ISSN: 1803-3814 (Printed), 2571-3701 (Online)
https://doi.org/10.13164/mendel.2019.1.087

EFFICIENT COMPUTATION OF FITNESS FUNCTION FOR EVOLUTIONARY CLUSTERING

Sergey Muravyov✉, Denis Antipov, Arina Buzdalova, Andrey Filchenkov

Computer Technology Lab, ITMO University, Russia

smuravyov@itmo.ru✉, dantipov@itmo.ru, abuzdalova@gmail.com, afilchenkov@itmo.ru

Abstract

Evolutionary algorithms (EAs) are random search heuristics which can solve various optimization problems. There are plenty of papers describing different approaches developed to apply evolutionary algorithms to the clustering problem, although none of them addressed the problem of fitness function computation. In clustering, many clustering validity indices exist that are designed to evaluate quality of resulting points partition. It is hard to use them as a fitness function due to their computational complexity. In this paper, we propose an efficient method for iterative computation of clustering validity indices which makes application of the EAs to this problem much more appropriate than it was before.

Keywords: *Clustering, evolutionary clustering, clustering validity indices, fitness function.*

Received: 09 May 2019
Accepted: 07 June 2019
Published: 24 June 2019

1 Introduction

The goal of clustering is to define a finite set of categories (clusters) to describe a dataset according to similarities among its objects [1]. There are plenty of applications of clustering: from image processing [2] and market segmentation [3] to web mining [4] and document categorization.

Clustering problem is initially formulated in general terms. Many mathematical formalisms were suggested to describe it. It was recognized quite a long time ago that the selection of a certain formalism to estimate the clustering quality will strongly influence both the algorithm that should be chosen and the result claimed to be the best [5], and that it is hard to compare different formalisms [6]. Kleinberg formulated three essential properties of clustering algorithms and proved that there is no way to create an algorithm that fits all these properties simultaneously [7].

There are two ways of evaluating clustering partitions, namely internal and external measures. External measures use some extra information about the task, for example, different benchmarks or class labels. These measures can not be applied in real life, because there might not be any external information about the problem to be solved. Internal measures use only the information about the structure of partition itself and only clustering validity indices (CVIs) satisfy this requirements.

Approaches to solve clustering task can be also divided into two categories: *indirect*, when we use a black-box optimization algorithm with built-in quality evaluation measure, for example, $k-$Means [8], Spectral algorithm [12], DBSCAN [9] that are actually conventional clustering algorithms; and *direct* when we can set quality measure and search for the optimal partition in the space of all possible partitions for a given task. This problem is a discrete optimization task, one of the ways to solve it is to apply evolutionary algorithms [10].

Evolutionary algorithms (EAs) is a general name of a class of the random search heuristics that were inspired by the ideas of natural evolution. Namely, they are based on iterative improvement of some existing solution via mutation, crossover and selection operators. Often evolutionary algorithms cannot find optimal solution in reasonable time, however they were found effective for finding good enough solutions pretty fast. This makes them suitable for solving NP-hard problems, which clustering problem belongs to.

To apply an EA one first needs to define a quality measure for all solutions,, which is often called *a fitness function*. The main problem of all CVIs is super linear time complexity that might be crucial for the performance of an evolutionary clustering algorithm, no matter what strategy is chosen.

In this paper, we propose new incremental methods for computation of different CVIs' and provide experiments on several primitive strategies to find out if our method works faster than conventional CVI computation with the same outcomes. We study how using of the incremental CVI evaluation can improve the performance of three considered EAs solving a clustering problem.

The structure of the paper is following. Section 2 contains the description of incremental methods and provides the comparison with conventional CVI computation on different datasets. In section 3, we provide a description of the strategies that were used to test the CVI incremental computation approach. Section 4 provides the information about experiments.

2 Incremental CVI evaluation

In this section, we provide the description of new methods of incremental CVI computation. We considered 19 CVIs in this paper: Calinski-Harabasz index, Dunn index, C-Index, Davies-Bouldin Index, Modified Davies-Bouldin Index, Silhouette index, gD31, gD41, gD51, gD33, gD43, gD53 indices, CS index, Sym-Index, SymDB index, COP index, SV index, OS index, S_Dbw index. Descriptions of all these CVIs can be found in [11]. Due to the size limits of the paper we provide detailed incremental methods only for Calinski-Harabaz index, COP index, Silhouette index and Modified Davies-Bouldin index. Implementations of incremental computation for the other indices can be found in our repository[1]. Although we do not provide the methods for all the CVIs used in the paper, in the Table 2 we show that incremental recalculation complexity is much better than full recalculation of each CVI.

Table 1: Time complexity comparison between full and incremental recalculation of CVIs

CVI	Full time complexity	Incremental time complexity
Calinski-Harabasz index	$O(n \log n)$	$O(n)$
Dunn index	$O(n^2)$	$O(1)$
Davies-Bouldin Index	$O(n \log n)$	$O(n)$
Modified Davies-Bouldin Index	$O(n \log n)$	$O(n)$
Silhouette index	$O(n^2)$	$O(n)$
C-Index	$O(n \log n)$	$O(n)$
gD31	$O(n^2)$	$O(n)$
gD41	$O(n^2)$	$O(n)$
gD51	$O(n^2)$	$O(n)$
gD33	$O(n^2)$	$O(n)$
gD43	$O(n^2)$	$O(n)$
gD53	$O(n \log n)$	$O(n)$
CS index	$O(n \log n)$	$O(n)$
Sym-Index	$O(n^2)$	$O(n)$
SymDB index	$O(n^2)$	$O(n)$
COP index	$O(n^2)$	$O(n)$
SV index	$O(n \log n)$	$O(n)$
OS index	$O(n^2 \log n)$	$O(n)$
S_Dbw index	$O(n \log n)$	$O(n)$

2.1 General notation

First, we introduce some formal notations of incremental computation. We denote by X a dataset, which consists of N points; C is its partition consisting of clusters $C_1, \cdots, C_m$. Centroid of cluster C_i is defined accordingly:

$$\overline{C_k} = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i. \tag{1}$$

The centroid of the entire dataset is defined as follows:

$$\overline{X} = \frac{1}{N} \sum_{x_i \in X} x_i. \tag{2}$$

The formula of the Euclidean distance between objects p and q, that has dimensionality n, of the dataset is:

$$d_e(p, q) = \sqrt{\sum_{k=1..n} (p_k - q_k)^2} \tag{3}$$

The mutation operator used inside EAs changes the structure of partition C by moving some points from one cluster to another. The task of incremental CVI recalculation is to effectively recalculate the value of CVI after such small change in the structure without recalculating it from scratch (which can take too much time). Further when discussing the iterative recalculation of CVI we denote by x_{id} the point that was moved by mutation, by C_a the initial cluster of this point and by C_b — the cluster this point was moved to.

[1] `https://github.com/AntipovDen/EvoClusterization`

2.2 Calinski-Harabasz index

This measure is calculated by the formula:

$$CH(X) = \frac{N-K}{K-1} \frac{\sum_{C_k \in C} |C_k| d_e(\overline{C_k}, \overline{X})}{\sum_{C_k \in C} \sum_{x_i \in C_k} d_e(x_i, \overline{C_k})}. \tag{4}$$

The numerator is the sum of the distances from the centroids of the clusters to the centroid dataset, the denominator is the sum over the distances from points to the centroids of their clusters. Accordingly, when recalculating, it is necessary to update the standing from $\overline{C_a}$ to $\overline{X}$, from $\overline{C_b}$ to $\overline{X}$ and also the distances from the points of the C_a cluster to its centroid, from the points of the C_b cluster to its centroid and remove the distance $d_e(x_{id}, C_a)$, and instead add the distance $d_e(x_{id}, C_b)$. But since updating the distances from all points to the centroid is a rather expensive operation, the following optimization was applied – if the centroid moved less than ϵ multiplied by the diameter of the dataset, the corresponding distances are not recalculated.

Complexity assessment: the measure itself is calculated for $O(n)$, where n is the size of the dataset, but since it is still necessary to additionally calculate the diameter for optimization during the recalculation, the diameter calculation adds $O(n \log n)$ to asymptotics, so that the final asymptotics is obtained just as follows. Incremental counting takes $O(n)$ for updating centroids. Further, if the distances from the points to the centroids are not recalculated (and recalculation will be necessary only in the worst case, when the centroid will move strongly), then the final operations will take $O(1)$, and in general the final estimate is $O(n)$. If we had to recalculate the distance to the centroids, then this adds another $O(n)$ pass with the calculation of the Euclidean metric.

2.3 COP index

The measure is calculated as follows:

$$COP(C) = \frac{1}{N} \sum_{C_k \in C} |C_k| \frac{1/|C_k| \sum_{x_i \in C_k} d_e(x_i, \overline{C_k})}{\min_{x_i \notin C_k} \max_{x_j \in C_k} d_e(x_i, x_j)} \tag{5}$$

During the initial calculation it is proposed to save $d_e(x_i, \overline{C_k})$, it is also necessary to save $d_e(x_i, x_j)$ for each i and j, all possible distances for counting maxima (two-dimensional array), and also all maxima for a minimum (one-dimensional array). In other words, for each cluster we will separately store an array of numerators and denominators.

During the recalculation procedure, it will be necessary to remove $d_e(x_i d, C_a)$ from the numerator for the term for C_a and add the term $d_e(x_{id}, C_b)$ for the numerator of C_b. The distances from the points of the C_a and C_b clusters to their centroids will not be recalculated if the centroids have shifted less than ϵ multiplied by the diameter of the dataset. Also, in the arrays for the denominator C_a, it will be necessary to add all possible distances $d_e(x_{id}, x_j)$, where $x_j \in C_a$, and remove all possible distances $d_e(x_{id,j})$, where $x_j \in C_b$, from the arrays for the denominator C_b. Then you just need to recalculate private for C_a and C_b.

Complexity assessment: the initial calculation of the measure requires the calculation of all possible $d_e(x_i, x_j)$, which takes $O(n^2)$ time, then you just have to count the minima and maxima $d_e(x_i, \overline{C_k})$, all this together takes $O(n^2)$ time, and it is still necessary to calculate the diameter to accelerate the incremental conversion, which takes $O(n \log n)$ time, but the final asymptotic remains $O(n^2)$.

During recalculation it is necessary to update the arrays of the corresponding numerators and denominators for C_a and C_b and recalculate the minima and maxima only for C_a and C_b and only if new distances for C_a are less or more, and for C_b only if x_{id} . It is also required to update centroids for $O(n)$ time and, in the worst case, to recalculate additionally the distances from points to centroids of the C_a and C_b clusters for $O(n)$, but the final asymptotics remains $O(n)$ time.

2.4 Silhouette index

The measure is calculated as follows:

$$Sil(C) = \frac{1}{N} \sum_{C_k \in C} \sum_{x_i \in C_k} \frac{b(x_i, C_k) - a(x_i, C_k)}{\max(a(x_i, C_k), b(x_i, C_k))}, \tag{6}$$

where

$$a(x_i, C_k) = 1/|C_k| \sum_{x_j \in C_k} d_e(x_i, x_j), \tag{7}$$

and

$$b(x_i, C_k) = \min_{C_l \in C \setminus C_k} (1/|C_k| \sum_{x_j \in C_l} d_e(x_i, x_j)). \tag{8}$$

Here, $a(x_i, C_k)$ and $b(x_i, C_k)$ are computed for all points in the dataset. Accordingly, it is necessary to keep them during the initial calculation. Then, when moving point x_{id} from cluster C_a to cluster C_b, you need to re-calculate $a(x_{id}, C_b)$ and $b(x_{id}, C_b)$, and also update the values of $a(x_i, C_k)$ and $b(x_i, C_k)$ for all points of the clusters C_a and C_b. Then you need to re-calculate the sum and get an answer.

Complexity assessment: counting $a(x_i, C_k)$ and $b(x_i, C_k)$ takes $O(n)$ times, respectively, counting them for all points takes $O(n^2)$ time. Further, the final answer is calculated for $O(n)$. During recalculation procedure, it is necessary to re-calculate the functions $a(x_i, C_k)$ and $b(x_i, C_k)$, but only for one point, which takes $O(n)$. Further, the values of these functions are updated in $O(1)$ for all points of the clusters C_a and C_b, which also takes $O(n)$, and the counting of the final answer also takes $O(n)$. So the final complexity of the recalculation is $O(n)$.

2.5 Modified Davies-Bouldin index

First let's consider the original Davies-Bouldin index formula:

$$DB(C) = \frac{1}{K} \sum_{C_k \in C} \max_{C_l \in C \setminus C_k} \frac{S(C_k) + S(C_l)}{d_e(\overline{C_k}, \overline{C_l})}, \tag{9}$$

where

$$S(C_k) = 1/|C_k| \sum_{x_i \in C_k} d_e(x_i, \overline{C_k}). \tag{10}$$

In the implementation, the function $S(C_k)$ is used, during the initial calculation it is proposed to save it. It is also necessary to calculate in advance the sums of all possible $S(C_k) + S(C_l)$, then divide them by $d_e(C_k, C_l)$ and store. Accordingly for the CVI computation it is necessary to find the maximum by the sets of this divisions.

Now, to recalculate this CVI, it is necessary to update $S(C_a)$ and $S(C_b)$ and the saved quotients, in which $S(C_a)$ and $S(C_b)$ were used, to find the maximum. But the operation of recalculating distances from a point to the centroid of its cluster is expensive – therefore, if the centroid has moved less than ϵ, multiplied by the diameter of the dataset, $S(C_a)$ and $S(C_b)$, will not be recalculated. At the end, the search for the maximum is performed again.

Complexity assessment: the initial calculation takes $O(n)$ to find all $S(C_k)$ and $O(n \log n)$ to find the diameter of the dataset, which is necessary to speed up the conversion. Recalculation takes $O(n)$ in the worst case, so if the centroids have greatly moved, and in the best case it is $O(K)$, where K is the number of clusters, but it is very small compared to the size of the dataset, therefore this asymptotics is equal to $O(1)$, and it is also necessary at the beginning to recalculate the centroids, which will take $O(n)$ time, so the final complexity is $O(n)$.

Next, consider the modified Davies-Bouldin index. The CVI is calculated as follows:

$$DB^*(C) = \frac{1}{K} \sum_{C_k \in C} \frac{\max_{C_l \in C \setminus C_k} (S(C_k) + S(C_l))}{\min_{C_l \in C \setminus C_l} d_e(\overline{C_k}, \overline{C_l})}. \tag{11}$$

Here, the recalculation principal practically coincides with the recalculation principal of the original Dadies-Bouldin index, the main difference is that we save not the divisions, but the sums from the numerator. To recalculate the denominator, it will be necessary to recalculate the corresponding $d_e(C_k, C_l)$ for combinations with C_a and C_b.

The complexity estimate is the same as the complexity of the original Davies-Bouldin index, that is $O(n \log n)$ to be recalculated, and $O(n)$ for recalculation, because the recalculation of the denominator will take a maximum of $O(K^2)$, which is $O(1)$ in the current scope, and the recalculation of the numerator essentially coincides with the recalculation of the division in the original Davies-Bouldin index.

3 Evolutionary strategies

In this section, we describe three evolutionary strategies used in our experiments. For all of them by *initial solution* we mean some partition that was obtained through application of some conventional clustering algorithm.

Greedy algorithm. This one of the most simple EAs iteratively mutates the initial solution while it manages to find some strictly better solution. Namely, it stops as soon as the solution that it obtains through the mutation of the best-so-far solution is not the new best one.

The mutation operator first calculates for each point x the distances to the centroids of each cluster that x does not belong to. Then it finds ρ_x, that is, the minimal of such distances for point x. Finally, it moves 2^t points, where t is the number of the current iteration, with minimal ρ_x to the clusters that are the nearest to

those points (in terms of the distance to the centroid). All ties in selection of the points to move as well as in choice of the nearest centroid are broken uniformly at random.

$(1 + 1)$ **EA.** This algorithm generally reproduces the principles of the greedy algorithm, but uses different stopping criteria and mutation operator. The $(1 + 1)$ EA keeps the best-so-far partition and on each iteration mutates it and replaces the current one with the new one if the new partition is not worse than the current one. It stops after its time limit is reached.

In contrast to the greedy algorithm, the mutation operator of $(1 + 1)$ EA is randomized. First, it also calculates ρ_x, the distance to the nearest cluster for each point x. Then it chooses some number of points to move, and each point x is chosen with a probability proportional to ρ_x^{-1}. The number of points to move is equal to the veiled mutation rate for the current population.

To increase the effectiveness of our algorithm, we use the 1/5-th rule to control the value of the mutation rate. Initially the mutation rate is one. After each successful iteration, which is such that the new solution is at least as good as the best-so-far solution, we divide the mutation rate by two (but it cannot become less than one point). After unsuccessful iteration we multiply the mutation rate by $2^{1/4}$, but bound it with the half of the size of the solution. This rule while being easy to implement has been shown to be effective both in practical and theoretical analysis [13].

$(1 + 4)$ **EA.** The main difference of this EA from the $(1 + 1)$ EA is that in each iteration it generates and evaluates 4 new solutions in parallel. This increases the cover of the search space and helps to find new better solutions. However, parameter control for this algorithm is more complicated, so we do not use the 1/5-th rule for it. Instead, we generate the 4 solutions with different mutation rates, namely 1, 2, 4, and 8, using the same mutation operator as for the $(1 + 1)$ EA.

4 Experiments

In Section 2, we showed that the theoretical time complexity of the proposed incremental CVI recalculation methods is lower than the time complexity of the conventional recalculation. Here we provide experiments to check this statement in practice. Also we check the reliability of the incremental computation. Some incremental CVI computations have approximations, for example approximate calculation of centroids. So we have to check if incremental CVI computations make the same results as conventional CVI computations.

Experimental setup was the following: we took 5 datasets from the UCI database[2]: glass, iris, wholesale, seeds, and ionosphere. We used 4 CVIs mentioned in Section 2. Each strategy from Section 3 was performed 10 times for each dataset and each CVI (all runs had the same initial clustering obtained through Spectral clustering algorithm [12]). The results of the experiments are shown in Table 2 and Figure 1.

In Figure 1 we present the results of our study on the reliability of incremental CVI computation. We used the same experiment setting for each dataset and each CVI. We compare each algorithm using the incremental CVI computations and the conventional CVI computations. We set time limit of 5 minutes for all of the experiments.

We exclude the results for the greedy algorithm from Table 1, since this algorithm has not reached any CVI improvements regardless the way of CVI computation.

From Figure 1, we conclude that in most cases incremental CVI computation provide almost the same CVI rates as conventional computation, so we showed the reliability of suggested incremental CVI computation approach. We also note few exceptions that might be caused by finding some local minimum by $(1+1)$ strategy, especially *modified Davies-Bouldin* index on wholesale and iris datasets.

In Table 2, we present the results of our study on how much faster the optimization algorithms with incremental CVI are than the ones with conventional CVI. In these experiments for each pair dataset-CVI we first ran each algorithm with incremental CVI and time limit of 5 minutes. Then we ran the same algorithm with conventional CVI until it found some clustering that was at least as good as the one found by the corresponding run of the algorithm with incremental CVI (with the same initialization). These runs can take too much time, so we also added the time limit of 25 minutes for them.

In this approach we have a risk that a run of an algorithm with an incremental CVI does not improve the initial cluster. Therefore, the corresponding run of the algorithm with a conventional CVI does not perform any iteration, since its initial clustering is already as good as the one found by the first algorithm. We call such unfortunate runs *invalid runs* and we call all other runs *valid*.

Among the valid runs we also distinguish *successful runs*, that are the runs of an algorithm with conventional CVI that have finished before their time limit. The ratio of the successful runs to the valid runs is shown in the last column of Table 2.

In order to calculate the expected mean value of the algorithm runtime and the standard deviation of the runtime we use the same technique as in [14]. It was developed for the populations estimation but can be also

[2] `https://archive.ics.uci.edu/ml/datasets.php`

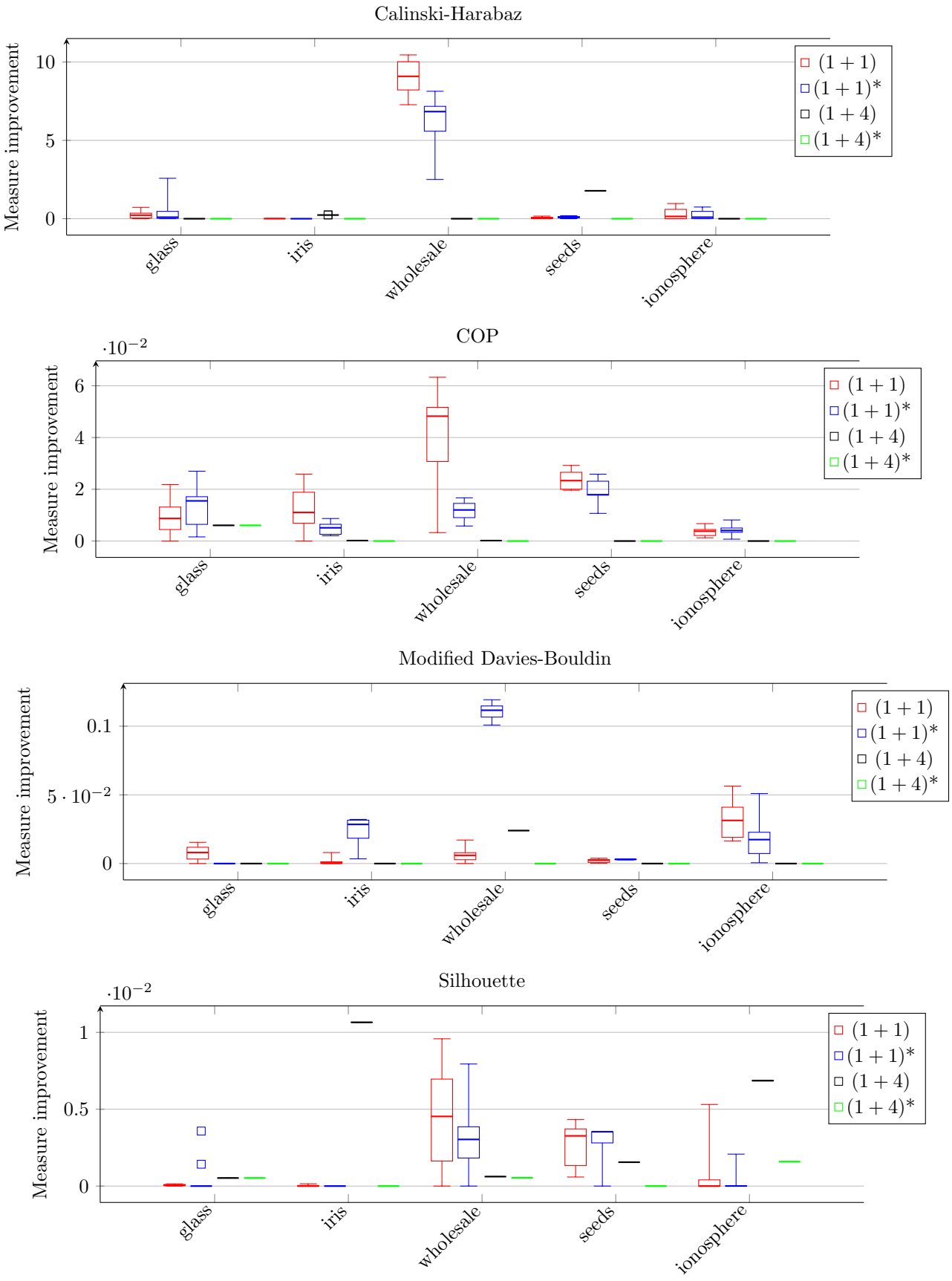

Figure 1: Comparison of different CVI improvements for each dataset. Algorithms without star (*) mark use incremental CVI computation, algorithms with a star use conventional CVI computation. Both types of algorithms perform 10 runs each and have 5 minutes time limit.

Table 2: Time estimations for the algorithms that use conventional CVI computation. Algorithms with incremental CVI computation reach the same measure rates in 5 minutes in all the cases.

Dataset	CVI	Mean, min		Deviation, min		Successful runs	
		$(1+1)$	$(1+4)$	$(1+1)$	$(1+4)$	$(1+1)$	$(1+4)$
glass	Calinski-Harabaz	75.1	—	86.6	—	2/8	—
	COP	22.8	> 25	30	unknown	5/9	0/10
	Modified Davies-Bouldin	> 25	—	unknown	—	0/9	—
	Silhouette	50.2	0.0298	61.2	0.000224	3/9	10/10
iris	Calinski-Harabaz	> 25	> 25	unknown	unknown	0/6	0/10
	COP	17.1	> 25	26.4	unknown	6/10	0/10
	Modified Davies-Bouldin	3.58	—	10.1	—	7/8	—
	Silhouette	> 25	> 25	unknown	unknown	0/4	0/10
wholesale	Calinski-Harabaz	3.52	—	8.79	—	9/10	—
	COP	68.1	> 25	70.1	unknown	3/10	0/10
	Modified Davies-Bouldin	0.192	> 25	0.139	unknown	10/10	0/10
	Silhouette	23.4	> 25	30.2	unknown	5/9	0/10
seeds	Calinski-Harabaz	31.3	> 25	41.9	unknown	4/9	0/10
	COP	234	—	237	—	1/10	—
	Modified Davies-Bouldin	6.45	—	14	—	8/10	—
	Silhouette	20	> 25	26.6	unknown	6/10	0/10
ionosphere	Calinski-Harabaz	75.1	—	86.6	—	2/8	—
	COP	37.8	—	48.4	—	4/10	—
	Modified Davies-Bouldin	3.98	—	8.81	—	9/10	—
	Silhouette	50.5	> 25	61.2	unknown	2/6	0/10

applied to the algorithm runtime calculation. Let E_S be the average of runtime for successful runs, R be the ratio of successful runs, G be the maximum runtime until restart and D_S be the standart deviation of runtime for successful runs. Then the expectation of the runtime until success E is:

$$E = E_S + \frac{1-R}{R} G. \tag{12}$$

The standart deviation of the runtime D can be estimated by the equation:

$$D = \sqrt{E_S^2 - E^2 + D_S^2 + \frac{1-R}{R}(G^2 + GE)}. \tag{13}$$

However, the expected mean can be calculated only if the number of successful runs is at least 1. If for some setting we do not have any successful runs, we can only conclude that the mean runtime is at least the time limit and cannot say anything about its deviation. In Table 2 in such cases we write "> 25" for the mean runtime and "unknown" for its deviation.

We also exclude the results for the greedy algorithm from Table 2, since the valid runs of this algorithm were observed only in one setting.

From Table 2 we conclude that in most settings the algorithm with conventional CVI computation require much more time to find a clustering that is at least as good as the one found by the same algorithm with incremental CVI computation in 5 minutes. The few exceptions are highlighted in Table 2. We note that for three of these five exceptions deviation is relatively high, and the 5 minutes time limit given for the algorithm with incremental CVI computation lies inside the confidence interval.

For the other two exceptions (*Silhouette* index on the *glass* dataset and *modified Davies-Bouldin* index on the *wholesale* dataset) we noticed that it took no more than 166 iterations for the algorithm with conventional CVI to reach the desired value of measure. This implies that the algorithm with iterative CVI also finds such good clustering in a small number of iterations and then does not improve it until it reaches its time limit.

5 Conclusion

In this paper, we present and examine a new fast method of fitness function computation for evolutionary clustering algorithms. The proposed approach is based on gathering temporary data from CVI computation on previous iteration. It appears that our approach achieves better results than the conventional CVI computation

in most of the considered cases. The suggested method is planed to be tested on more sophisticated evolutionary strategies [15].

Acknowledgement: The methods for incremental CVIs computation were developed under the research project financially supported by The Russian Science Foundation, Agreement 17-71-30029.

References

[1] Kaufman, L. and Rousseeuw, P. J. 2009. *Finding groups in data: an introduction to cluster analysis*, vol. 344. John Wiley & Sons, USA.

[2] Jain, A. K. and Dubes, R. C. 1998. *Algorithms for clustering data*, vol. 6. Prentice hall, Englewood Cliffs, NJ, USA.

[3] Bigus, J. P. 1996. *Data mining with neural networks: solving business problems from application development to decision support.* McGraw-Hill, New York, USA.

[4] Mecca, G., Raunich, S., and Pappalardo, A. 2007. A new algorithm for clustering search results *Data & Knowledge Engineering* 62, 3, pp. 504–522.

[5] Anderberg, M. R. 1973. *Cluster analysis for applications: probability and mathematical statistics: a series of monographs and textbooks*. Academic press, USA.

[6] Bonner, R. E. 1964. On some clustering techniques. *IBM journal of research and development* 8, 1, pp. 22–32.

[7] Kleinberg, J. M. 2003. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems 15 (NIPS 2002)*. MIT Press, pp. 463–470.

[8] Arthur, D. and Vassilvitskii, S. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*.ACM, New Orleans, Louisiana, pp. 1027–1035.

[9] Ester, M. et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining – KDD 1996*. AAAI Press, Portland, Oregon, pp. 226–231.

[10] Chakrabarti, D., Kumar, R., and Tomkins, A. 2006. Evolutionary clustering. In *Proceedings of the 12th ACM international conference on Knowledge discovery and data mining – SIGKDD 2006*. ACM, Philadelphia, PA, pp. 554–560.

[11] Arbelaitz, O. et al. 2013. An extensive comparative study of cluster validity indices. *Pattern Recognition* 46, 1, pp. 243–256.

[12] Ng, A. Y., Jordan, M. I., and Weiss, Y. 2002. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems* 14, 2, pp. 849–856.

[13] Doer, B. and Doer, C. 2015. Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th. In *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2015*. ACM, pp. 1335–1342.

[14] Buzdalov, M. and Buzdalova, A. 2013. Adaptive Selection of Helper-Objectives for Test Case Generation. In *2013 IEEE Congress on Evolutionary Computation*. IEEE, pp. 2245–2250. DOI: 10.1109/CEC.2013.6557836

[15] Hruschka E. R. et al. 2009. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 2, 39, pp. 133–155.