the objects can be computed by a suitable kernel. In this paper, we simply use the Euclidean distance.

The larger the value of ρ is, the more perturbed the offspring is. On the contrary, with small values of ρ, the search space is restricted, and as a result, the probability of remaining stuck in local minima increases. A similar issue also regards the number of nearest neighbors (parameter γ). A large value of γ allows moving an object $\mathbf{x}_i$ to far clusters and a small value of γ permits moving $\mathbf{x}_i$ only to near clusters. Setting γ too large results in random moves of data objects and thus wasting a lot of computing resources because very far objects can be assigned to the same cluster. But setting γ too small can limit the search space too much and keep the algorithm confined close to local minima.

To solve the above problems, we propose a method inspired by Simulated Annealing (Kirkpatrick et al. 1983). At the beginning, both parameters are assigned large values to allow the algorithm to explore potential solution regions. Then, these parameters are gradually decreased to help the algorithm exploit the most promising regions. This scheme automatically shifts in a gradual manner from diversification to intensification. In detail, in the first generation, ρ is assigned to ρ_{max} and then in the next generations, ρ is decreased sequentially by multiplying by a decay factor ρ_{dec} such that the value of ρ in the last generation is ρ_{min}. Mathematically, the probability ρ_t of moving an object at the t-th generation is computed as $\rho_t = \rho_{max}\rho_{dec}^t$ and $\rho_{max}\rho_{dec}^T = \rho_{min}$ where T is the number of generations. Formally, ρ_{dec} is computed based on ρ_{min} and ρ_{max} as follows:

$$\rho_{dec} = \sqrt[T]{\frac{\rho_{min}}{\rho_{max}}} \tag{7}$$

Algorithm 4: Neighbor-Oriented Mutation Operator

```
Input  : A solution $\mathbf{C}_o$, the perturbation probability $\rho$, the number of nearest
         neighbors $\gamma$.
Output: The perturbed solution $\mathbf{C}_o$.
begin
    for each object $\mathbf{x}_i \in \mathbf{X}$ do
        Let $\mathbf{X}_{nn}$ be the set of the first $\gamma$ nearest neighbors of the object $\mathbf{x}_i$.
        Pick randomly a nearest neighbor $\mathbf{x}_j \in \mathbf{X}_{nn}$.
        if the number of objects in the cluster of $\mathbf{x}_i$ is greater than 1 then
            With the probability of $\rho$, moving $\mathbf{x}_i$ to the cluster of $\mathbf{x}_j$ by assigning:
            $\mathbf{C}_o(i) = \mathbf{C}_o(j)$.
        end
    end
    return $\mathbf{C}_o$
end
```

Similarly, the number of nearest neighbours γ is first set to γ_{max} and then sequentially decreased by multiplying by a decay factor γ_{dec}. However, we only decrease γ in the first $T/2$ generations and keep γ as γ_{min} in the remaining generations to guarantee a large enough perturbation for the algorithm to escape from local minima. In detail, for the t-th generation where $t < T/2$, $\gamma_t = \gamma_{max}\gamma_{dec}^t$ where γ_{dec} is computed such that $\gamma_{max}\gamma_{dec}^{T/2} = \gamma_{min}$. In other words, γ_{dec} is computed as:

$$\gamma_{dec} = \sqrt[T/2]{\frac{\gamma_{min}}{\gamma_{max}}} \tag{8}$$

For the t-th generation where $t \geq T/2$, γ_t is set to γ_{min}. In the implementation, γ_t is a double variable and rounded to an integer by a built-in ceiling function when calling the *Neighbor-Oriented Mutation* operator.

3.2.6 Initialization

The initialization phase plays an important role in guiding the algorithm towards the true Pareto front. If the initial population contains only random solutions which are very different from the negative clusterings, then the algorithm explores well the region where the dissimilarity of the alternative clusterings and the negative ones is high. Analogously, if solutions similar to the negative clusterings are included into the initial set, then the algorithm often produces high-quality clusterings but similar to the negative ones. Here we assume that the negative clusterings are of high quality because they are usually obtained from single objective clustering algorithms. From this observation, we generate the initial population such that half of them are dissimilar clusterings and the rest are high-quality clusterings as follows.

Generating dissimilar clusterings: Let P be the initial population size and K_{neg} and K_{alter} be the number of clusters in negative clusterings and alternative clusterings, respectively. We generate $P/2$ dissimilar solutions from pairs of negative clusterings and individual negative clusterings as follows.

For each pair of two negative clustering solutions $\mathbf{C}_1$ and $\mathbf{C}_2$, we first find a perfect matching $\mathbf{M}$ between their clusters. Then, for each pair of matched clusters $\mathbf{C}_1^i \rightarrow \mathbf{C}_2^{\mathbf{M}(i)}$, we compute a common cluster $\mathbf{C}^i_{com} = \mathbf{C}^i_1 \cap \mathbf{C}^{\mathbf{M}(i)}_2$, and a xor cluster $\mathbf{C}^i_{xor} = (\mathbf{C}^i_1 \cup \mathbf{C}^{\mathbf{M}(i)}_2) \setminus \mathbf{C}^i_{com}$. Then we randomly merge two nearest common clusters or xor clusters until their total number equals K_{alter} to generate a dissimilar offspring $\mathbf{C}_o$. The distance between two common clusters or two xor clusters is computed as the Euclidean distance between their centroids. The offspring is very dissimilar from its parents because in their parents, the objects in two common or two xor clusters are in different clusters, but in the offspring they are placed into the same cluster. Note that we do not merge a common cluster and a xor cluster because they can reproduce a cluster which equals one of parents' clusters. If the number of clusters in two negative solutions are different, before matching, we sequentially split the largest cluster of the solution with the smaller number of clusters into two sub-clusters by **K-Means** until the number of clusters in two solutions are the same.

For individual negative clustering $\mathbf{C}_1$, we first extract its K_{neg} clusters $\{\mathbf{C}^i_1\}^{K_{neg}}_{i=1}$. Next, for each cluster $\mathbf{C}^i_1$, we use **K-Means** (Lloyd 1982) to partition this cluster into K_{alter} sub-clusters $\{\mathbf{C}^{ij}_1\}^{K_{alter}}_{j=1}$. The remaining objects in $\mathbf{X} \setminus \mathbf{C}^i_1$ are assigned to the j-th nearest sub-cluster $\mathbf{C}^{ij}_1$, with probability $\alpha^{-j} / \sum^{K}_{t=1} \alpha^{-t}$ to form a dissimilar offspring $\mathbf{C}_o$. The parameter α is a factor determining the perturbation level of the offspring solution. In other words, the probability of assigning an unassigned object to its $(j+1)$-th nearest sub-cluster is α times smaller than the probability of assigning that object to the j-th nearest sub-cluster. The smaller α is, the more perturbed the offspring is, therefore we vary α from $\alpha_{min} = 2$ to $\alpha_{max} = 10$ to generate a diverse set of dissimilar solutions. In detail, from each cluster $\mathbf{C}^i_1$ and a value $\alpha \in \{\alpha_{min}, \dots, \alpha_{max}\}$, we generate $\frac{P/2 - N_{\overline{\mathbb{C}}}(N_{\overline{\mathbb{C}}}-1)/2}{N_{\overline{\mathbb{C}}} K_{neg}(\alpha_{max}-\alpha_{min}+1)}$ dissimilar offspring where $N_{\overline{\mathbb{C}}}$ is the number of negative clusterings. The distance between an object and a sub-cluster is computed as the Euclidean distance between that object and the sub-cluster centroid. The offspring generated by the above strategy is very dissimilar to their parents because the objects in each cluster $\mathbf{C}^i_1$ of their parents $\mathbf{C}_1$ are now assigned to different clusters. This strategy is similar to the ensemble clustering algorithm proposed by Gondek and Hofmann (2005), but different because of the perturbation parameter α to diversify the offspring set.

Generating high-quality clustering: We generate $\frac{P/2}{N_{\overline{\mathbb{C}}}}$ high quality offspring from each negative clustering $\mathbf{C}_1$ as follows. First, we extract its K_{neg} clusters $\{\mathbf{C}^i_1\}^{K_{neg}}_{i=1}$. If $K_{neg} > K_{alter}$, we merge sequentially two nearest clusters until the number of clusters is K_{alter}. In the case where $K_{neg} < K_{alter}$, we split iteratively the largest cluster into two sub-clusters by **K-Means** until the number of clusters equals K_{alter}. Then, we compute the cluster centroids $\{\boldsymbol{\mu}^i_1\}^{K_{alter}}_{i=1}$ and assign each object to its i-th nearest centroid with the probability $\alpha^{-i} / \sum^{K_{alter}}_{t=1} \alpha^{-t}$ to obtain a new offspring. Similar to the procedure of generating dissimilar offspring, we also vary α from $\alpha_{min} = 2$ to $\alpha_{max} = 10$ for diversifying the high-quality offspring set.

3.3 Sequential generation of alternative clusterings

Based on the **COGNAC** algorithm (presented in Sect. 3.2), we propose the algorithm **SGAC** (the abbreviation of **S**equential **G**eneration of **A**lternative **C**lusterings) to generate a sequence of alternative clusterings as in Algorithm 5. First, the negative clustering set contains only the initial negative clustering and the alternative clustering set is empty. This initial negative clustering is often obtained from popular single objective clustering algorithms like **K-Means** (Lloyd 1982), **Hierarchical Clustering** (Lance and Williams 1967). Then in each iteration, the user will select one of the alternative clusterings returned by **COGNAC**. We defer the detailed discussion of the selection technique in Sect. 3.4. This alternative clustering is added to both sets of negative and alternative clusterings. Therefore,

Algorithm 5: **SGAC**

```
Input  : The initial negative clustering solution $\overline{\mathbf{C}}$, the number of alternative
         clusterings $M$
Output: The set of alternative clusterings
begin
    $\overline{\mathbf{S}} = \{\overline{\mathbf{C}}\}$
    $\mathbf{S}' = \emptyset$
    for $m = 1$ to $M$ do
        $\mathbf{S}^* = \mathbf{COGNAC}(\overline{\mathbf{S}})$
        The user selects one alternative clustering $\mathbf{C}'$ from $\mathbf{S}^*$.
        $\overline{\mathbf{S}} = \overline{\mathbf{S}} \cup \{\mathbf{C}'\}$
        $\mathbf{S}' = \mathbf{S}' \cup \{\mathbf{C}'\}$
    end
    return $\mathbf{S}'$
end
```

the alternative clustering generated in each next iteration is guaranteed to be different from the previous alternative clusterings. Finally, the set of all different alternative clusterings is returned to the user.

3.4 Analyzing the Pareto front

In order to reduce the number of solutions presented to users, we apply a traditional clustering algorithm like **K-Means** to partition the solution set into K clusters. Because the range of dissimilarity and quality are different, when clustering the solutions, we normalize their dissimilarity and quality as follows:

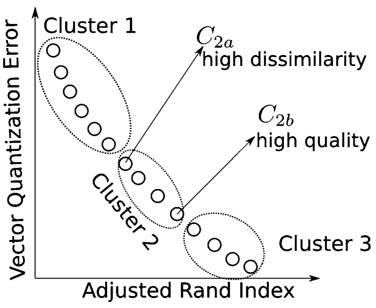
$$\mathbb{D}'(\mathbf{C}, \overline{\mathbf{S}}) = \frac{\mathbb{D}(\mathbf{C}, \overline{\mathbf{S}}) - \mu_{\mathbb{D}}}{\sigma_{\mathbb{D}}} \tag{9}$$

$$\mathbb{Q}'(\mathbf{C}) = \frac{\mathbb{Q}(\mathbf{C}) - \mu_{\mathbb{Q}}}{\sigma_{\mathbb{Q}}} \tag{10}$$

where $\mu_{\mathbb{D}}$, $\sigma_{\mathbb{D}}$, $\mu_{\mathbb{Q}}$, $\sigma_{\mathbb{Q}}$ are the mean and standard deviation of dissimilarity and quality of the solution set, respectively. For each cluster of solutions $\mathbf{S}_i$, the ranges of its dissimilarity and quality are represented in two border solutions: the one with the highest dissimilarity and lowest quality, and the other one with the highest quality and lowest dissimilarity. Therefore, users only need to consider the two border solutions of each cluster and quickly discard unsuitable clusters. If the user is satisfied with one of the border solutions, the algorithm can stop. Otherwise, the user selects a cluster of solutions with a reasonable range of dissimilarity and quality. Then, he can analyze that cluster more deeply by partitioning it again into sub-clusters and repeating the above process until a satisfactory solution is met.

Figure 4 shows an example of partitioning the Pareto solutions into 3 clusters. In cluster 2, the solution $\mathbf{C}_{2a}$ is the solution with the highest dissimilarity and lowest quality, and the solution $\mathbf{C}_{2b}$ is the solution with the highest quality and lowest dissimilarity. Assume that the range of dissimilarity and quality of two solutions $\mathbf{C}_{2a}$ and $\mathbf{C}_{2b}$ satisfies the users' requirement. If they satisfy with one of the two border solutions, they can stop the algorithm. Otherwise, if users want to have finer solutions, they can run a traditional clustering

Fig. 4 Analyzing the Pareto front with clustering



algorithm like **K-Means** (Lloyd 1982) to partition cluster 2 into three other sub-clusters and repeat the whole process.

Besides, when plotting all solutions, the figures are very difficult to read, therefore, we filter the similar solutions on the Pareto front as follows. First, we sort all solutions in the descending order of the dissimilarity objective. Then, we add the first solution with the highest dissimilarity to the filtered Pareto front $\mathbf{S}_{filtered}$. For each next solution $\mathbf{C}$, we compute the normalized difference on each objective between $\mathbf{C}$ and the previously added solution $\mathbf{C}'$. Denote $\mathbf{S}^*$ the Pareto solution set returned by **COGNAC**. The normalized difference (w.r.t a negative clustering set $\overline{\mathbf{S}}$) in dissimilarity Δ_D and in quality Δ_Q of two solutions $\mathbf{C}$ and $\mathbf{C}'$ are computed as:

$$\Delta_{\mathbb{D}}\big(\mathbf{C},\mathbf{C}'\big) = \frac{|\mathbb{D}(\mathbf{C},\overline{\mathbf{S}}) - \mathbb{D}(\mathbf{C}',\overline{\mathbf{S}})|}{\mathbb{D}_{max} - \mathbb{D}_{min}} \tag{11}$$

$$\mathbb{D}_{max} = \max_{\mathbf{C}\in\mathbf{S}^*} \mathbb{D}(\mathbf{C},\overline{\mathbf{S}}) \tag{12}$$

$$\mathbb{D}_{min} = \min_{\mathbf{C}\in\mathbf{S}^*} \mathbb{D}(\mathbf{C},\overline{\mathbf{S}}) \tag{13}$$

$$\Delta_{\mathbb{Q}}(\mathbf{C},\mathbf{C}') = \frac{|\mathbb{Q}(\mathbf{C}) - \mathbb{Q}(\mathbf{C}')|}{\mathbb{Q}_{max} - \mathbb{Q}_{min}} \tag{14}$$

$$\mathbb{Q}_{max} = \max_{\mathbf{C}\in\mathbf{S}^*} \mathbb{Q}(\mathbf{C}) \tag{15}$$

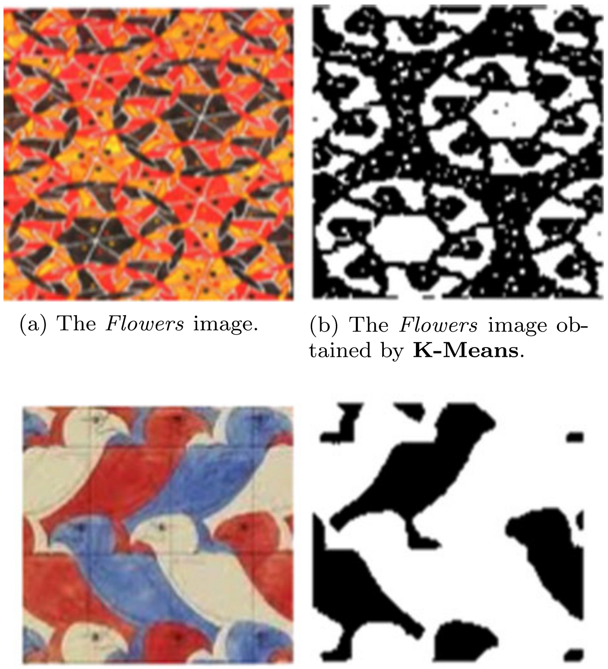
$$\mathbb{Q}_{min} = \min_{\mathbf{C}\in\mathbf{S}^*} \mathbb{Q}(\mathbf{C}) \tag{16}$$

If the normalized difference on two objectives $\Delta_{\mathbb{D}}(\mathbf{C},\mathbf{C}')$ and $\Delta_{\mathbb{Q}}(\mathbf{C},\mathbf{C}')$ between two solutions $\mathbf{C}$ and $\mathbf{C}'$ are equal to or greater than a threshold δ, then $\mathbf{C}$ is added to the filtered Pareto front $\mathbf{S}_{filtered}$. The above process is repeated until all solutions are considered. This technique can also be applied before partitioning the approximated Pareto front into K clusters to remove similar solutions.

4 Experiments

In this section, we describe the test datasets in Sect. 4.1 and the experiments to compare the performance of our algorithm with that of other state-of-the-art algorithms on the first alternative clustering in Sect. 4.2. Then, we illustrate the ability of our algorithm for generating a sequence of different alternative clusterings and compare the result with that of other two algorithms in Sect. 4.3. Finally, we analyze the parameter sensitivity of **COGNAC** in Sect. 4.4.

Fig. 5 Escher images



(a) The *Flowers* image.

(b) The *Flowers* image obtained by **K-Means**.

(c) The *Birds* image.

(d) The *Birds* image obtained by **K-Means**.

4.1 Datasets

In order to measure the performance of alternative clustering algorithms, we use four datasets with "ground truth" alternative clusterings. The first and second dataset are two Escher images with multiple interpretations to the human eye as in Fig. 5a, 5c. The *Flowers* and *Birds* images' size are 120×119 and 106×111, respectively. The RGB color space of each image is then converted in the L*a*b* color space (an important attribute of the L*a*b*-model is the device independence: the colors are defined independently of the device that they are displayed on). The difference in the color of two pixels can be computed as the Euclidean distance between their a* and b* values. The negative clustering of each image in Fig. 5b, 5d is obtained by running **K-Means** to partition each image into two clusters.

The third dataset is the *CMU Face* dataset on the UCI repository (Frank and Asuncion http://archive.ics.uci.edu/ml). We use all 624 face images of 20 people taken with varying pose (straight, left, right, up), expression (neutral, happy, sad, angry), eyes (wearing sunglasses or not). The size of each image is 960 (32×30). We then apply PCA as in Vinh and Epps (2010) to reduce the number of dimensions to 39. The labelling of each image by the name of the person in that image is used as the negative clustering of this dataset.

The fourth dataset is the *WebKB*[1] dataset. It contains HTML documents collected mainly from four universities: Cornell, Texas, Washington, Wisconsin, and classified under four groups: course, project, faculty, students. We select 500 features with highest information gain (conditioned on group names) to reduce the number of dimensions. Then, we remove

[1] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/.

Table 1 Dataset characteristics

Dataset	Cardinality	Number of dimensions	Number of clusters
CMUFaces	624	39	20
WebKB	1041	500	4
Birds	11766	2	2
Flowers	14280	2	2

Table 2 Parameter setting for **COGNAC**

Number of generations	200
Population size	200
Mutation rate	0.2
ρ_{max}	0.3
ρ_{min}	0.1
γ_{max}	40
γ_{min}	10

stop words, stemming, and use TF-IDF weighting to construct the feature vectors. The resulting dataset consists of 1041 documents. The labelling on four groups is used as the negative clustering.

Table 1 summarizes the datasets.

4.2 Comparisons on the first alternative clustering

We compare the performance of our algorithm **COGNAC** on the first alternative clustering with that of four other state-of-the-art algorithms, namely **COALA** (Bae and Bailey 2006), **AFDT** (Davidson and Qi 2008), **AFDT2** (Qi and Davidson 2009), **MinCEntropy**$^{++}$ (Vinh and Epps 2010) on four datasets in Sect. 4.1.

4.2.1 Experimental setup

The parameters of **COGNAC** are set as in Table 2. For a specific configuration, **COALA**, and **AFDT2** can only produce one solution. In order to generate a set of different solutions from **COALA**, the parameter declaring user reference on clustering quality and dissimilarity w is changed from 0.1 to 1.0 with step of 0.1. Large w values imply better clustering quality and smaller clustering dissimilarity. Similarly, the trade-off parameter a of **AFDT2** is changed from 1.0 to 2.8 with step of 0.2. Increasing a improves the clustering dissimilarity and decreases the clustering quality. The default values for w and a are set to 0.6 (Bae and Bailey 2006) and 2.0 (Qi and Davidson 2009), respectively. **AFDT** has no parameters and can only produce one solution. For **MinCEntropy**$^{++}$, we also vary its trade-off parameter m from 1 to 10 for generating a diverse set of solutions.

Except for the *CMUFaces* dataset where the number of clusters in alternative clusterings K_{alter} is set to 4, on other datasets, K_{alter} is set as the number of clusters in negative clusterings. Besides, on each dataset, **COGNAC**, **MinCEntropy**$^{++}$ and the base algorithm **K-Means** of **AFDT** and **AFDT2** are run 10 times to reduce the randomness effect. **COGNAC**[2]

[2]**COGNAC** source code is available on the authors' website.

and **COALA**[3] are implemented in C++ and Java, respectively. The other three algorithms **AFDT**, **AFDT2**[4] and **MinCEntropy**$^{++}$[5] are implemented in Matlab.

4.2.2 Experimental results

Figure 6 shows the performance of five algorithms on four datasets. In the figures, we also plot the negative clusterings to present the trade-off between clustering quality and dissimilarity. We denote the negative clusterings as **NC** in the plots. Besides, in order to keep the figures readable, we only plot some representative solutions on the Pareto front produced by **COGNAC** (by applying the filter procedure as in Sect. 3.4). On two large datasets *Birds* and *Flowers*, **COALA** cannot finish after 24 hours of CPU time.

As it can be observed, on most datasets our **COGNAC** provides diverse sets of high quality (in both clustering quality and dissimilarity) solutions. All solutions of **COALA**, **AFDT** and **AFDT2** are above the Pareto front of **COGNAC**. Thus, for each clustering solution produced by these three algorithms, there is always a solution produced by **COGNAC** of better quality in both objectives. Especially, on the *WebKB* and *Birds* datasets, our algorithm produces solutions of much higher quality. When comparing **COGNAC** and **MinCEntropy**$^{++}$, on the *Birds* dataset, the solution of **MinCEntropy**$^{++}$ on the left-up corner is outperformed significantly by other solutions of **COGNAC**. On the other datasets, the solutions of two algorithms slightly dominate each other. However, **COGNAC** provides a much more diverse set of solutions compared to that of **MinCEntropy**$^{++}$. In other words, increasing the trade-off parameter of **MinCEntropy**$^{++}$ with equal intervals does not guarantee to obtain a diverse set of solutions.

Analyzing the Pareto front: We also apply the analysis procedure in Sect. 3.4 to identify meaningful alternative clusterings returned by **COGNAC**. In all datasets, we partition the Pareto front into 5 groups and check the border solutions. On the *CMUFaces* dataset, we perform another step of partitioning on the last group. Please see the appendix for the plots of all border solutions. As for the other algorithms, we select the solutions returned when running them with their default parameters.

In order to see whether the alternative clusterings discovered by **COGNAC** are similar to the expected alternative clusterings, on the *CMUFaces* and *WebKB* dataset, we compute the ratio of *dominant* poses and university-based documents in each cluster, respectively. Table 3a and 3b show the cluster statistics of alternative clusterings of five algorithms on the *CMUFaces* and *WebKB* dataset. As it can be seen, with the negative clusterings of 20 people for the *CMUFaces* dataset, and of 4 groups {course, faculty, students, staff} for the *WebKB* dataset, **COGNAC** and **MinCEntropy**$^{++}$ produce the alternative clusterings matching closely to the expected alternative clusterings of the two datasets. On the *CMUFaces* dataset, **COALA** can only detects three alternative clusters {up, left, right}. Although **AFDT** and **AFDT2** can also discover all four alternative clusters of the *CMUFaces* dataset, their dominant ratios are much smaller than those of **COGNAC** and **MinCEntropy**$^{++}$. On the *WebKB* dataset, **COALA**, **MinCEntropy**$^{++}$, and **COGNAC** return the solutions which are very similar to the expected alternative clustering. In contrast, **AFDT** and **AFDT2** produce poor solutions in this case. Especially, **AFDT2** can only discover two alternative clusters.

As for the *Birds* and *Flowers* dataset, we select the images which are meaningful to human-eye interpretation from the border solution set of **COGNAC**. Figures 7f and 8f show

[3]http://ericbae.com/2011/05/11/clown-clustering-package/.

[4]http://wwwcsif.cs.ucdavis.edu/~qiz/code/code.html.

[5]https://sites.google.com/site/vinhnguyenx/publications.

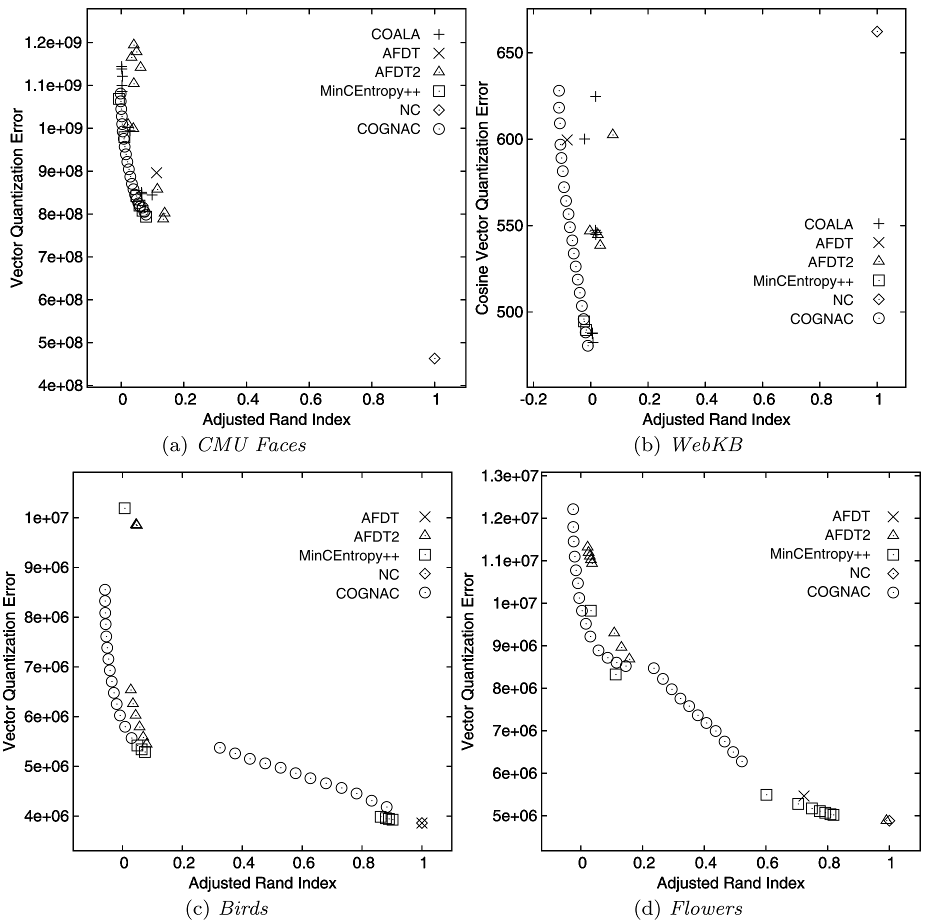


Fig. 6 Performance comparison of five algorithms on four datasets

these alternative clusterings. It can be seen that **COGNAC** has discovered successfully the high-quality alternative clusterings of these images. Although **AFDT2** also finds two meaningful solutions on these datasets, its solutions are much noisier than those of **COGNAC**. Besides, **AFDT** fails on both datasets as its solutions (in Fig. 7c and Fig. 8c) are very similar to the negative clusterings. Likewise, on the *Flowers* dataset, **minCEntropy**$^{++}$ also returns the alternative solution (in Fig. 8e) which is almost the same as the negative one.

4.3 Sequential generation of alternative clusterings

In this section, we use a synthetic dataset with multiple clusterings and the *Flowers* dataset to illustrate the effectiveness of **COGNAC** for generating a set of different alternative clusterings (by applying the **SGAC** procedure in Algorithm 5). We also compare our algorithm with **MinCEntropy**$^{++}$ and **Alter-Spect**.

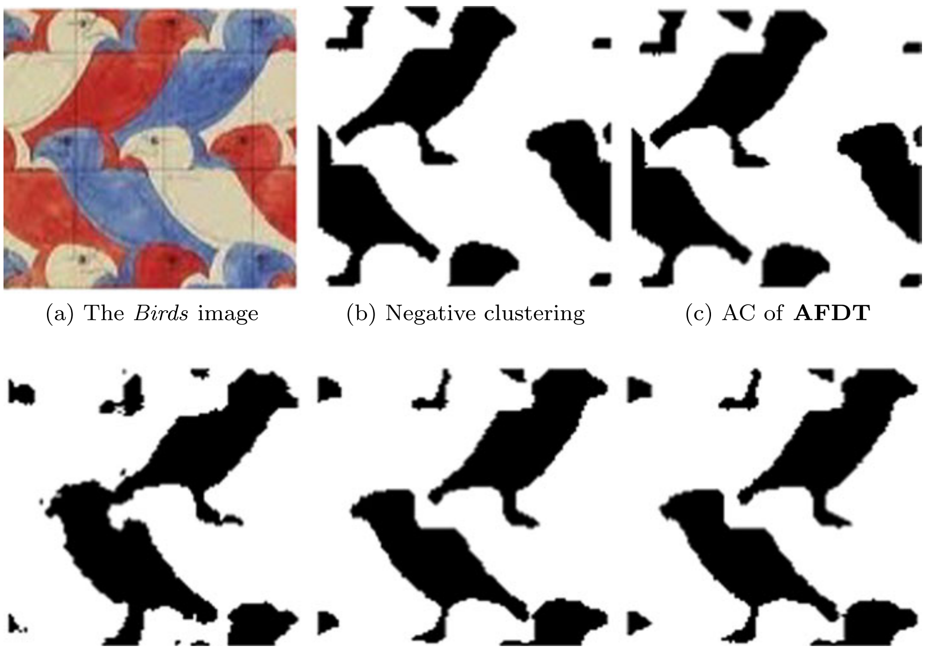
Table 3 Alternative clusterings on *CMUFaces* and *WebKB*

Algorithm	Cluster 1	Cluster 2	Cluster 3	Cluster 4
COALA	up (0.32)	left (0.87)	right (0.52)	right (0.58)
AFDT	up (0.30)	straight (0.34)	left (0.33)	right (0.30)
AFDT2	up (0.29)	straight (0.28)	left (0.32)	right (0.71)
MinCEntropy^{++}	up (0.54)	straight (0.75)	left (0.86)	right (0.89)
COGNAC	up (0.63)	straight (0.44)	left (0.70)	right (0.86)

(a) Alternative clusterings of five algorithms on *CMUFaces*

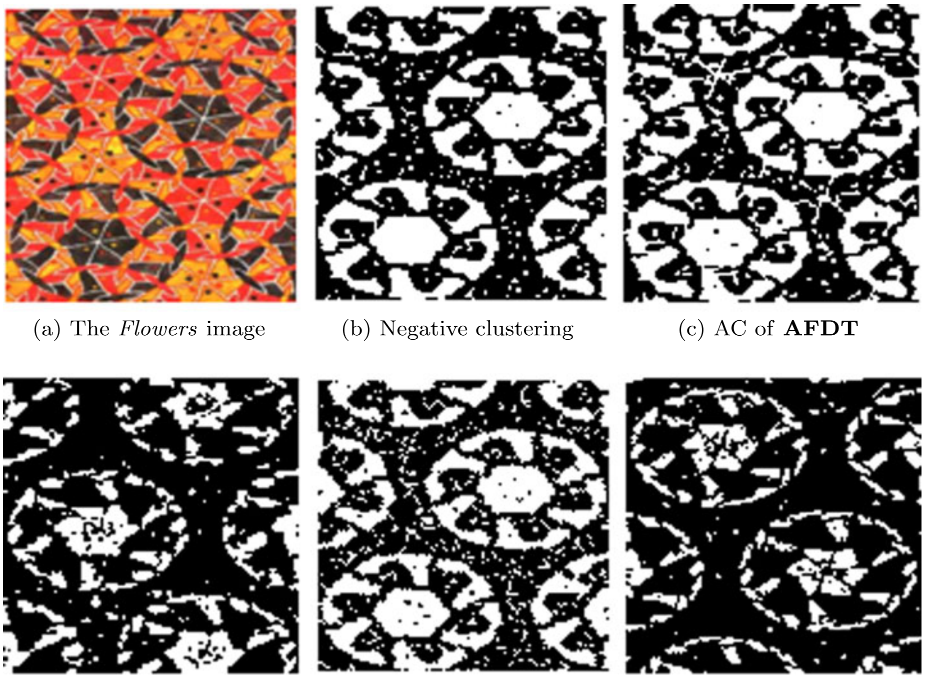
Algorithm	Cluster 1	Cluster 2	Cluster 3	Cluster 4
COALA	texas (0.85)	cornell (0.97)	wisconsin (0.93)	washington (0.94)
AFDT	texas (0.39)	texas (1.00)	cornell (0.98)	wisconsin (0.39)
AFDT2	wisconsin (0.50)	wisconsin (0.29)	wisconsin (0.67)	washington (1.00)
MinCEntropy^{++}	texas (1.00)	cornell (1.00)	wisconsin (0.69)	washington (1.00)
COGNAC	texas (0.98)	cornell (0.99)	wisconsin (0.82)	washington (0.97)

(b) Alternative clustering on WebKB


(a) The *Birds* image (b) Negative clustering (c) AC of **AFDT**

(d) AC of **AFDT2** (e) AC of **minCEntropy**$^{++}$ (f) AC of **COGNAC**

Fig. 7 The alternative clusterings (AC) of four algorithms on the *Birds* dataset


(a) The *Flowers* image (b) Negative clustering (c) AC of **AFDT**

(d) AC of **AFDT2** (e) AC of **minCEntropy**$^{++}$ (f) AC of **COGNAC**

Fig. 8 The alternative clusterings of four algorithms on the *Flowers* dataset

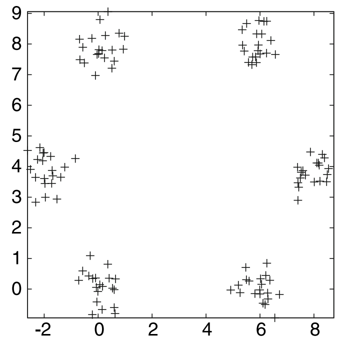
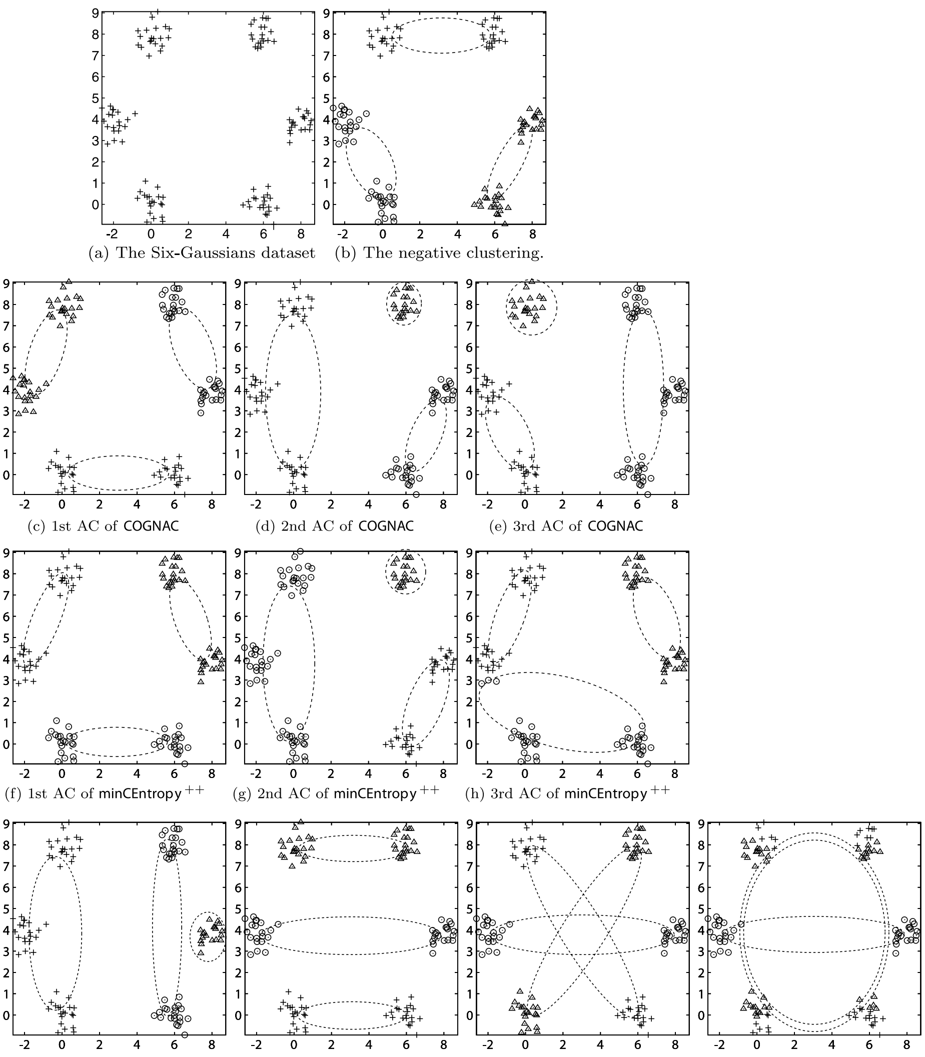


Fig. 9 The *Six-Gaussians* dataset

4.3.1 Experimental setup

The synthetic dataset consists of six Gaussian sub-clusters with the centroids at $\{(0, 0), (6, 0), (8, 4), (6, 8), (0, 8), (-2, 4)\}$ and the standard deviation of 0.5 for each coordinate. Each sub-clusters consists of 20 points as plotted in Fig. 9. Because this dataset has six natural sub-clusters, when setting the number of clusters to three, there are many possible ways to partition it. The parameters of **COGNAC** are set as in Table 2. For **MinCEntropy**$^{++}$, the parameter m (declaring that quality is m times important than dissimilarity) is set to its default value. We run both **COGNAC** and **MinCEntropy**$^{++}$ 10 times with different random



(a) The Six-Gaussians dataset (b) The negative clustering.

(c) 1st AC of COGNAC (d) 2nd AC of COGNAC (e) 3rd AC of COGNAC

(f) 1st AC of minCEntropy^{++} (g) 2nd AC of minCEntropy^{++} (h) 3rd AC of minCEntropy^{++}

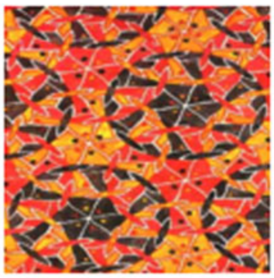
(i) 1st AC of Alter-Spect (j) 2nd AC of Alter-Spect (k) 3rd AC of Alter-Spect (l) 4th AC of Alter-Spect

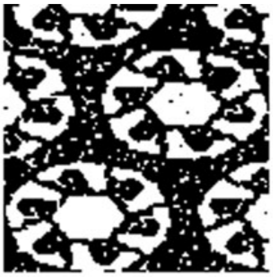
Fig. 10 Alternative clusterings (AC) of three algorithms on the *Six-Gaussians* dataset

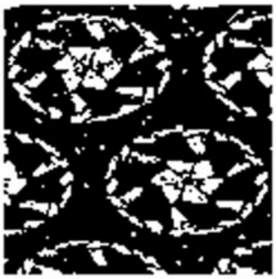
seeds and record the best results. We apply the analysis procedure in Sect. 3.4 to select the best alternative clustering solution.

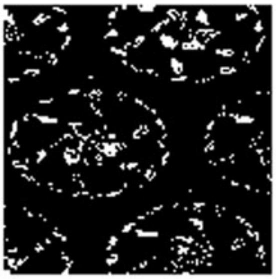
4.3.2 Experimental results

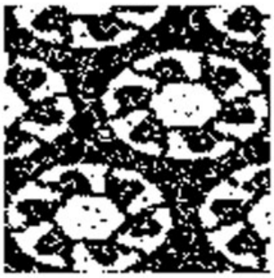
We run **COGNAC**, **MinCEntropy**$^{++}$ and **Alter-Spect** on the *Six-Gaussians* dataset with the initial negative clustering as in Fig. 10b. The sets of different alternative clusterings returned by three algorithms are shown in Fig. 10. We plot one more solution for **Alter-Spect** because its first solution can be very similar to the negative clustering. It can be seen that the alternative clusterings obtained by **COGNAC** are very different from each other and of high quality. On the contrary, **MinCEntropy**$^{++}$ can only generate the first two alternative clus-

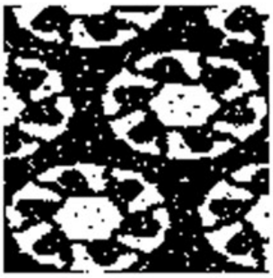

(a) The *Flowers* dataset.

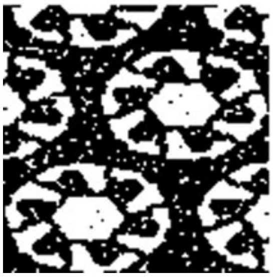

(b) The negative clustering.

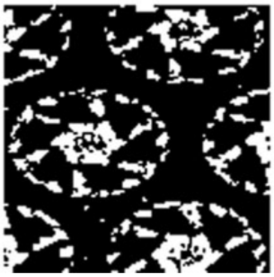

(c) 1st AC of **COGNAC**


(d) 2nd AC of **COGNAC**


(e) 1st AC of **minCEntropy**$^{++}$


(f) 2nd AC of **minCEntropy**$^{++}$


(g) 1st AC of **Alter-Spect**


(h) 2nd AC of **Alter-Spect**

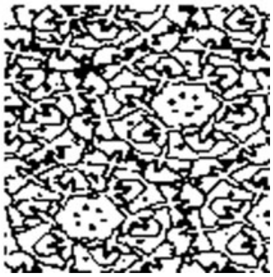
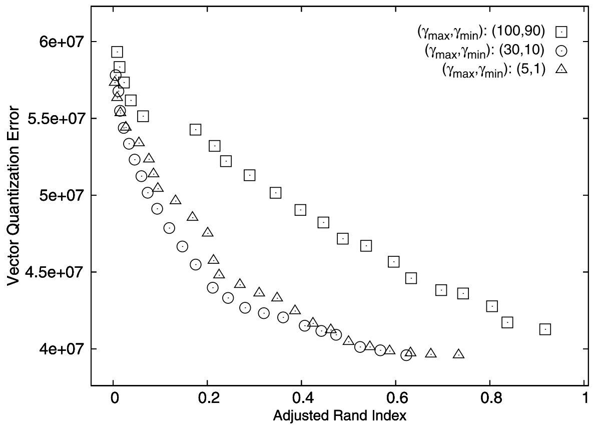
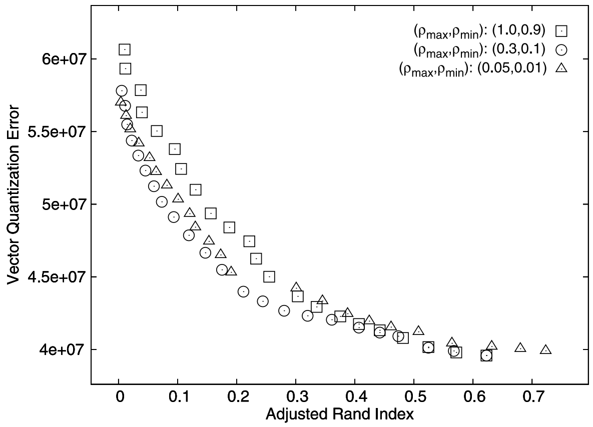

(i) 3rd AC of **Alter-Spect**

Fig. 11 Alternative clusterings of three algorithms on the *Flowers* dataset

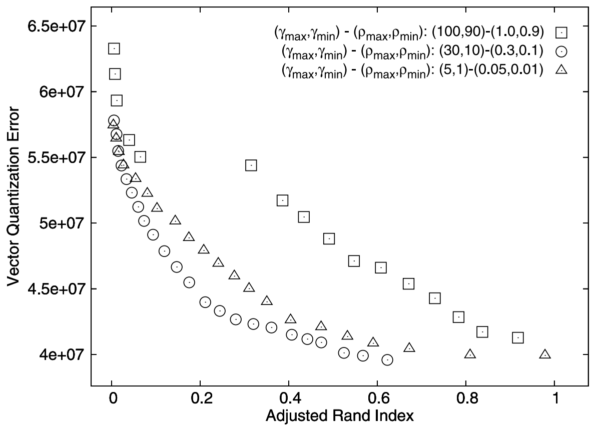
Fig. 12 **COGNAC** performance on the *CMUFaces* dataset



(a) **COGNAC** performance on the *CMUFaces* dataset with different configurations on the number of nearest neighbors.



(b) **COGNAC** performance on the *CMUFaces* dataset with different configurations on the perturbation probability.



(c) **COGNAC** performance on the *CMUFaces* dataset with different configurations.

terings in Fig. 10f, Fig. 10g. The third solution generated by **MinCEntropy**$^{++}$ in Fig. 10h is very similar to the first solution in Fig. 10f with some mistakes. Although **Alter-Spect** can also produce a set of alternative clusterings, these solutions are less meaningful to human interpretation because very far sub-clusters are grouped together, as in the third solution in Fig. 10k.

The results on the *Flowers* dataset are plotted in Fig. 11. **COGNAC** successfully discovers the other two alternative clusterings (with red and yellow colors) as depicted in Fig. 11c, 11d. On the contrary, **minCEntropy**$^{++}$ produces alternative clusterings which are very similar to the negative clustering, as plotted in Fig. 11e, 11f. As for **Alter-Spect**, only its second solution in Fig. 11h is meaningful because the other two solutions are very similar to the negative clustering.

4.4 Parameter sensitivity analysis

In this section, we perform three experiments to study the sensitivity of **COGNAC** to parameters $\rho_{max}, \rho_{min}, \gamma_{max}, \gamma_{min}$. In all experiments, we fix the number of generations to 100 and run the algorithm on the reduced *CMUFaces* dataset containing images of four people (an2i, at33, boland, ch4f). Besides, there is no parameter in the *Cluster-Oriented Recombination* operator, therefore we only use the *Neighbor-Oriented Mutation* operator to modify the solutions,

We first fix the perturbation probability (ρ_{max}, ρ_{min}) to $(0.3, 0.1)$ and set the maximum and minimum number of nearest neighbors $(\gamma_{max}, \gamma_{min})$ to $(100, 90)$, $(30, 10)$, $(5, 1)$, respectively. The first setting with large values can be considered as the representative for the traditional mutations, where an object can be assigned to arbitrary (or very far) clusters. As it can be seen in Fig. 12a, setting $(\gamma_{max}, \gamma_{min})$ to very large value like $(100, 90)$ leads to the poorest performance. Because in this case, **COGNAC** can put very far objects to the same cluster but according to local structures of the dataset, the near objects are often partitioned in the same cluster. However, setting $(\gamma_{max}, \gamma_{min})$ to very small value like $(5, 1)$ can make the algorithm stuck at local minima, hence a moderate setting of $(\gamma_{max}, \gamma_{min})$ like $(30, 10)$ results in the best performance.

To study the sensitivity to the perturbation probability, we fix the number of nearest neighbors $(\gamma_{max}, \gamma_{min})$ to $(30, 10)$, and set the perturbation probability (ρ_{max}, ρ_{min}) to $(1.0, 0.9)$, $(0.3, 0.1)$, and $(0.05, 0.01)$. Figure 12b shows that setting (ρ_{max}, ρ_{min}) to large values like $(1.0, 0.9)$ results in the poorest performance because large perturbation levels can destroy the good properties of current objects. However, small perturbation levels are not enough for **COGNAC** to escape from local minima, therefore similarly to the case of the number of nearest neighbors $(\gamma_{max}, \gamma_{min})$, a moderate setting often results in the best performance as presented in Fig. 12b.

Finally, we compare three configurations of $\{(\gamma_{max}, \gamma_{min}) - (\rho_{max}, \rho_{min})\}$ in the decreasing order of perturbation level which are: $\{(100, 90) - (1.0, 0.9)\}$, $\{(30, 10) - (0.3, 0.1)\}$, $\{(5, 1) - (0.05, 0.01)\}$. The middle configuration $\{(30, 10) - (0.3, 0.1)\}$ outperforms the other configurations, as shown in Fig. 12c.

5 Conclusion

In this paper, we proposed an explicit multi-objective algorithm for alternative clustering, called **COGNAC** and a derived algorithm called **SGAC** for the sequential generation of alternative clusterings. **COGNAC** and **SGAC** not only provide solutions outperforming those produced by other state-of-the-art algorithms, but they also possess attractive features. Firstly, they are very flexible and they can accept arbitrary objectives, therefore they can be used as a baseline when comparing with different alternative clustering algorithms. In addition, **SGAC** can be used to generate a *sequence* of alternative clusterings by adding the newly found alternative clustering to the negative clustering set and re-running **COGNAC**. Each newly generated clustering is guaranteed to be different from the previously found ones. Finally, **COGNAC** approximates the whole Pareto front in a single run, but its complexity only scales up linearly with the dataset size, when deployed with two objectives VQE and ARI.

Currently, **COGNAC** simply returns the whole Pareto front to the user. Then, the user applies some techniques proposed in this paper to analyze and visualize the obtained solution set. Although judging the best solution from the Pareto front is a responsibility of the users, helping users to explore the Pareto front *interactively* in a more efficient manner is an interesting and non-trivial task. In the future, we plan to integrate interactive techniques in multi-objective optimization (Branke et al. 2008; Battiti and Passerini 2010) with our algorithm so that users can quickly direct the search to explore the regions of their interest.

Appendix: Analysis of the Pareto front

In this section, we plot all border solutions of **COGNAC** on four datasets by applying the analysis procedure in Sect. 3.4. We arrange the solutions in the ascending order of dissimilarity.

Table 4a and 4b show the 10 border solutions of the first partitioning and 20 border solutions of the second partitioning (on the last group) on the *CMUFaces* dataset, respectively. Table 5 presents the 10 border solutions of the first partitioning on the *WebKB* dataset. Figure 13 and Fig. 14 depict the 10 border solutions of the first partitioning on the *Birds* and *Flowers* dataset. As it can be seen, the first alternative solution is very similar to the negative solution, but then the dissimilarity is increased gradually, and at some point, the solution is transformed into a completely different one. Besides, as the border solution with the highest dissimilarity of a group and the border solution with the highest quality of the next group are very close to each other, therefore their results are very similar.

Table 4 Alternative clusterings (in two partitionings) of **COGNAC** on the *CMUFaces* dataset

Cluster 1	Cluster 2	Cluster 3	Cluster 4
up (0.26)	straight (0.39)	left (0.77)	right (0.93)
up (0.35)	straight (0.31)	left (0.88)	right (0.86)
up (0.35)	straight (0.32)	left (0.80)	right (0.87)
straight (0.34)	left (0.77)	left (0.41)	right (0.84)
straight (0.34)	left (0.77)	left (0.44)	right (0.83)
straight (0.33)	left (0.63)	right (0.89)	right (0.63)
straight (0.33)	left (0.65)	right (0.89)	right (0.59)
straight (0.41)	left (0.62)	left (0.70)	right (0.84)
straight (0.41)	left (0.67)	left (0.71)	right (0.85)
straight (0.41)	left (1.00)	left (0.70)	right (0.86)

10 alternative clusterings in the first partitioning

Cluster 1	Cluster 2	Cluster 3	Cluster 4
straight (0.41)	left (0.67)	left (0.71)	right (0.85)
straight (0.41)	left (0.80)	left (0.70)	right (0.85)
straight (0.41)	left (0.70)	left (0.77)	right (0.86)
straight (0.41)	left (0.89)	left (0.71)	right (0.86)
straight (0.41)	left (0.92)	left (0.70)	right (0.85)
straight (0.40)	left (0.94)	left (0.70)	right (0.86)
straight (0.41)	left (0.94)	left (0.70)	right (0.86)
straight (0.41)	left (0.70)	left (0.97)	right (0.86)
straight (0.40)	left (0.97)	left (0.70)	right (0.86)
straight (0.41)	left (0.64)	left (1.00)	right (0.86)
straight (0.38)	left (0.91)	left (0.72)	right (0.88)
up (0.63)	**straight (0.44)**	**left (0.70)**	**right (0.86)**
up (0.63)	straight (0.44)	left (0.70)	right (0.86)
up (0.36)	straight (0.42)	left (0.70)	right (0.86)
straight (0.42)	straight (0.35)	left (0.70)	right (0.87)
up (0.39)	straight (0.42)	left (0.70)	right (0.86)
up (0.41)	straight (0.42)	left (0.70)	right (0.86)
straight (0.42)	left (0.43)	left (0.69)	right (0.86)
up (0.42)	straight (0.41)	left (0.70)	right (0.86)
straight (0.41)	left (1.00)	left (0.70)	right (0.86)

20 alternative clusterings in the second partitioning on the last group

Table 5 10 alternative clusterings (in the first partitioning) of **COGNAC** on the *WebKB* dataset

Cluster 1	Cluster 2	Cluster 3	Cluster 4
texas (0.98)	**cornell (0.99)**	**wisconsin (0.82)**	**washington (0.97)**
texas (1.00)	cornell (1.00)	wisconsin (0.65)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.64)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.57)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.56)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.51)	washington (0.99)
texas (1.00)	cornell (1.00)	wisconsin (0.50)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.45)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.44)	washington (1.00)
texas (1.00)	cornell (1.00)	wisconsin (0.37)	washington (1.00)

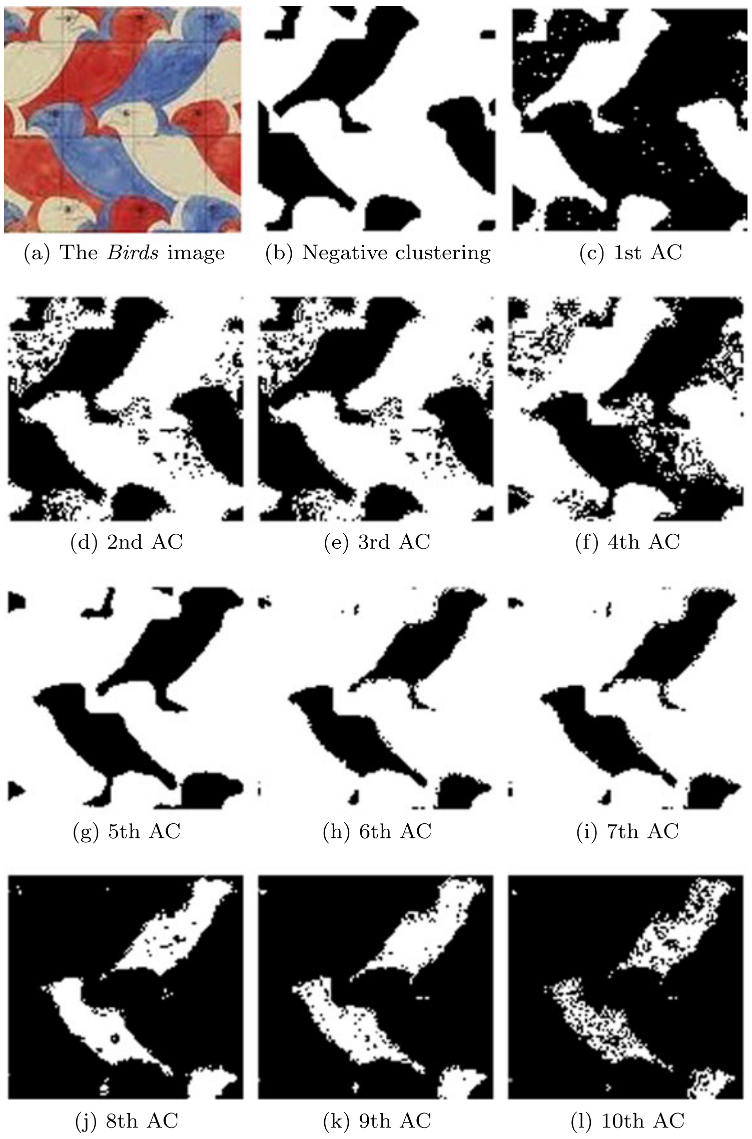


Fig. 13 10 alternative clusterings (AC) of **COGNAC** in the first partitioning on the *Birds* dataset

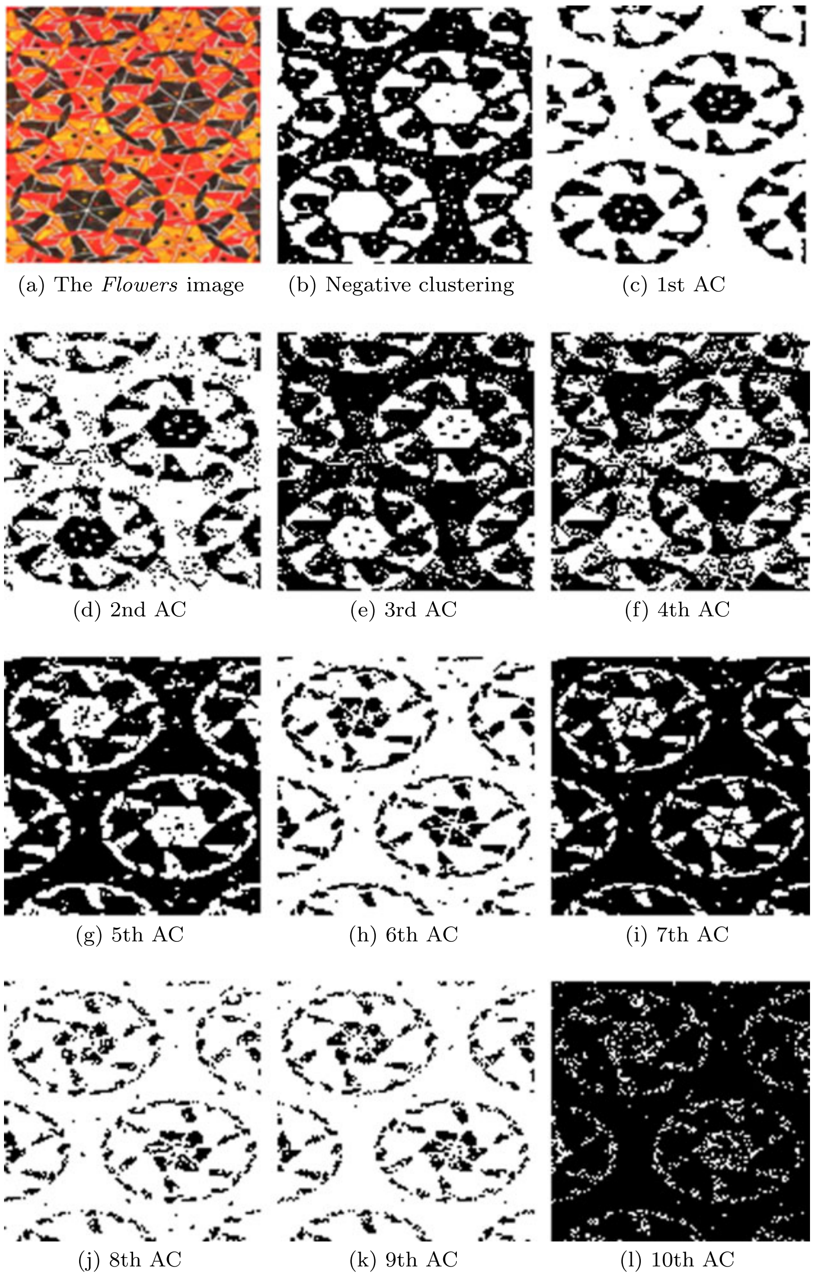


Fig. 14 10 alternative clusterings (AC) of **COGNAC** in the first partitioning on the *Flowers* dataset

References

Bae, E., & Bailey, J. (2006). Coala: a novel approach for the extraction of an alternate clustering of high quality and high dissimilarity. In *Sixth international conference on data mining, 2006. ICDM'06* (pp. 53–62). New York: IEEE.

Battiti, R., & Passerini, A. (2010). Brain–computer evolutionary multiobjective optimization: a genetic algorithm adapting to the decision maker. *IEEE Transactions on Evolutionary Computation*, *14*(5), 671–687.

Battiti, R., Brunato, M., & Mascia, F. (2008). Reactive search and intelligent optimization. In *Operations research/computer science interfaces*. Berlin: Springer. ISBN: 978-0-387-09623-0.

Branke, J., Deb, K., & Miettinen, K. (2008). *Multiobjective optimization: interactive and evolutionary approaches* (Vol. 5252). New York: Springer.

Cui, Y., Fern, X. Z., & Dy, J. G. (2007). Non-redundant multi-view clustering via orthogonalization. In *Seventh IEEE international conference on data mining, 2007. ICDM 2007* (pp. 133–142). New York: IEEE.

Dang, X. H., & Bailey, J. (2010). Generation of alternative clusterings using the Cami approach. In *The SIAM international conference on data mining* (pp. 118–129).

Dasgupta, S., & Ng, V. (2010). Mining clustering dimensions. In *Proceedings of the 27th international conference on machine learning* (pp. 263–270).

Davidson, I., & Qi, Z. (2008). Finding alternative clusterings using constraints. In *The eighth IEEE international conference on data mining* (pp. 773–778). New York: IEEE.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II-ii. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197.

De Bie, T. (2011). Subjectively interesting alternative clusters. In *Proceedings of the 2nd MultiClust workshop: discovering, summarizing, and using multiple clusterings. Athens, Greece: CEUR workshop proceedings (CEUR-WS. org)* (online) (pp. 43–54).

Falkenauer, E. (1994). A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary Computation*, *2*(2), 123–144.

Frank, A., & Asuncion, A. *UCI machine learning repository*. URL http://archive.ics.uci.edu/ml.

Gondek, D., & Hofmann, T. (2003). Conditional information bottleneck clustering. In *3rd IEEE international conference on data mining, workshop on clustering large data sets* (pp. 36–42). Princeton: Citeseer.

Gondek, D., & Hofmann, T. (2005). Non-redundant clustering with conditional ensembles. In *Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining* (pp. 70–77). New York: ACM.

Günnemann, S., Färber, I., & Seidl, T. (2012). Multi-view clustering using mixture models in subspace projections. In *Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 132–140). New York: ACM.

Handl, J., & Knowles, J. (2007). An evolutionary approach to multiobjective clustering. *IEEE Transactions on Evolutionary Computation*, *11*(1), 56–76.

Hruschka, E. R., Campello, R. J. G. B., Freitas, A. A., & de Carvalho, A. C. P. L. F. (2009). A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *39*(2), 133–155.

Hubert, L., & Arabie, P. (1985). Comparing partitions. *Journal of Classification*, *2*(1), 193–218.

Jain, P., Meka, R., & Dhillon, I. S. (2008). Simultaneous unsupervised learning of disparate clusterings. *Statistical Analysis and Data Mining*, *1*(3), 195–210.

Kirkpatrick, S., Gelatt, C. D. Jr, & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Lance, G. N., & Williams, W. T. (1967). A general theory of classificatory sorting strategies II. Clustering systems. *The Computer Journal*, *10*(3), 271–277.

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE Transactions on Information Theory*, *28*, 129–137.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, *5*(1), 32–38.

Niu, D., Dy, J. G., & Jordan, M. I. (2010). Multiple non-redundant spectral clustering views. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 831–838).

Qi, Z. J., & Davidson, I. (2009). A principled and flexible framework for finding alternative clusterings. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 717–726). New York: ACM.

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 846–850.

Tishby, N., Pereira, F. C., & Bialek, W. (2000). *The information bottleneck method*. arXiv:physics/0004057.

Truong, D. T., & Battiti, R. (2012). A cluster-oriented genetic algorithm for alternative clustering. In *Proceedings of the 12th international conference on data mining (ICDM12)* (pp. 1122–1127). New York: IEEE.

Vinh, N. X., & Epps, J. (2010). Mincentropy: a novel information theoretic approach for the generation of alternative clusterings. In *Proceedings of the 10th international conference on data mining (ICDM10)* (pp. 521–530). New York: IEEE.