DOI: 10.15439/2016F168
ACSIS, Vol. 9. ISSN 2300-5963

Correlation clustering: divide and conquer

László ASZALÓS*, Mária Bakó†
* University of Debrecen
Faculty of Informatics
26 Kassai str., H4028 Debrecen, Hungary
Email: aszalos.laszlo@inf.unideb.hu
† University of Debrecen
Faculty of Economics
138 Böszörményi str., H4032 Debrecen, Hungary
Email: bakom@unideb.hu

***Abstract*—The correlation clustering is an NP-hard problem, hence its solving methods do not scale well. The contraction method and its improvement enable us to construct a divide and conquer algorithm, which could help us to clustering bigger sets. In this article we present the contraction method and compare the effectiveness of this new new and our old methods.**

I. Introduction

CLUSTERING is an important tool of unsupervised learning. Its task is to group objects in such a way, that the objects in one group (cluster) are similar, and the objects from different groups are dissimilar. It generates an equivalence relation: the objects being in the same cluster. The similarity of objects are mostly determined by their distances, and the clustering methods are based on distance.

Correlation clustering is an exception, it uses a tolerance (reflexive and symmetric) relation. Moreover it assigns to each partition (equivalence relation) a cost, i.e. number of pairs of similar objects that are in different clusters plus number of pairs of dissimilar objects that are in the same cluster. Our task to find the partition with the minimal cost. Zahn proposed this problem in 1965, but using a very different approach [1]. The main question is the following: *which equivalence relation is the closest to a given tolerance (reflexive and symmetric) relation?* Many years later Bansal at al. published a paper, proving several of its properties, and gave a fast, but not quite optimal algorithm to solve the problem [2]. Bansal have shown, that this is an NP-hard problem.

The number of equivalence relations of n objects, i.e. the number of partitions of a set containing n elements is given by Bell numbers B_n, where $B_1 = 1$, $B_n = \sum_{i=1}^{n-1} \binom{n-1}{k} B_k$. It can be easily checked that the Bell numbers grow exponentially. Therefore if $n > 15$, in a general case we cannot achieve the optimal partition by exhaustive search. Thus we need to use some optimization methods, which do not give optimal solutions, but help us achieve a near-optimal one.

If the correlation clustering is expressed as an optimization problem, the traditional optimization methods (hill-climbing, genetic algorithm, simulated annealing, etc.) could be used in order to solve it. We have implemented and compared the results in [3].

This kind of clustering has many applications: image segmentation [4], identification of biologically relevant groups of genes [5], examination of social coalitions [6], improvement of recommendation systems [7] reduction of energy consumption [8], modeling physical processes [9], (soft) classification [10], [11], etc.

In a previous paper [12] we presented the contraction method with many different interpretations, and later we constructed an improvement for the contraction method [13]. In this paper we introduce a new method which is based on the contraction method and its improvement, and this new method is a divide and conquer algorithm. By the measurements the new method is not much worse than the old one, therefore the new method could help us to solve concrete problems with thousands of objects.

The structure of the paper is the following:

In Section 2 we define correlation clustering mathematically and shortly present the contraction method and its improvement. Section 3 describes the divide and conquer algorithms. Next, we show the results of the measurements, and in Section 5 the recursive variant of the method. Later we present the results according to Barabási-Albert random graphs. In Section 7 we give our plans and discuss the technical details. Finally we conclude the results.

II. Correlation clustering

In the paper we use the following notations: V denotes the set of the objects, and $T \subset V \times V$ the tolerance relation defined on V. We handle a partition as a function $p : V \to \{1, \dots, n\}$. The objects x and y are in a common cluster, if $p(x) = p(y)$. We say that objects x and y are in conflict at given tolerance relation and partition iff value of $c_T^p(x, y) = 1$ in (1).

$$c_T^p(x,y) \leftarrow \begin{cases} 1 & \text{if } (x,y) \in T \text{ and } p(x) \neq p(y) \\ 1 & \text{if } (x,y) \notin T \text{ and } p(x) = p(y) \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

We are ready to define the cost function of relation T according to partition p:

$$c_T(p) \leftarrow \frac{1}{2} \sum c_T^p(x,y) = \sum_{x<y} c_T^p(x,y) \qquad (2)$$

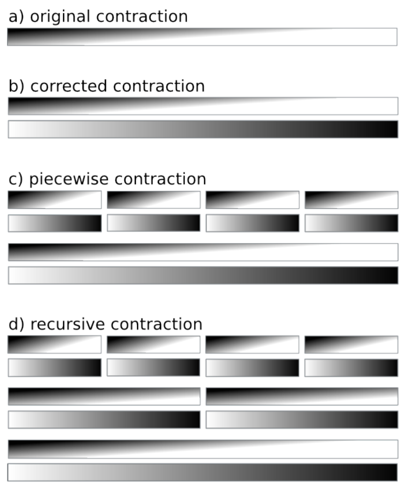

Fig. 1. Different variants of contraction method.

Our task is to determine the value of $\min_p c_T(p)$, and a partition p for which $c_T(p)$ is minimal. Unfortunately this exact value cannot be determined in practical cases, except for some very special tolerance relations. Hence we can only get approximative, near optimal solutions.

We can define the attraction between two objects: if they are similar then the attraction between them is 1; if they are dissimilar then the attraction between them is -1 (they repulse each other); otherwise—which can occur at a partial tolerance relation—the attraction is 0.

$$a(x,y) \leftarrow \begin{cases} 1, & \text{if } (x,y) \in T \\ -1, & \text{if } (x,y) \notin T \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

We can generalize (3) for object i and for clusters g and h:

$$a(x,g) = \sum_{y \in g} a(x,y) \text{ and } a(g,h) = \sum_{y \in h} a(y,g).$$

We leave it to the reader to check, that if these sums are positive and we join these element and clusters—by getting a partition p' containing the clusters $g \cup \{x\}$ or $g \cup h$—then $c_T(p) \geq c_T(p')$. This means that by joining attractive clusters, the cost decreases.

The contraction method starts with a partition where each cluster is a singleton. Next it selects clusters g and h for which $a(g,h)$ is maximal (and positive), and joins these clusters. It continues until there are no attractive clusters left. This contraction is presented on part a of Fig. 1 as a diagonally painted rectangle.

At dense Erdős-Rényi random graphs (ER in the following) each decision could have a vast impact, at the pure contraction method we cannot correct previous decisions—we just join, and not slit—, therefore we need something else in order to make a decision. This is the place, where the attraction between a node and a cluster is taken into consideration: we take every combination of nodes and clusters. If we find that a node is attracted much harder by any other cluster than by its clusters, then we put that node into the other cluster. Moreover if some node is repulsed by all the clusters, we construct an extra cluster for this node and we move it into this new cluster. This is the improvement of the contraction, that corrects the faults of the contraction. This correction step on part b of Fig. 1 denoted as a vertically painted rectangle. This correction step follows the contraction step. Here we apply both steps for the whole V i.e. for all objects together.

We discussed the properties of the contraction and this improvement in [13].

III. Divide and conquer algorithms

The divide-and-conquer strategy solves a problem by:

- breaking it into sub-problems that are themselves smaller instances of the same type of problem—*divide*
- recursively solving these sub-problems—*conquer*
- appropriately combining their answers—*combine*

One may think that only the second step (conquer) could be hard, in general each step needs some work: remember the splitting of the array at quick-sort for the divide step, and the merging of the ordered sequences for the combine step of the merge sort.

As practice has shown, we can amend the outcome of the contraction with its improvement, so we treat the contraction and its improvement as a unit. This is the reason why part c and d of Fig. 1 are built from units (of contraction and correction) of part b.

At a correlation clustering problem we have a set V of n objects, so the division is simple: split the whole set into two subsets of $n/2$ objects (or three subsets of $n/3$ objects, and so on).

The subsets are similar to the original sets, so at the conquer step we only need to apply the unit for each subset independently, as first half of part c of Fig. 1 presents.

The outcome of correlation clustering is a partition, so the outcome of the correlation clustering of the subsets are independent partitions as Fig. 2 shows. At the combine step of the algorithm we need to check whether the clusters of the partitions of the subsets could be parts of a cluster, which is member of the partition of the whole set. Fortunately we have a tool to answer this question, the contraction does exactly this: checks whether two cluster could be joined. Hence we need to apply the unit of contraction and correction for the whole set V, as the second half of part c of Fig. 1 presents.

Based on the algorithm of the contraction (`contract`) and its improvement (`correct`) we can easily implement in Python this divide and conquer algorithm, as Alg. 1 shows. Here `uf` is the UnionFind data structure that handles the data of the partition. `self.size` stores the number of objects in V, and `no_parts` denotes the number of subsets. The contraction and its improvement were implemented in such a way, that they get the subsets of the objects with their (`lower` and `upper`) bounds. Whilst they only use one data structure to store attraction between clusters and objects, they handle these data independently for different subsets of objects.

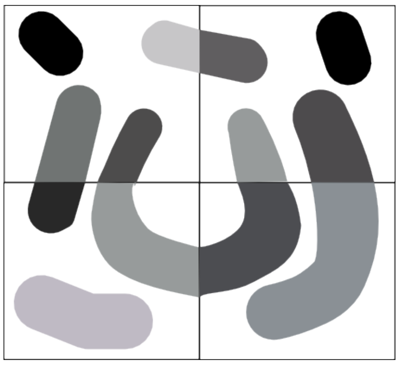

Fig. 2. Outcome of partitioning the subsets

Algorithm 1 Corrected contraction

```
def contract_in_pieces(self, uf, no_parts):
    sub_size = math.ceil(self.size/no_parts)
    for lower in range(0, self.size, sub_size):
       upper = min(lower + sub_size, self.size)
       self.contract(uf, lower, upper)
       self.correct(uf, lower, upper)
    self.contract(uf, 0, self.size)
    self.correct(uf, 0, self.size)
```

In the last two lines of Alg. 1, we combine the clusters of the subsets by applying the contraction to the whole set of objects. (We note that Python starts indexing with 0, and at defining intervals, the upper limit is exclusive.)

The implemented methods and the testing environment is available at https://github.com/aszalosl/DC-CC.

IV. Effectiveness of the new method

At the first measurements we use traditional tolerance relations, i.e. the signed graph of the problem is total. The structure of the relation—or the structure of the graph—determines the cost $c_T(p)$, e.g. despite that two graphs have the same number of negative and positive edges, one graph belongs to an equivalence relation—so its cost is 0—, while the cost of the other graph is a large number. It is extremely unlikely that we will obtain an equivalence relation by randomly generating a tolerance relation, and by our former experiments related to total graphs the deviation of the costs is small.

Although the number of positive and negative edges does not precisely describe the graph (and hence the relation), it helps us to understand the processes. We will use the rate calculated by the number of positive edges and the number of all edges, and denote this ratio by q. On Fig. 3 we present rates of different $c_T(p)$'s as a function of q (of T), where the p is the partition that the algorithm generates from T. We used 100 random relations with the same q, and we display the mean here. The base is the contraction method without any improvement/correction, this belongs to the level 1.0. The solid line denotes the contraction with its improvement. As the solid line does not exceed the level 1.0, it really is an improvement.

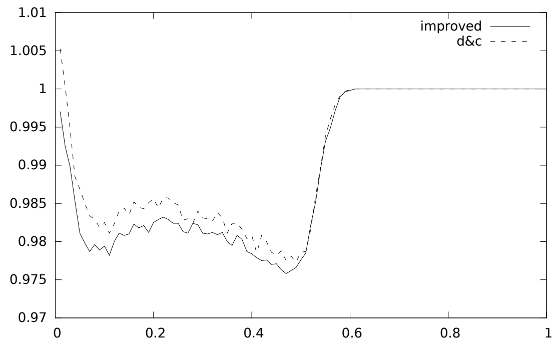

Fig. 3. Comparison of the original, the improved and the divide and conquer method with 10 subsets. (Less is better.)

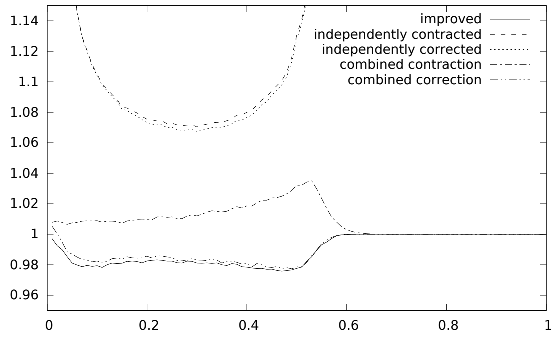

Fig. 4. Results of different steps of the piecewise contraction.

The dashed line denotes the outcome of the divide and conquer algorithm when we split the original problem into ten sub-problems. As it is slightly above the solid line, it almost reaches the level of the improved algorithm.

Fig. 4 goes into details. The solid line below shows the result of the corrected contraction, where we applied both process for the whole set V. On the other hand at the fist stage of the piecewise contraction we need to execute the contraction (in parallel) for the subsets of V. As the graph of the problem is a total graph, i.e. everything is connected, the attraction of the objects in different subsets suffers from cost function, so its dashed line is at the top. The piecewise correction (second stage) is an improvement, so its dotted line is below the previous line. The clusters of the partitions of the subsets could be joined at the next stage, which a total contraction—we apply it for the whole set V—and with this we reach the level 1.0 somewhere. And the last stage—the total correction—get below this level, and approaches the improved contraction.

A huge sample could smooth the lines, but we do not believe that using big samples gives clearly answer which parameter is the best for Fig. 5. Here we compare the results of dividing the original set into different number of subsets. The lines intersect each-other several times on the graph, therefore we cannot announce the absolute winner, maybe the biggest parameter gives the best (smallest) result.

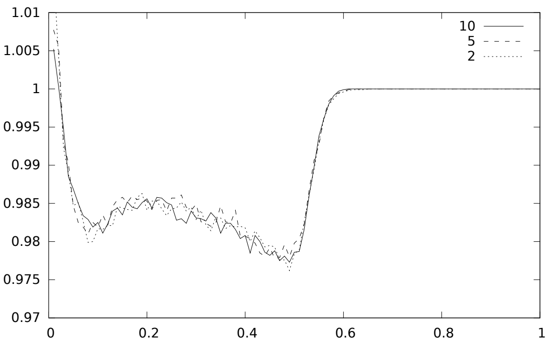

Fig. 5. Comparison of the divide and conquer method with different number of subsets.

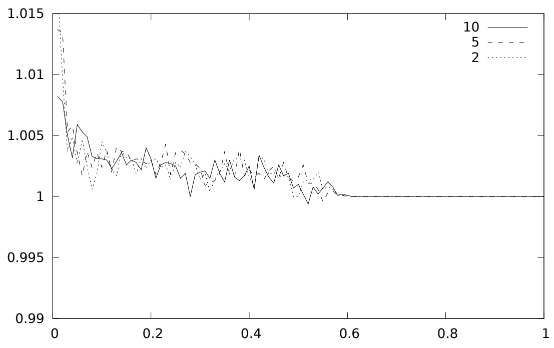

Fig. 6. Comparison of the divide and conquer method with different number of subsets.

Does the improved or the divide and conquer method give better result? We created Fig. 6, to show the rate of their result in each case. As the lines are mostly above the level 1.0, the divide and conquer method gives a bit worse result, than the improved algorithm. This is a reasonable result: if we have less nodes, we only know half of the information and hence our decisions are less sound and we make more mistake. We tested 100 complete graphs each with 200 nodes and divided them into 10-element sub-graphs, and when we compared the result, strangely the divide and compare was better with one percent than the improved contraction alone.

If we allow partial tolerance relations, the diversity increases. Fig. 7 uses the same number of objects, but the graph of the relation is an ER graph with different ps. As there are less edges left, they could create a more complex structure than in a total graph, so the variance is bigger than before, and this increases as the parameter p decreases. However, the situation itself is similar, so the improved method is slightly better than the divide and compare one.

V. Recursive algorithm

If we want to cluster 10^6 objects, it does not help us to cluster ten set of objects with 10^5 members in each, although previous experiments suggest that less subsets are better. Most of the divide and conquer algorithm uses recursion, let us apply it for this problem! Alg. 2 receives an UnionFind structure

Algorithm 2 Recursive contraction

```
def rec_c(self, uf, l, lower, upper):
    if lower + l < upper:
        mid = (lower + upper)//2
        self.rec_c(uf, l, lower, mid)
        self.rec_c(uf, l, mid, upper)
    self.contract(uf, lower, upper)
    self.correct(uf, lower, upper)
```

to store the clustering, the maximal length l of a primitive cluster—that will not be divided further—, and the bounds of the subset. If the size of the cluster is bigger then l, we divide it into half, and recursively call both parts. Next, we apply the contraction, and later the correction for the whole subset.

Last part of Fig. 1 shows, that we apply the contraction and corrections for quarter sets, next on the half sets, and finally the whole set. The parameter l enables us to try several values, as Fig. 8 shows. It may be hard to read from the picture, but we believe that smaller values give smaller cost, hence are better. We have calculated the sum of the rates for a big sample and this sum is less for smaller ls, so we get closer to the result of the improved contraction from above. Taking a look at the last part of Fig. 1, we can say, that is contains numerous corrections. Can we omit all of them but the last? (We need to delete Line 7 from Alg. 2, and put it into the main program.) Yes, we can, but the result is poor as Fig. 8 shows.

The difference between piecewise and recursive contraction is small, hence further research is needed to decide which one could help us to cluster large sets.

VI. Sparse random graphs

Until now we examined ER graphs. In nature, this kind of graph is not common, so we take a look at other types of graphs. The other well-known random graph type is the Barabási-Albert (BA in the following). At generating this type of graphs we use preference attachment. Here each new node is connected to existing nodes with a probability that is proportional to the number of links that the existing nodes already have. This means, that the oldest nodes construct a relatively dense graph, and the young nodes connect them

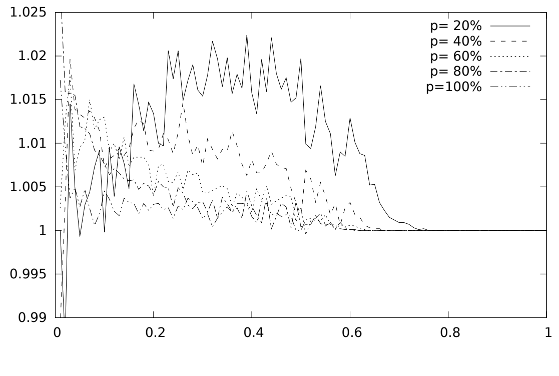

Fig. 7. Contacting Erdős-Rényi random graphs with different probabilities.

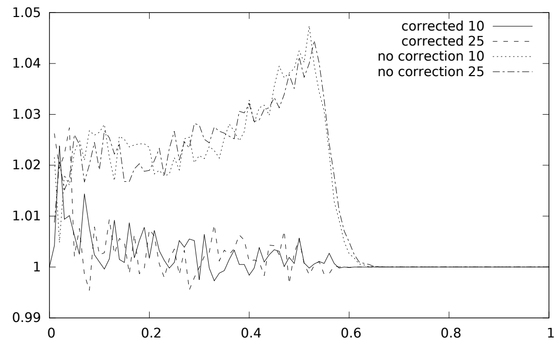

Fig. 8. Different lengths at recursive contraction and at a variant.

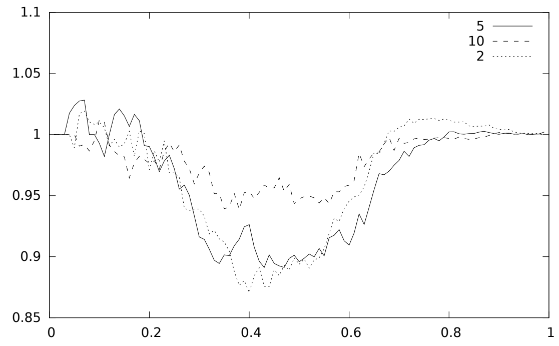

Fig. 9. Different number of subsets at clustering BA graphs.

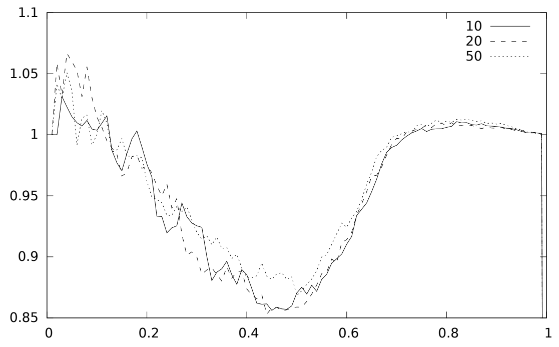

Fig. 10. Different primitive sizes at clustering BA graphs.

TABLE I
RUNNING TIME OF DIVIDE AND CONQUER CLUSTERINGS.

size	number of parts				max size of primitive cluster			
	2	5	10	20	10	20	25	50
ER graphs								
50	0.82	0.74	0.72	0.73	0.74	0.76	0.81	0.96
100	0.69	0.59	0.56	0.55	0.56	0.57	0.61	0.69
200	0.52	0.39	0.35	0.33	0.34	0.35	0.37	0.41
BA graphs								
500	0.92	0.91	1.08	1.20	0.78	0.77	0.77	0.76
1000	0.91	0.90	1.07	1.21	0.74	0.73	0.73	0.73
2000	0.89	0.89	1.06	1.21	0.71	0.70	0.70	0.70

weakly. By slicing the set of objects into subsets based on their (serial) numbers, we put the oldest nodes into the same subset. Applying our methods to these kind of subsets we get very similar results as before. To discover the real tendencies, we mixed up the numbers of the objects, and used the simplest slicing.

Our previous research had shown that the clustering of ER type and BA type random graphs gives different results and different tendencies [13]. Therefore we repeated our previous tests for these graphs, too. As sparse graphs have only a few edges, here we used 500 objects and 3/2 type BA graphs. Fig. 9 shows that strange finding, that the cost for five subset is minimal, meaning that for two or ten subsets the cost is higher. We have used different samples in this tests, hence we need test the same graphs to get more reliable results. Fig. 10 shows that smaller primitive subsets give better result, but we tested this on different samples too. The variance is big at BA graphs, so it is better to use the same graphs here.

Some tendencies are the same as at ER types, but the rate of the improved contraction and piecewise/recursive contraction is the opposite. At ER graphs the improved contraction is slightly better, but at BA graphs the the latter methods produce significantly better results.

VII. TECHNICAL DETAILS

We have not yet mentioned the speed of the methods. Of course, the parameter q influences the speed. If q is low, then there is little chance of contraction, so at the combine step of the algorithm we have almost the same number of clusters as at the beginning. This means, that we do the same task twice. If q is high, then most of the objects are contracted into few clusters, and contracting them does not take much time. At the clustering of the subsets we need to cope with smaller complexity, therefore we hope that at high q-s the divide and conquer algorithm could run faster than originally.

We did not perform very detailed analysis for each value of q, but we calculated the mean of the running time with different parameters. Even this shows the tendencies, and determines the future research directions.

Table I shows the rates of running times of the different divide and conquer algorithms. The base is the running time of the improved contraction method, and if the number is less than 1.0, then the divide and conquer algorithm is faster.

At the two different kinds of random graphs the algorithms behave differently. At ER graphs as we divide the original set into more and more parts, the running time decreases. At BA graphs more and more parts require extra overhead, and eventually its running time becomes higher than the original method's.

If we use a recursive method, the calculations speed up at ER graphs with smaller sets. At two hundred nodes, we can reduce the running time to its third. We hope, that at bigger sets we can save even more time. At BA graphs we can reduce the running time too. Unfortunately, it does not speed up as much as the ER graphs. Here the more level of recursion does not help, moreover it has some overhead, so the running time is longer than at smaller level of recursions.

Do not forget, that BA graphs have $O(n)$ edges, while ER graphs have $O(n^2)$ edges, and this property holds for its subgraphs. To work with $k \cdot c \cdot \left(\frac{n}{k}\right)^2$ instead of $c \cdot n^2$ is better, while there is no real difference $k \cdot c \cdot \left(\frac{n}{k}\right)$ and $c \cdot n$.

The former property holds for ER graphs with probability $p \neq 1.0$, hence the rate running time of divide and conquer clusterings is very close to the numbers in the table.

For UnionFind data structures there is an excellent implementation: disjoint-set forests. Unfortunately this implementation does not contain the *replace* operator, which is necessary to correct the errors of the contractions. There are algorithms which enable the deletion [14], but this is a logical deletion and not a physical one, which would be needed to replace the old value with a new one.

Our implementation currently uses an array assigning the identifier of a cluster to each object, and an associative array assigning the cluster to an identifier. In this case, the operation union have linear complexity according to size of the smaller cluster, and the other operations have constant complexity.

Our implementation caches values $a(x, g)$ and $a(h, g)$ for every valid x, g and h. As these values change at contraction and at correction; we need to update the stored values. This requires many small tricks, but is much faster than the version calculating these values again and again.

VIII. Conclusion and further work

Based on our contraction method and its improvement we constructed a divide and conquer algorithm. Our hypothesis was that it gives poor results, because it uses less information, than the original method, but in some cases runs faster than the original algorithm. Surprisingly, the new method generates result close the improved variant of the former method at ER graphs, and gives better result for BA graphs. The former data structures and its methods are not usable for us, so we applied less sophisticated algorithms. The similar results and a faster calculation suggests, that this could be a fruitful direction. We need a more detailed comparison of methods, to discover the limits of this algorithm; and extra work to implement the parallel version, which uses the advantages of the divide and conquer method.

References

[1] C. Zahn, Jr, “Approximating symmetric relations by equivalence relations,” *Journal of the Society for Industrial & Applied Mathematics*, vol. 12, no. 4, pp. 840–847, 1964. [Online]. Available: http://dx.doi.org/10.1137/0112071

[2] N. Bansal, A. Blum, and S. Chawla, “Correlation clustering,” *Machine Learning*, vol. 56, no. 1-3, pp. 89–113, 2004. [Online]. Available: http://dx.doi.org/10.1023/B:MACH.0000033116.57574.95

[3] L. Aszalós and M. Bakó, “Advanced search methods (in Hungarian),” http://morse.inf.unideb.hu/~aszalos/diak/fka, 2012.

[4] S. Kim, S. Nowozin, P. Kohli, and C. D. Yoo, “Higher-order correlation clustering for image segmentation,” in *Advances in Neural Information Processing Systems*, 2011, pp. 1530–1538.

[5] A. Bhattacharya and R. K. De, “Divisive correlation clustering algorithm (dcca) for grouping of genes: detecting varying patterns in expression profiles,” *bioinformatics*, vol. 24, no. 11, pp. 1359–1366, 2008. [Online]. Available: dx.doi.org/10.1093/bioinformatics/btn133

[6] B. Yang, W. K. Cheung, and J. Liu, “Community mining from signed social networks,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 10, pp. 1333–1348, 2007.

[7] T. DuBois, J. Golbeck, J. Kleint, and A. Srinivasan, “Improving recommendation accuracy by clustering social networks with trust,” *Recommender Systems & the Social Web*, vol. 532, pp. 1–8, 2009. [Online]. Available: http://dx.doi.org/10.1145/2661829.2662085

[8] Z. Chen, S. Yang, L. Li, and Z. Xie, “A clustering approximation mechanism based on data spatial correlation in wireless sensor networks,” in *Wireless Telecommunications Symposium (WTS), 2010*. IEEE, 2010, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/WTS.2010.5479626

[9] Z. Néda, R. Florian, M. Ravasz, A. Libál, and G. Györgyi, “Phase transition in an optimal clusterization model,” *Physica A: Statistical Mechanics and its Applications*, vol. 362, no. 2, pp. 357–368, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.physa.2005.08.008

[10] L. Aszalós and T. Mihálydeák, “Rough clustering generated by correlation clustering,” in *Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing*. Springer Berlin Heidelberg, 2013, pp. 315–324. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2007.1061

[11] L. Aszalós and T. Mihálydeák, “Rough classification based on correlation clustering,” in *Rough Sets and Knowledge Technology*. Springer, 2014, pp. 399–410. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-11740-9_37

[12] L. Aszalós and T. Mihálydeák, “Correlation clustering by contraction,” in *Computer Science and Information Systems (FedCSIS), 2015 Federated Conference on*. IEEE, 2015, pp. 425–434.

[13] L. Aszalós and T. Mihálydeák, “Correlation clustering by contraction, a more effective method,” to appear in Studies in Computational Intelligence.

[14] S. Alstrup, M. Thorup, I. L. Gørtz, T. Rauhe, and U. Zwick, “Union-find with constant time deletions,” *ACM Transactions on Algorithms (TALG)*, vol. 11, no. 1, p. 6, 2014.