Consensus Clustering Algorithms: Comparison and Refinement

Andrey Goder* Vladimir Filkov†

Computer Science Department
University of California
Davis, CA 95616

Abstract

Consensus clustering is the problem of reconciling clustering information about the same data set coming from different sources or from different runs of the same algorithm. Cast as an optimization problem, consensus clustering is known as *median partition*, and has been shown to be NP-complete. A number of heuristics have been proposed as approximate solutions, some with performance guarantees. In practice, the problem is apparently easy to approximate, but guidance is necessary as to which heuristic to use depending on the number of elements and clusterings given. We have implemented a number of heuristics for the consensus clustering problem, and here we compare their performance, independent of data size, in terms of efficacy and efficiency, on both simulated and real data sets. We find that based on the underlying algorithms and their behavior in practice the heuristics can be categorized into two distinct groups, with ramification as to which one to use in a given situation, and that a hybrid solution is the best bet in general. We have also developed a refined consensus clustering heuristic for the occasions when the given clusterings may be too disparate, and their consensus may not be representative of any one of them, and we show that in practice the refined consensus clusterings can be much superior to the general consensus clustering.

1 Introduction

Clustering is a very useful technique for mining large data sets since it organizes the data, based on similarity, into smaller groups which are easier to handle in a downstream analysis. Often, different clusterings of the same data can be obtained either from different experimental sources or from multiple runs of non-deterministic clustering algorithms. *Consensus clustering* formalizes the idea that combining different clusterings into a single representative, or consensus, would emphasize the common organization in the different data sets and reveal the significant differences between them. The goal of consensus clustering is to find a consensus which would be representative of the given clusterings of the same data set.

Multiple clusterings of the same data arise in many situations; here we mention two classes of instances. The first class is when different attributes of large data sets yield different clusterings of the entities. For example, high-throughput experiments in molecular biology and genetics often yield data, like gene expression and protein-protein interactions, which provide a lot of useful information about different aspects of the same entities, in this case genes or proteins. Specifically, in gene expression data different experimental conditions can be used as attributes and they can yield different clusterings of the genes. In addition to the individual value of each experiment, combining the data across multiple experiments could potentially reveal different aspects of the genomic system and even its systemic properties [1]. One useful way of combining the data from different experiments is to aggregate their clusterings into a consensus or representative clustering which may both increase the confidence in the common features in all the data sets but also tease out the important differences among them [2, 3].

A second class of instances results from situations where multiple runs of the same non-deterministic clustering or data mining algorithms yield multiple clusterings of the same entities. Non-deterministic clustering algorithms, e.g. K-means, are sensitive to the choice of the initial seed clusters; running K-means with different seeds may yield very different results. This is in fact desirable when the data is non-linearly separable, as the multiple *weak* clusterings could then be combined into a stronger one [4]. To address this, it is becoming more and more common to analyze jointly the resulting clusterings from a number of K-means runs, seeded with different initial centers. One way to aggregate all those clusterings is to compute a consensus among them, which would be more robust to the initial conditions.

*andrey.goder@gmail.com
†filkov@cs.ucdavis.edu

1.1 Consensus Clustering Formalization: the Median Partition Problem The version of the consensus clustering problem in which we are interested is the following: given a number of clusterings of some set of elements, find a clustering of those elements that is as close as possible to all the given clusterings. To formalize this problem we need two things: a representation of a clustering of a set, and a measure $d(,)$ of distance between clusterings.

We identify a clustering of a set A, $|A| = n$, with a set partition π of A, such that if two elements a, b are clustered together in the clustering of A then they are in the same cluster, or block, in π.

A natural class of distances between set partitions can be defined by counting the clustering agreements between them for each pair of elements in the set. Given two set-partitions π_1 and π_2 let a equal the number of pairs of elements co-clustered in both partitions, b equal the number of pairs of elements co-clustered in π_1, but not in π_2, c equal the number of pairs co-clustered in π_2, but not in π_1, and d the number of pairs not co-clustered in either partition (in other words a and d count the number of clustering agreements in both partitions, while b and c count the disagreements). As our distance measure we chose the *symmetric difference distance* (sdd), defined as

$$d(\pi_1, \pi_2) = b + c = \binom{n}{2} - (a + d).$$

This distance measure is a metric and has a nice property that it is computable in linear time (with respect to the number of elements n) [5, 2], which is one of the reasons why we chose it. Another reason is that it can be updated very fast following one element moves in set partitions, as related to the local search heuristics [2], which are described later.

It is of note also that the sdd is related to the popular *Rand Index*, which is defined as $R = (a+d)/\binom{n}{2}$. The sdd and the Rand are not corrected for chance, i.e. the distance between two independent set partitions is non-zero on the average, and is dependent on n. A version corrected for chance, known as the *adjusted Rand Index*, as well as other pair-counting measures (e.g. *Jaccard Index*) exist [6, 7]. The adjusted Rand and the Jaccard indexes are given by complex formulas, and although they can also be computed in linear time, we are not aware of fast update schemes for them. We will use the adjusted Rand in Sec. 5 where the algorithms' complexity is not an issue. In addition, several of the consensus clustering heuristics we use run efficiently only with the symmetric difference distance.

Now we can state the formal version of consensus clustering, known as the *median partition* (MP) problem. Given a set of k set partitions, $\{\pi_1, ..., \pi_k\}$, and $d(,)$ the symmetric difference distance metric, we want to find a set partition π^* such that

$$\pi^* = \operatorname*{argmin}_{\pi} \sum_{i=1}^{k} d(\pi_i, \pi). \tag{1.1}$$

The median partition problem with the symmetric difference distance is known to be NP-complete [8]. For more on the history of the problem and variants see [2].

1.2 Prior work A number of algorithms have been published for approximating the median partition problem [3, 9]. Various theoretical results, including performance guarantees have been derived, although the gap between the performance of these heuristics and their upper bounds is wide [3, 9, 10].

The existing algorithms can be naturally grouped into three groups: the first consists of only one algorithm, the simplest non-trivial heuristic for MP: choose one of the input partitions as the consensus. The second group is comprised of clustering-based algorithms [10], and the third of local-search heuristics [3]. Ailon et al. [10] and Filkov and Skiena [2] provide some theoretical upper bounds as well as performance comparisons for some of those heuristics. Bertolacci and Wirth [9] give a more extensive comparison, while only focusing on clustering-based algorithms and using sampling of the data for better performance.

1.3 Our Contributions In this paper we compare a comprehensive collection of existing median partition heuristics and provide definitive comparisons of their performance in variety of situations. In particular, our contributions are:

- we have written a C++ library for set partitions, supporting many common operations. We have implemented six heuristics that find approximate solutions to the median partition problem: Best Of K, Best One Element Moves, Majority Rule, Average Link, CC Pivot, and Simulated Annealing, with several variations of each. The code is publicly available from *http://www.cs.ucdavis.edu/∼filkov/software/conpas*.
- We provide a statistical approach to evaluating the algorithms' performance by standardizing the distribution of the sum of distances between given partitions and a consensus.
- We assess the efficacy of the six algorithms on both real and simulated data, and find that clustering-based algorithms are generally faster while local-

search heuristics give generally better results. A hybrid algorithm between the two is the best bet in practice.

- Finally, we provide a new approach of refined consensus clustering, that resolves the key problem of determining whether the computed consensus is a good one. A general problem with finding the median partition is that the given partitions may be dissimilar enough that it does not make sense to find a median. In such cases particular subsets of the partitions may in fact be similar, but a challenge is to find subsets which should be grouped together, and to provide medians for just those subsets. We take a statistical approach to this problem, using significance testing to determine when partitions are similar enough to be grouped together.

This paper is organized as follows. In the next section we describe the algorithms and the theory necessary for their evaluation. In Section 3 we present the data and the experimental methodology we employed to compare the algorithms. The results and discussion are given in Section 4. In Section 5 we report on the refined consensus clustering problem, and we conclude in Section 6.

2 Theory and Algorithms

Let S denote the sum of distances $\sum_{i=1}^{k} d(\pi_i, \pi)$ between a proposed consensus π and a set of k given partitions $\{\pi_1, \ldots, \pi_k\}$, where $d(,)$ is the symmetric difference distance. Each of the algorithms we implemented finds a consensus partition which is an approximation to π^*, the set partition that minimizes S. To compare the performance of the algorithms on sets of different number of elements n and number of partitions k, in this section we show how to normalize S in order to make it independent of n and k. In addition, we restate a lower-bound on the minimum sum of distances, which has been derived and reported previously [9].

2.1 Normalization of the Sum of Distances The idea is to relativize S with respect to purely random choices for the initial set partitions and the consensus. To correct for the number of set partitions, k, we simply divide S by it, since we are assuming that random set partitions are independent. The correction for n is more subtle, and involves standardizing S by transforming its distribution into a normal distribution. We define the Normalized distance, d_n, as a z-score

$$d_n(\pi_1, \pi_2) = \frac{d(\pi_1, \pi_2) - \mathbf{E}(D_n)}{\sqrt{\mathbf{Var}(D_n)}}. \tag{2.2}$$

To calculate d_n we give approximations for the expectation and variance of $d(,)$, as follows.

First, we calculate the probability that the co-clustering of two elements i and j is in disagreement in two different partitions (i.e. i and j are co-clustered in one partition and not co-clustered in the other, or vice versa). Let π_1 and π_2 be two set partitions of size n chosen uniformly at random from the B_n partitions of n elements, where B_n is the n-th Bell number (B_n can be computed in a number of ways, e.g. using the recurrence $B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k$, where $B_0 = 1$ [11]). We define the Bernoulli random variable $X_{ij}^{(m)}$ as 1 if i and j are co-clustered in π_m and 0 otherwise. First we give the distribution of $X_{ij}^{(m)}$.

Lemma 2.1.

$$\mathbf{P}(X_{ij}^{(m)} = 1) = \frac{B_{n-1}}{B_n}. \tag{2.3}$$

Proof. There are B_{n-1} ways to cluster the $n-1$ elements other than j. Then there is exactly one way to place j in the same cluster as i. Thus there are B_{n-1} set partitions where i and j are co-clustered.

Now let $Z_{ij} = |X_{ij}^{(1)} - X_{ij}^{(2)}|$. Z_{ij} is 1 if the clustering of i and j disagrees between π_1 and π_2, and 0 otherwise. Then, by definition of the symmetric difference distance,

$$D_n = d(\pi_1, \pi_2) = \sum_{i<j} Z_{ij}. \tag{2.4}$$

We observe that the Z_{ij} are identically distributed Bernoulli random variables. Then we have that

Lemma 2.2.

$$\mathbf{E}(Z_{ij}) = 2\frac{B_{n-1}}{B_n}\left(1 - \frac{B_{n-1}}{B_n}\right), \tag{2.5}$$

and

$$\mathbf{E}(D_n) = 2\binom{n}{2}\frac{B_{n-1}}{B_n}\left(1 - \frac{B_{n-1}}{B_n}\right). \tag{2.6}$$

Proof. Z_{ij} equals 1 exactly when one of the $X_{ij}^{(m)}$ is 1 and the other 0. Thus,

$$\begin{aligned} \mathbf{E}(Z_{ij}) &= \mathbf{P}(Z_{ij} = 1) \\ &= 2\mathbf{P}(X_{ij}^{(m)} = 1)(1 - \mathbf{P}(X_{ij}^{(m)} = 1)), \end{aligned}$$

in which substituting (2.3) yields (2.5). To show (2.6) we take the expected value of (2.4) and substitute (2.5).

While the Z_{ij}'s are not pairwise independent in general (for all i and j), most of the pairs are. Therefore this suggests that D_n should have a normal distribution for large enough n. (We confirmed that this approximate normality holds for all our data sets by running an Anderson-Darling test [12] to test for normality, and we found that D_n is normal with p-value (significance) < 0.01 for $n \geq 50$).

A normally distributed variable that is a sum of n Bernoulli random variables has a variance that is approximately $np(1-p)$. This suggests that

$$\mathbf{Var}(D_n) \approx \left(1 + \frac{2B_{n-1}^2}{B_n^2} - \frac{2B_{n-1}}{B_n}\right) \mathbf{E}(D_n).$$

While these values are slow to compute for large n using recurrences for B_n, we can use approximation formulas for the ratio B_n/B_{n+1}, like the following asymptotic one, derived from an asymptotic formula for B_n given in [13].

$$\frac{B_n}{B_{n+1}} \to \frac{\log n}{n} \text{ as } n \to \infty.$$

Applying this approximation above we find, as $n \to \infty$,

$$\mathbf{E}(D_n) \to \frac{n \log(n-1)}{n-1} \left(n - \log(n-1) - 1\right), \text{and}$$

$$\begin{aligned} \mathbf{Var}(D_n) &\to \mathbf{E}(D_n) \\ &+ \mathbf{E}(D_n) \frac{2\log(n-1)(1-n+\log(n-1))}{(n-1)^2}. \end{aligned}$$

As d_n is approximately normally distributed, then so is the sum

$$S_n = \sum_{i=1}^{k} d_n(\pi_i, \pi),$$

as well as the quantity S_n/k. S_n/k has the nice property that it is a z-score and can be compared to other z-scores and to tables of z-scores to find out the significance of such a score compared to a random event. It is this last quantity, S_n/k, that we will use when comparing the algorithms' performance over sets of data of different number of elements n. We refer to it as the *Average Normalized Sum of Distances*. When the number of elements is the same we use a simpler quantity, the *Average Sum of Distances*, where

$$Avg.\ SOD = \frac{S}{k \cdot \binom{n}{2}}.$$

2.2 Theoretical Lower Bound Since the median partition problem is NP-complete, it is useful to have performance bounds on the heuristics. While upper bounds of the heuristics' performance exist [3, 10] and are useful, lower bounds on the optimal solution for a given input are more important in judging how well any heuristic does in practice. A good lower bound is given in [9], and we restate it here. Let $\{\pi_1, ..., \pi_k\}$ be a set of set partitions, and let $X_{ij}^{(m)}$ be defined as in Sec. 2.1. Then for any set partition π, the sum of distances from it to the k given ones is at least

$$\sum_{i<j} min\left(\sum_{p=1}^{k} X_{ij}^{(p)},\ k - \sum_{p=1}^{k} X_{ij}^{(p)}\right). \tag{2.7}$$

We use this lower bound to estimate the worst-case distance between our solutions and the optimal solution.

2.3 Algorithms All of the algorithms we implemented and tested produce a consensus set partition from an input set of k set partitions $\{\pi_1, ...\pi_k\}$. Recall that the optimal consensus is the set partition which minimizes S.

- **Best Of K (BOK)** In this algorithm we compute S using each input partition π_i as the consensus, in turn. Then we choose that π_i that gives the minimum value for S. It can be shown that this algorithm is a 2-approximation and has running time $O(k^2 n)$ [2].

- **Clustering-based Algorithms** For these three algorithms we first compute a distance matrix $M = (m_{ij})$, where m_{ij} is the proportion of the k input partitions that have i and j clustered apart. Then we choose a cutoff value, τ. This preprocessing step takes $O(n^2 k)$ time.

  In the **Majority Rule (MR)** algorithm (also called quota rule [3]) we begin with every element clustered separately. Then for every pair (i, j) where $m_{ij} < \tau$, we join together the blocks containing i and j. The running time of this algorithm is dominated by the preprocessing step.

  In **Average Link (AL)**, which is the standard average-link agglomerative clustering algorithm, we likewise begin with every element clustered separately. Then we find the two blocks with the smallest average distance, computed using M, and merge them together. We continue doing this until the two closest blocks have a distance $\geq \tau$. This algorithm can be implemented to run in $O(n^2(k + \log n))$, including the preprocessing.

In the **CC Pivot algorithm** [10] we repeatedly choose a pivot element p uniformly at random from the unclustered elements. Then we form a block containing p and every element i where $m_{pi} < \tau$. We continue recursively on the remaining elements, until everything has been clustered. Ailon *et al.* [10] show that this gives an expected $\frac{11}{7}$-approximation algorithm. The running time of this algorithm is dominated by the preprocessing.

- **Local Search-based Heuristics** Given a set partition π, a one element move is any transformation $\pi \mapsto \pi'$ that moves some element a from one block into another one. For example, $\{\{1,2,4,6\},\{3,4\}\}$ is transformed into $\{\{2,4,6\},\{1,3,4\}\}$ with a one element move that sends element 1 from the first block into the second. Given that we allow moves into the "empty" block, we can reach any set partition from any other one with some sequence of at most $n-1$ one element moves.

  Thus, we can begin with some initial candidate partition π and apply one element moves to π to improve it as a consensus. Filkov and Skiena [3] give an efficient way to compute the change in S that results from a one element move, allowing for the two following algorithms to be implemented with $O(n^2k)$ preprocessing time and $O(n)$ time for each one element move.

  In the **Best One Element Moves (BOEM)** algorithm we start with an initial consensus partition–we test starting with the results of BOK or AL. Then we repeatedly perform the best one element move as long as it decreases S.

  In the **Simulated Annealing (SA)** algorithm we use simulated annealing with one element moves. Thus at any given step we always accept any move that decreases S and those that increase S with some decreasing probability. The actual running time depends on the cooling schedule of the SA.

3 Implementation, Data Sets, and Testing Methodology

We have written a C++ library for set partitions, supporting many common operations. Our library has procedures for enumerating all set partitions and selecting set partitions uniformly at random, using algorithms from [14]. It also generates the noisy set partitions described in Sec. 3.1. We have implemented six heuristics that find approximate solutions to the median partition problem: Best Of K, Best One Element Moves, Majority Rule, Average Link, CC Pivot, and Simulated Annealing, with several variations of each. The code is available from *http://www.cs.ucdavis.edu/∼filkov/software/conpas*.

		Mushrooms $n = 8124, k = 22$ Lower Bound: 0.3812	
	Heuristic	Avg. SOD	Time/s
	BOK	0.4068	**1**
Clustering	MR ($\tau = 0.5$)	0.5066	22
	MR ($\tau = 0.25$)	0.3963	21
	AL ($\tau = 0.5$)	0.3940	3727
	AL ($\tau = 0.25$)	0.4117	3641
	CCPivot ($\tau = 0.5$)	0.3993	1222
	CCPivot ($\tau = 0.25$)	0.4366	1210
Local Search	BOEM (BOK seed)	0.3992	21
	BOEM (AL seed)	**0.3919**	3899
	SAFast (BOK seed)	0.4913	47
	SAMedium (BOK seed)	0.4065	542
	SASlow (BOK seed)	**0.3919**	5780
	SASlow (AL seed)	0.3929	9428

Table 1: Results and running times for the heuristics on the Mushrooms data set

In our implementation, we tested the clustering algorithms for two different thresholds, $\tau = 0.5$ and $\tau = 0.25$. For the local search algorithms, we choose three different cooling schedules for SA, which we call SAFast, SAMedium, and SASlow, each starting with the results of BOK. We also test two hybrid algorithms: SASlow and BOEM starting with the consensus partition produced by Average Link. All tests were done an AMD Opteron 2.6 GHz processor with 4 GB of RAM.

3.1 Data Sets In testing the above algorithms we considered both artificial and real data. The real data includes the Mushrooms data set from the UCI repository [15], which has 8124 mushrooms and 22 attributes for each mushroom, such as cap color, stalk shape, habitat, etc. We form one cluster for each attribute, where every mushroom with a particular value for that attribute is clustered together. Any mushrooms with missing values were also clustered together. The other dataset used was the yeast stress experiment [1], which has the responses of 6152 yeast genes to 170 different stress conditions. We use KNNimpute [16] to fill in missing values in the data. Then we clustered it in two ways: by experiment and by gene, using standard agglomerative average-link clustering with a cutoff of 0.5.

To describe the artificial data, we first introduce the concept of noise. Given a set partition π of n elements, we can introduce some percentage of noise x into π by performing xn random one element moves on π. We

		Yeast (by Gene) $n = 170, k = 6152$ Lower Bound: 0.4015	
	Heuristic	Avg. SOD	Time/s
	BOK	0.4217	826
Clustering	MR ($\tau = 0.5$)	0.4773	3
	MR ($\tau = 0.25$)	0.4434	**2**
	AL ($\tau = 0.5$)	0.4102	24
	AL ($\tau = 0.25$)	0.4442	11
	CCPivot ($\tau = 0.5$)	0.4094	119
	CCPivot ($\tau = 0.25$)	0.4438	121
Local Search	BOEM (BOK seed)	0.4119	834
	BOEM (AL seed)	0.4086	25
	SAFast (BOK seed)	0.4085	844
	SAMedium (BOK seed)	0.4083	850
	SASlow (BOK seed)	**0.4082**	954
	SASlow (AL seed)	0.4083	136

Table 2: Results and running times for the heuristics on the yeast dataset, clustered by gene

		Yeast (Experiment) $n = 6152, k = 170$ Lower Bound: 0.4031	
	Heuristic	Avg. SOD	Time/s
	BOK	0.4314	**20**
Clustering	MR ($\tau = 0.5$)	0.5326	87
	MR ($\tau = 0.25$)	0.4331	85
	AL ($\tau = 0.5$)	0.4154	1778
	AL ($\tau = 0.25$)	0.4341	1692
	CCPivot ($\tau = 0.5$)	0.4169	4824
	CCPivot ($\tau = 0.25$)	0.4341	4908
Local Search	BOEM (BOK seed)	0.4256	149
	BOEM (AL seed)	**0.4118**	1832
	SAFast (BOK seed)	0.4335	160
	SAMedium (BOK seed)	0.4302	572
	SASlow (BOK seed)	0.4122	4850
	SASlow (AL seed)	0.4122	6540

Table 3: Results and running times for the heuristics on the yeast dataset, clustered by experiment

choose an element uniformly at random and a block to move it to uniformly at random, excluding its own block but including the "empty" block. Thus, for example, if $n = 100$ at 10% noise we would perform 10 such moves.

Our simulated data consists of generating k noisy partitions of n elements, with x noise, all based upon the same randomly chosen partition. We generated two artificial noise studies to test our algorithms. In the first, we fixed k and x and varied n from 50 to 500, and in the second we fixed n and x and varied k from 25 to 150.

4 Results and Discussion

The results for the real data sets are given in Tables 1, 2, and 3. The values given are the Avg. SODs: sums of Rand distances, averaged over 100 experiments, and averaged by the number of set partitions, k and all pairs, $\binom{n}{2}$. The best values in each column are shown in bold. The running time is given in seconds. For each data set we also provide a lower bound on Avg. SOD, as calculated by the formula in Sec. 2.2. For the three data sets the best scoring algorithms, BOEM/AL (twice) and SASlow get, respectively, to within 2.8%, 1.7%, and 2.1%, of the lower bound, and hence of the optimum.

Fig. 1 and Fig. 2 show the results of the noise study for the 4 best algorithms. We ran each algorithm 100 times on random data and averaged the results. Shown on the y-axis is the Average Normalized SOD, i.e. S_n/k averaged over 100 runs. The y-axis values can be also thought of as z-scores, and their magnitude for our algorithms indicates that all our results are very reasonable and are much better than chance.

Fig. 1 compares the performance (in terms of the Average Normalized SOD) of the four top algorithms over data sets of varying sizes. The stability of the results indicate with confidence that the greedy algorithm BOEM, seeded with the result from Average Link performs the best. The apparent non-linear trend of the results is probably due to at least two factors: (1) our asymptotic approximation and normality assumption from Sec. 2.1 fail for smaller values of n, and (2) the algorithms do better for smaller values of n.

Fig. 2 shows that the performance of the algorithms is not dependent on k, the number of given set partitions. Still, it is interesting to examine the variability in the results, and although some algorithms appear to have a smaller variability than others, the differences in the absolute values of the results are too small to claim that with any statistical certainty.

Overall, BOEM/AL was not our slowest algorithm, and often it is not the fastest, but it performed best here and in the above study on the real data, so we recommend it for any general use.

5 Refined Consensus Clustering

It is unlikely that any real large dataset is composed of clusterings that are very similar. Thus a consensus clustering will likely not give very useful information. A challenge is to determine in what situations a consensus clustering is appropriate. To address this problem here we propose to refine the set of clusterings to

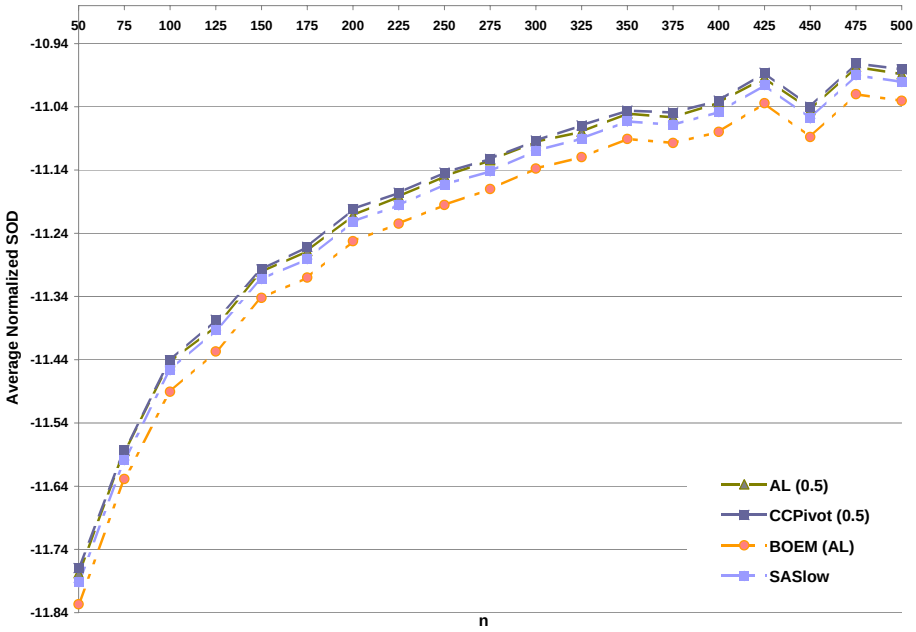

Figure 1: Evaluation of the behavior of the top 4 algorithms with varying n. Here $k = 50$ and the noise is 10%.

smaller sets, each of which will have a better overall consensus. Our approach, *refined consensus clustering*, effectively amounts to clustering the given clusterings and calculating consensuses for the resulting clusters of set partitions. While previous approaches to cluster clusterings exist [17], they are application specific and do not benefit from the full median partition framework that we have built.

We use an agglomerative clustering approach (the Average Link method) to cluster the set partitions. We employ the *adjusted Rand* [6] distance as our dissimilarity measure of choice, because it is a normalized distance, and in this algorithm it does not affect the running time adversely. We begin with all clusterings in separate groups and join together those groups with the smallest average distance. The average is taken over all pairs between the two groups.

In the standard agglomerative clustering method we would continue joining groups until the smallest average distance exceeded some threshold, e.g. 0.5. Instead we present a novel approach, using the idea of the "noise level" of a group of clusterings. Recall from Sec. 3.1 that we generated a noisy set of partitions with a particular noise level. In general, every group of set partitions can be thought of as having been generated around some consensus with a given noise level. We formalize this by taking the sums of the pairwise distances between set partitions, i.e. for some set of partitions $\Pi = \{\pi_1, \pi_2, ..., \pi_k\}$ we define

$$SP(\Pi) = \sum_{i<j} d(\pi_i, \pi_j).$$

Considering all groups of a given noise level, their sum of pairwise distances SP form a distribution. Thus, given any particular group of set partitions, we can calculate a p-value to determine whether it has a given level of noise or more. In practice we do this by sampling the distribution (which depends on n and k) as we do not have a way to compute it analytically.

In our algorithm we have a cutoff noise level, c, and we continue clustering until the next cluster formed would have a noise level of c or higher. This ensures that the resulting clusters will be reasonably similar and that their consensus will be useful.

5.1 Implementation We have implemented this procedure as part of our general set partition library.

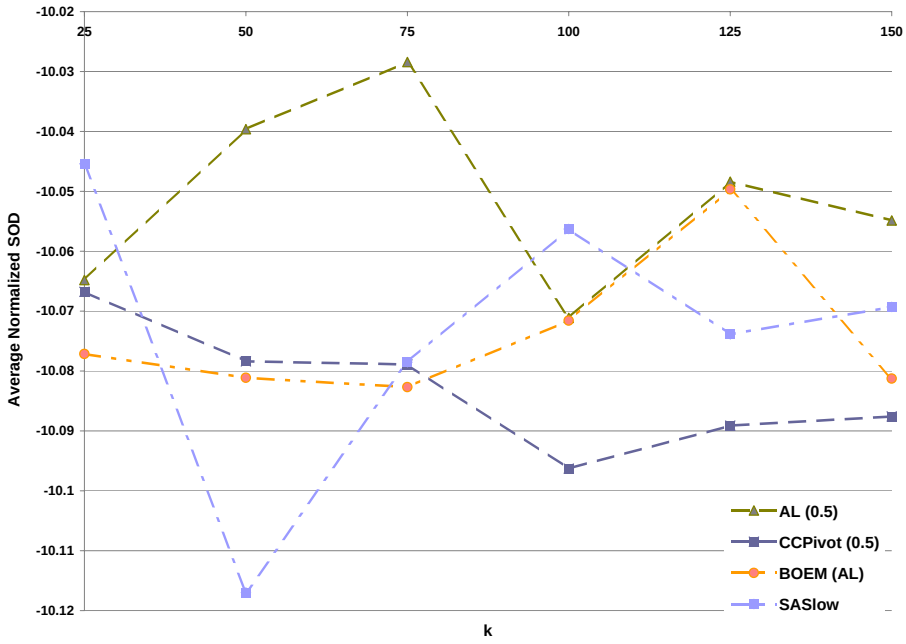

Figure 2: Evaluation of the behavior of the top 4 algorithms with varying k. Here $n = 100$ and the noise is 10%.

It takes as input a list of set partitions and an optional noise threshold and generates as output the clustering of those partitions along with a consensus partition, as calculated by the BOEM/AL algorithm, for each non-trivial cluster (size > 2). The default threshold we use is 0.25, as experimentally we have found it to be roughly the point at which the partitions become mostly random and a consensus is no longer useful. We leave a detailed study of determining this threshold for future work.

5.2 Refined Consensus Results As an example application of our refined clustering method, we ran our algorithm on the Mushroom dataset and the Yeast dataset, both described in Sec. 3.1. The results are found in Tables 4 and 5. For the Mushroom data, the algorithm found two subsets of attributes to be significant at the level of noise of 0.25. For the Yeast data, the algorithm found only one subset of experiments to be significant at the level of noise of 0.25. We found a consensus clustering for each, and computed its Avg. SOD. These clusters have much better consensuses than the general datasets, suggesting that the respective consensuses are more representative.

6 Conclusion and Future Work

The problem of consensus clustering is important because of the prevalence of large data sets and their availability from different sources. A number of heuristics exist for solving a version of this problem called median partition, but no extensive studies have been done comparing their results over a variety of real and artificial data sets.

In this paper we reported on the development of software libraries for solving the median partition problem, performed a comparison of the algorithms, and proposed a refined consensus clustering as a more objective way of finding consensuses. Our results indicate that of the two classes of heuristics that we implemented, the clustering-based methods are generally faster, while the local-search heuristics result in a better performance. We propose a hybrid algorithm, a greedy local search method starting from an initial consensus provided by Average Link Clustering (i.e. BOEM/AL) which has an excellent performance/cost ratio in practice, and most often is the very best performer. Our code is publicly available as a resource for the community, and can be used for various applications.

Subset of Mushroom Attribs.	Avg. SOD	Lower Bound
ring number, gill spacing, veil color, ring type	0.0591	0.0591
population, stalk shape, bruises	0.1551	0.1287

Table 4: Results of refined consensus clustering on the Mushroom dataset

Subset of Experiments	Avg. SOD	Lower Bound
Nitrogen Depletion 12 h 1.5 mM diamide (90 min) Nitrogen Depletion 1 d Nitrogen Depletion 2 d YPD 12 h ypd-2 dtt 015 min dtt-2 constant 0.32 mM H2O2 (40 min) rescan DBY7286 + 0.3 mM H2O2 (20 min)	0.0678	0.0678

Table 5: Results of refined consensus clustering on the Yeast dataset

Several future directions naturally fall out of this study. We are aware that there is at least one other class of algorithms for solving the consensus clustering problem, which use information theoretic metrics [18, 19]. It would be useful to compare those with the combinatorial ones in our library. The refined consensus clustering is still a work in progress. A most immediate improvement would be an automated way to determine the cutoff threshold for the significance. Also, the results at this point are difficult to read out. A visual representation, maybe in terms of some heatmaps of the clusters, would greatly aid their interpretation.

References

[1] A. Gasch, P. Spellman, C. Kao, O. Carmen-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown. Genomic expression programs in the response of yeast cells to environment changes. *Mol. Bio. Cell*, 11:4241–4257, 2000.

[2] V. Filkov and S. Skiena. Integrating microarray data by consensus clustering. *International Journal on Artificial Intelligence Tools*, 13(4):863–880, 2004.

[3] V. Filkov and S. Skiena. Heterogeneous data integration with the consensus clustering formalism. *Proceedings of Data Integration in the Life Sciences*, pages 110–123, 2004.

[4] Alexander Topchy, Anil K. Jain, and William Punch. Combining multiple weak clusterings. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 331, Washington, DC, USA, 2003. IEEE Computer Society.

[5] M. Bender, S. Sethia, and S. Skiena. Efficient data structures for maintaining set partitions. *Proceedings of Seventh Scandinavian Workshop on Algorithm Theory*, pages 83–96, 2000.

[6] L. Hubert and P. Arabie. Comparing partitions. *J. Class.*, 2:193–218, 1985.

[7] E.B Fawlkes and C.L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78:553–584, 1983.

[8] Y. Wakabayashi. The complexity of computing medians of relations. *Resenhas IME-USP*, 3:323–349, 1998.

[9] M. Bertolacci and A. Wirth. Are approximation algorithms for consensus clustering worthwhile? In *Proceedings of the Seventh SIAM ICDM*, 2007.

[10] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the thirty-seventh annual ACM Symposium on Theory of Computing*, pages 684–693, 2005.

[11] H. Wilf. *generatingfunctionology*. A K Peters, 3rd edition, 2006.

[12] H.C. Thode Jr. *Tests for Normalcy*. Marcel Dekker, NY, 2002.

[13] J. Pitman. Some probabilistic aspects of set partitions. *American Mathematical Monthly*, 104:201–209, 1997.

[14] A. Nijenhuis and H. Wilf. *Combinatorial Algorithms*. Academic Press, 1978.

[15] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases. 1998.

[16] O. Troyanskaya et al. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17:520–525, 2001.

[17] A.D. Gordon and M. Vichi. Partitions of partitions. *Journal of Classification*, 15:265–285, 1998.

[18] D. Cristofor and D. Simovici. Finding median partitions using information-theoretical-based genetic algorithms. *Journal of Universal Computer Science*, 8(2):153–172, 2002.

[19] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining partitionings. In *Proc. of AAAI*, pages 93–98. AAAI/MIT Press, 2002.