Communication Papers of the Federated Conference on Computer Science and Information Systems pp. 21–28
DOI: 10.15439/2019F93
ISSN 2300-5963 ACSIS, Vol. 20

Correlation Clustering: Let All The Flowers Bloom!

László Aszalós
Faculty of Informatics at University of Debrecen,
26 Kassai út H4028 Debrecen , Hungary
Email: aszalos.laszlo@inf.unideb.hu

Mária Bakó
Faculty of Economics at University of Debrecen,
138 Böszörményi str., H4032 Debrecen, Hungary
Email: bakom@unideb.hu

***Abstract*—Correlation clustering is a NP-hard problem, and for large signed graphs finding even just a good approximation of the optimal solution is a hard task. In this article we examine the effect of ranking the nodes and processing them in order of ranks. We demonstrate that based on the rate of positive edges in the graph it is worth using different optimisation methods. We show that all building blocks of our methods are necessary under certain circumstances.**

I. Introduction

ONE OF the typical tools of unsupervised learning is cluster analysis. Here, we wish to put similar elements into the same group, and put different elements into different groups. The clustering methods are usually based on the distance or density of the objects. There is an exception—correlation clustering—where we only have a similarity relation of the objects. This is a tolerance relation, i.e. reflexive and symmetric, but not necessarily transitive. The result of a clustering is a partition of objects, which can be understood as an equivalence relation, i.e. a reflexive, symmetric and transitive relation. In 1965 Zahn asked the following question [14]: *What is the closest equivalence relation for a given tolerance relation?*

In 2004 Bansal et al. have showed that this task is NP-hard and gave a non-optimal solution [6], which of course only produces an approximate solution. [6] also contains a generalization of the original problem, where the edges have weights, and instead of distances we want to optimize the sum of weights of *misplaced* edges. The generalized problem has many applications (mainly in image processing), and there are numerous approximation methods available in the literature, each based on a different approach: from minimal cut to linear programming [1], [8], [9], [10], [11], [13].

The clustering problem itself can be formulated as a combinatorial optimisation task, and therefore we can apply the usual methods: hill-climbing, scattered search, bee algorithm, harmony search, etc. In recent years, the authors have examined most of these [2], [5] and moreover have developed a new method (contraction) to solve the original clustering problem. Various implementations of this new method were born, on one hand to accelerate the methods, and on the other to achieve as close an approximation as possible.

Obviously, a general method cannot offer the best approximation in every particular case. Therefore, we can achieve a better approximation than the methods described above if we choose the most fitting method for each individual task. The contraction itself is a universal method, i.e. it can be used for all tolerance relations; however, implementations can be developed to fit the specialities of the given problem to be solved. As there are no widely accepted benchmark tolerance relations, we use random, signed graphs to test our methods. The random signed graphs can be easily identified by partial tolerance relations. (A positive edge denotes that the relation holds between the nodes of the edge; a negative edge denotes that the relation does not hold between the nodes of the edge; a missing edge means partiality, that is we have no information about similarity or dissimilarity.) Two types of random graphs are common. The Erdős-Rényi graph is a dense graph, where there is an edge between any pair of nodes with probability p. For any random *colouring* of the same graph—that is we assign positive or negative signs to the edges—the distance between the tolerance relation (which belongs to the signed graph) and the closest equivalence relation is almost the same due the symmetry of this kind of graphs. This means that the colouring does not affect the distance between the tolerance and the equivalence relations. Phase transformation of correlation clustering—determining the size of the biggest cluster in the nearest equivalence relation—can be solved relatively easily [12].

Barabási-Albert's scale-free—later abbreviated to BA-type—graph is a sparse graph, so the number of edges will be proportional to the number of vertices. Because there are only a few edges in the graph, it follows that usually there are only a few adjacent vertices, therefore the *colouring* of a given edge has a much stronger effect as before. Furthermore, these kind of graphs are not symmetric. Hence the distance of tolerance relations correspond to the random colourings of the same graph and the closest equivalence relation can vary within wide limits, as Fig. 1.3. in [3] demonstrates this. There is not yet a conjecture for the phase transformation problem for BA-type graphs. Measurement results for large graphs could possibly help to formalise this conjecture. This is the reason why we wish to develop a both efficient and accurate method for BA-type graphs, and determine which methods are the best for certain conditions.

The structure of the article is the following: after the introduction, in Section II we present our notations including the elements needed to build our methods. Next we delineate the overall results, and review our experiments. In Section IV we discuss the results and the roles of the building blocks of our methods. Finally we conclude our results and we give our

further plans.

II. System of notation

A. Tolerance and equivalence relation

Let a set of objects be denoted by V, and their similarity (tolerance) relation by T, where $T \subset V \times V$. The partition p is a function: $p : V \to \mathbb{N}$, where objects x and y are in the same cluster if and only if $p(x) = p(y)$. Accordingly we can interpret the cluster of x as $g_x = \{y \in V | p(x) = p(y)\}$. Of course, the partition p determines an equivalence relation *where elements are in a common cluster in* p.

By the distance between the tolerance relation T and the equivalence relation based on p we mean the number of cases—called as a *conflict*—in which exactly one of the two relations holds, i.e.

- xTy and $p(x) \neq p(y)$ or
- xTy does not hold, but $p(x) = p(y)$.

Our task in case of a given tolerance relation T is to determine the partition p for which the distance between T and p is minimal, or in other words the number of conflicts is minimal. The exponentially growing Bell numbers [7] give the number of partitions, so the exhaustive search—apart from some special cases—is not applicable, hence we can only get approximate solutions.

Although it is not feasible in reality, but the objects can be thought of as magnets, where the similar ones are attracted to each other, and the different ones toss each other. If we left these magnets alone, after some time they would form a state of equilibrium. Here the magnets construct groups. These groups can be considered as clusters and the whole collection of clusters as partitions. The steps of the magnets we want to model are:

- another magnet group is more attractive for a magnet than its own group, so it is moved to that group
- two magnet groups attract each other, so these groups are joined.

The first step is to be later referred to as *motion*, while the other as *merging*. The value of attraction between objects x and y—denoted by $a(x, y)$—will be:

- 1, if the relation holds between them, i.e. the objects are similar,
- -1, if the relation does not hold between them, i.e. the object are dissimilar,
- 0, if the relationship is not interpreted between the two objects.

The attraction $a : V \times V \to \mathbb{N}$ of objects can be generalized as the attraction between an object and a cluster, that is $a : V \times 2^V \to \mathbb{N}$ and as the attraction between two clusters, that is $a : 2^V \times 2^V \to \mathbb{N}$ which are defined as $a(x, g) = \sum_{y \in g} a(x, y)$ and $a(g, h) = \sum_{x \in g} a(x, h)$, where g, $h \subset V$.

The summing here corresponds to the superposition known from physics. We left to the reader to check that at the motion and merging steps, the distance between T and the partition that is changed by either of these two steps will only decrease. Therefore, if we can no longer reduce the number of conflicts using motion and merging, then the resulting partition is actually a local minimum, and we stop.

B. Optimization methods

It is easy to check that for $V = \{a, b, c\}$ if aTb and aTc holds but bTc does not, then more local minima exist. In this simple case the number of conflicts (the distance) at these local minima are the same. In case of thousands of objects, the situation becomes very complicated, and many of the local minima aren't global minima, moreover there are significant differences between distances at minima.

The question is, in what order do we take our steps to avoid these traps, and get close to the global minima? [3] presented three structures, where the accuracy of the last one was up to twelve percent worse than the others'. So we omit this structure from this article, and so it is not shown in Fig. 1. First, we reiterate the other two structures. Then, we look at the different implementations of the merging and motions steps, and finally look at how these come together into methods.

In the case of the upper structure in Fig. 1—called *sequential*—a motion step is followed by a merging step, and if at least one of them can be executed, then the partition changes, and so we can try to reduce the conflicts again. If the partition has not changed during the motion and merging steps, our algorithm stops. In case of lower structure—called *repeated*—we apply the motion step until it reduces the distance between T and p. Once the motion step can no longer change the partition, we begin to merge. We repeat the merging step until it reduces the distance between T and p. If at least one merging step has made a reduction to the distance (denoted in the Fig. 1 by `any?`) in this cycle, that is, if the partition has been changed by some merging steps, then we can begin a new round, and test the motion step again. When the motion and merging steps do not change the partition, we must stop.

Whilst the motion step only changes the partition locally, and in small increments; the merging step—especially in the final stages of the optimisation, when we have big clusters—can cause significant global changes. This is the reason why the third structure—which is similar to the *repeated* structure, with the position of the motion and merging steps swapped—results in larger distances in the tests because the method reached a local minimum much earlier.

C. Alternative implementations of the motion step

To determine the motion step, the forces of $a(x, g)$ should be calculated for each object x and each cluster g, and for each object x we need to find its most attractive cluster h for which $a(x, h) = \max_g a(x, g)$. If the object x is moved to another cluster h—because that cluster is more attractive for the object than its own cluster, i.e. $a(x, h) > a(x, g_x)$—object x may become a status quo breaker there, and an earlier element of this new cluster is transferred to a third cluster and this chain reaction could continue. However, the previously calculated values of attractiveness of objects and clusters will be invalidated by changing the partition and need to recalculated.

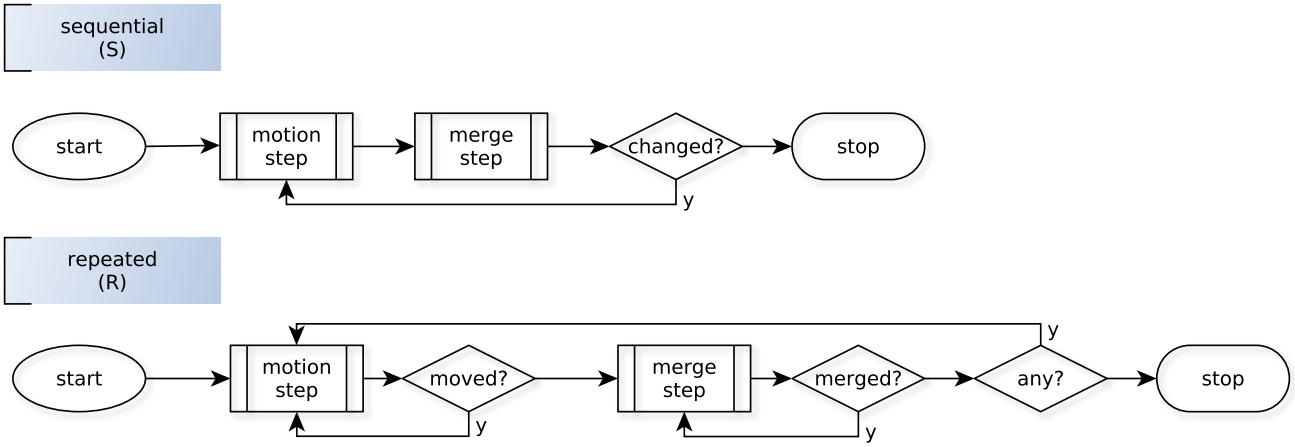

Fig. 1. Structure of our optimization methods.

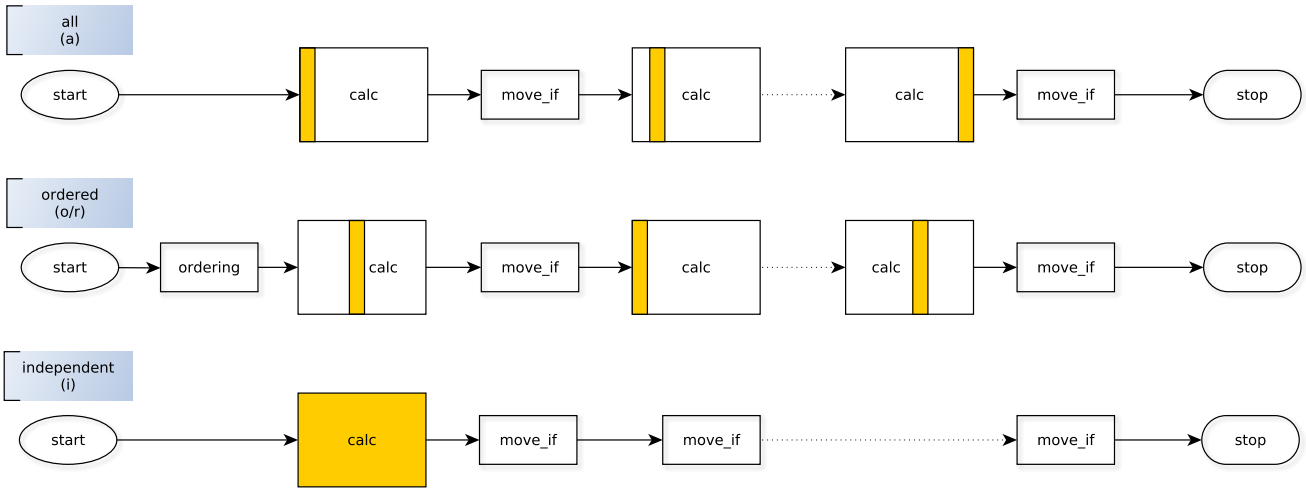

Fig. 2. Alternative flows of motion step.

We had an implementation of the motion step (flow *all* in Fig. 2) that takes all the objects one-by-one, and calculates the actual attractions of the objects. If some other (maybe empty) cluster is more attractive than its own cluster, it moves the object into that cluster. In the figure the colouring of the portion of the rectangle represents the range of the calculation. Here, we take every object $x \in V$ and calculate the values $\max_g a(x, g)$ and $a(x, g_x)$. The latter gives the attraction of its own cluster. If the former is larger - that is, another cluster has a greater attraction, then object x must move. Because we examine each object in a row, this process received the name *all*, and the abbreviation a. In the figure, the colouring of the corresponding rectangle shows that you do not need to compute attractions $a(y, g)$ for all objects y and for all clusters g, but only the ones that belong to object x: $a(x, g)$ for all clusters g. This determines the movement of x, and so we can move to the next object in the queue, and determine the attractions belonging to it.

The flow *independent* in Fig. 2—as the colouring shows—starts by calculating all the attractions $a(x, g)$ for all $x \in V$ and all clusters g. (That is why the entire rectangle is painted.) Next it determines the set

$$M = \{x \in V \,|\, \max_g a(x, g) > a(x, g_x)\},$$

i.e. the set of objects to move. In each step we need to select some object from the set M. If this object x moves from cluster g to the cluster h, then the set M in the next step will be reduced to $M \backslash (g \cup h)$ because the clusters g and h have changed, therefore the values previously calculated for these clusters are no longer valid and therefore should not be used. As the set M becomes empty, the process stops. Since the clusters affected by the motions (both from and to) are all different due to the deletions, the flow has been assigned the name *independent* and the abbreviation i.

We began to research the representatives of the clusters, and the methods to specify them [4]. To find the representatives of the clusters, a (initially unit) vector has to be multiplied by the matrix of the similarity relation and normalised again and again until a fixed point is obtained. In order to keep the running time under control, only a fixed number of steps were performed and the objects were ranked according to the result vector. Considering this rank, we can create two variants of the flow *all*, where this rank determines the order of processing, and not the (arbitrary chosen) identifiers of the objects. If we process objects in increasing order, we get the flow *ordered*, abbreviated by o, while if we choose the descending order, then flow *reversed*, or shortly r.

D. Alternative implementations of the merging step

For the merging step we use the attractions between clusters: $a(g, h)$. As merging causes a global change, a cautious approach allows merging two clusters only once. We are using the greedy approach, so we are interested in the most profitable merging, i.e. merge those two clusters for which the number of conflicts decreases the most. This is illustrated by the flow *one* in Fig. 3, and we will refer to it by O.

It is clear that the attractions of the merging clusters towards a third cluster will add up, so the attractions of the clusters can be maintained relatively easily. The flow *recalculate* (abbreviated by R) visualises this process. It is important to note—and this is also indicated by the colouring of the individual rectangles—that there is no need to recalculate all the attractions, therefore we can save ourselves from performing a lot of calculations by using this variant, instead of repeating flow *one*. Although the final result is exactly the same, the amount of calculation needed in the second case is only a fraction of the original.

The recalculation of forces may be omitted if we take the value of attractions in decreasing order, and we merge the clusters only if they have not yet changed in this process. This variant is called *independent*, and abbreviated by I.

III. Test results

A. Overall results

The combination of the structures and the implementations of the motion and merging steps provide an applicable method. In the names of the methods the first letter is the code of the structure (Fig. 1). The middle letter of the name of a method is the code of the motion step (Fig. 2), and the last letter is the code of the merging step (Fig. 3). Similarly to [3] we created a graph showing the results achieved with different methods.

We tested the various methods with 100 different BA type randomly generated tolerance relations of 5000 objects. We used the same tolerance relations as input for each method, and summed up the resulting conflicts and the running times of our Python implementations. These sums are presented in Fig. 4. Note that, the axis y has a logarithmic scale in order to equally allow for the demonstration of the fast and the slow methods.

The triangles in the upper left corner denote methods using the single merger (O). This location means that these are very accurate methods, but they can take up to ten times more time to run than the other two kinds of merging steps. In the lower right corner of the figure, crosses denote the methods applying the independent merger (I), these are fast but inaccurate.

We can find a coterie on the left side of Fig. 4. From left to right the pairs SaR-RaR, SrR-RrR and SoR-RoR overlap. While method RiR is above these (red circle), its pair, the SiR (black circle) is on the right side of the figure.

Based on this graph, we could think that the two new motion steps (r and o) are worse than the previous ones (a and i). Keep in mind that this figure was created by summing up the number of conflicts and running time, which can easily cover important details.

B. Detailed results

Let us consider a more sophisticated approach. We took 1200 different BA-type random tolerance relation of 500 objects, and examined 101 different colourings for each of them. Here the rate q of positive edges varies from 0 to 1, and we took the average of conflicts for each ratio.

From the slow methods, only SaO was kept in this test and it is represented with a triangle in the top left corner of Fig. 4, because it minimised the number of conflicts and therefore we consider it is as a benchmark. Since we approached the same distances with each of the 17 optimization methods, we would not really see any striking differences if we were to present the result on a typical graph. For this reason, we divide all the values by the corresponding values of the benchmark (Fig. 5). We can separate some of the methods, but most of them are very close to each other. However, it can be seen that some lines cross the level of 1, giving better results than the benchmark. We have no method that is always better than the benchmark, but we can find ones that are better for a small q, and ones are better for a big q.

For example method SaR can only tighten the benchmark around $q = 1$. The only difference between the methods SaR and SaO is that the latter merges once before attempting to move all the elements again, while the former continues to merge as long as possible. If only a few clusters with large number of elements constitute a partition, then there is no problem with successive merging, but in other cases this step causes too much change that cannot be counterbalanced by motion steps.

Consider the method RiR, which is dominated by the above-mentioned method SaR in the Pareto sense in Fig. 4, but rates of Fig. 5 shows that in many cases it exceeds the SaO method for both small and large q values. Therefore the Pareto set of Fig. 4 is not enough in itself to provide a method for finding the closest partition to a given tolerance relation.

Instead of presenting additional individual graphs, we made two summary figures (Fig. 6 and 7), where $q = 0$ corresponds to the left of the picture and $q = 1$ to the right. The methods with worst performance were not included in these figures. Fig. 6 shows the cases (method and q) which overtake the

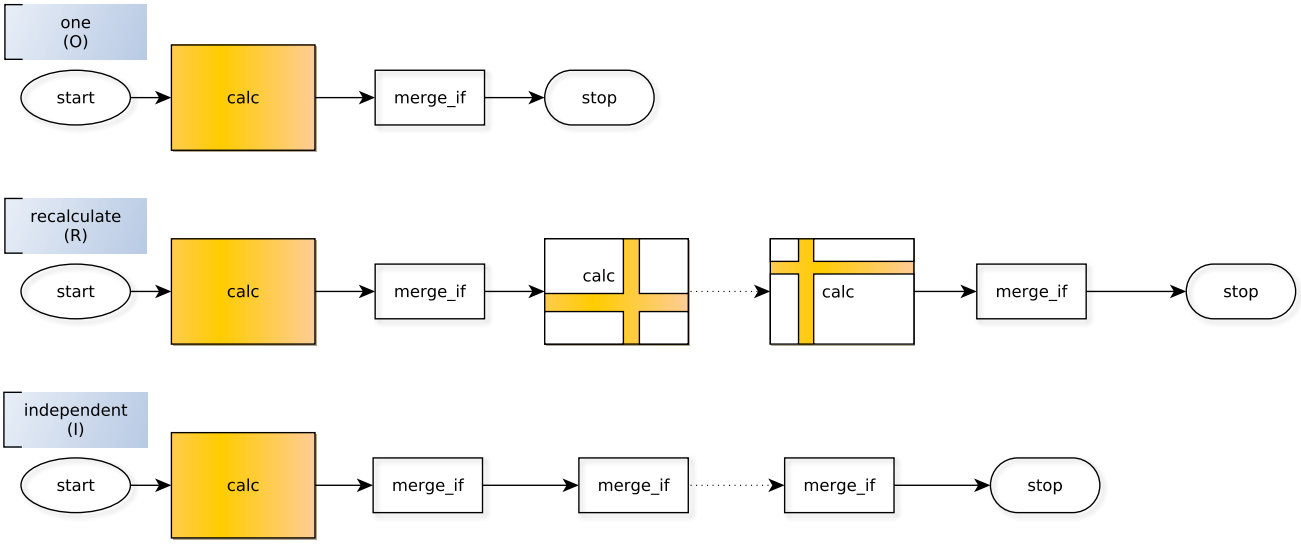

Fig. 3. Alternative implementations of merging step

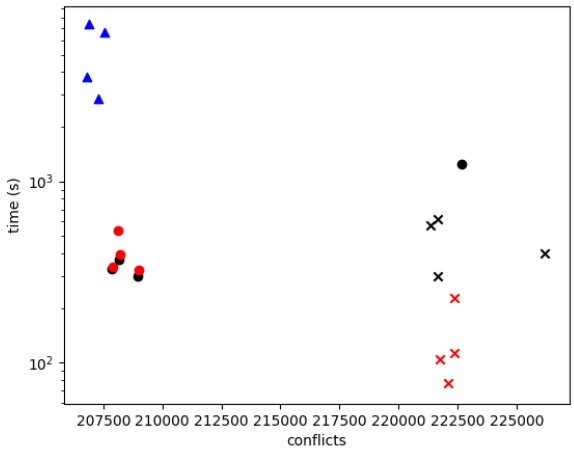

Fig. 4. Cumulative results on random relations: running time and efficiency of the variants of contraction. Shape and colour denote the merging step and the optimization method

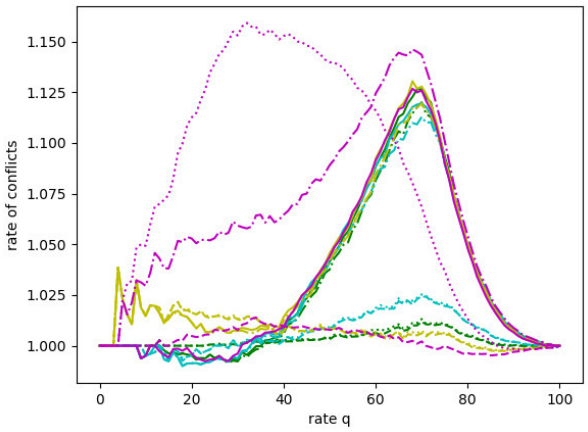

Fig. 5. Rate of number of conflicts and the benchmark value at different rates q of positive edges. Lower value is better.

benchmark, while Fig. 7 shows the cases when the benchmark is the winner. The greater the difference, the darker the colour. In order to make the differences more visible, in Fig. 7 we set the same black colour for the bigger differences.

It is interesting to compare the numbers and the coloured strips belonging to the methods; but for most of us these pictures and numbers contain too much information. So it follows that they should be compressed somehow, highlighting only the relevant information. For this reason, we have summed up the numbers (rates) grounding Fig. 5. We did not do a complete summing on interval $[0, 1]$—similarly to Fig.4—, but we constructed four subintervals and summed on them, as a

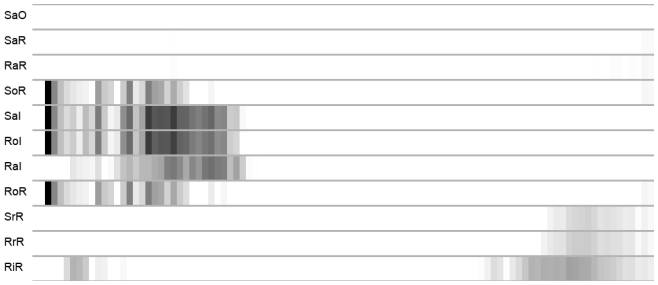

Fig. 6. Methods and rates q which give better result than the benchmark method SaO.

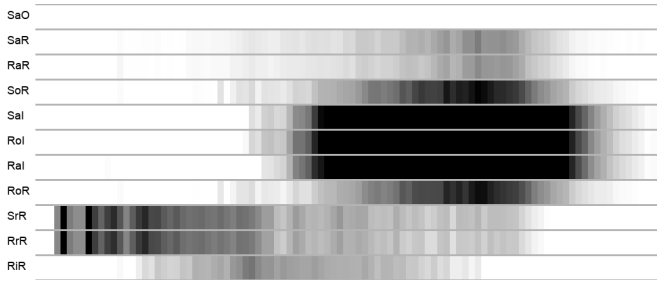

Fig. 7. Methods and rates q which give worse result than the benchmark method SaO.

TABLE I
SUBSUMS OF OUR OPTIMIZATION METHODS

Method	$\sum_{0}^{0.38}$	$\sum_{0.39}^{0.58}$	$\sum_{0.59}^{0.7}$	$\sum_{0.71}^{1}$
SaO	0.0	0.0	0.0	0.0
RaR	1.9	7.1	9.6	10.0
RiR	16.6	16.6	4.2	-7.2
SaR	2.3	8.1	11.3	11.1
RoR	-5.8	16.9	23.9	24.0
SoR	-5.2	18.2	25.0	24.8
RrR	54.8	14.7	6.1	1.4
SrR	55.7	15.7	7.3	2.6
SoI	-15.4	72.7	116.7	108.6
SaI	-10.9	68.3	120.3	111.0
RoI	-14.3	79.5	125.3	110.4
RaI	-9.5	75.3	129.2	112.2
RiI	-7.2	81.0	132.5	107.7
SrI	42.6	75.3	122.6	109.8
RrI	44.5	81.9	133.4	111.2
SiI	162.8	172.4	164.8	122.7
SiR	368.1	282.9	115.4	36.2

Table I shows. Here, the rows of the table are sorted by their sums, so the better methods are at the top, and the mostly weak methods are at the end in the table. Here the running time could not be taken into account because it was not recorded.

There are several negative numbers in the first column of Table I, i.e. several methods perform better for small q than the benchmark. As q is small, tolerance is achieved between a few pairs of objects, so many small clusters will be included in the resulting partition. Therefore the merging steps do not cause very significant changes, so we can iterate the merging of clusters inside one merging step. According to the information summarised in this table, the best results belong to the methods SoI and RoI. To verify our findings, we made some experiments similar to Fig. 4. However, whilst q was arbitrary in that random tolerance relations used, here we tighten it for the given intervals. Fig. 8 reinforces that these methods really are the best, where RoI is more than three times faster than SoI. Only method RaI precedes both of them in speed but not in precision, and these three methods are the Pareto set (that is, there does not exist a method which is faster and more accurate at the same time).

There are no negative numbers in the second and third columns of Table I, i.e. no method can overtake the slowly and

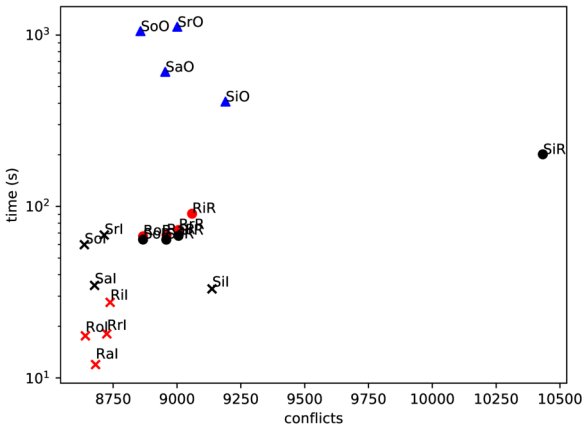

Fig. 8. Cumulative results on random relations where $q \in [0, 0.38]$: running time and efficiency of the variants of contraction

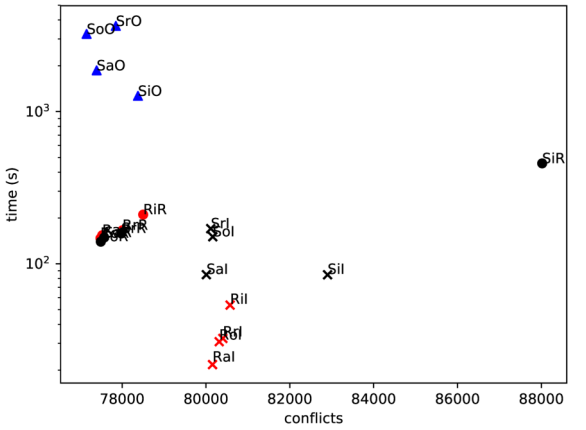

Fig. 9. Cumulative results on random relations where $q \in [0.38, 0.58]$: running time and efficiency of the variants of contraction

confidently evolving benchmark. Fig. 9 shows that the method SoO—not included in the table—is the absolute winner, even ahead of the benchmark. But it takes a long time to complete it. The method SoO is 22 times slower than the third ranked method RoR and 23 times slower than fourth ranked method SoR.

The methods RaR-SaR, which got small numbers in the second column of the table are the fifth and sixth most effective methods, and the methods SrR-RrR have scored the eighth and ninth place, but the methods RoR-SoR are better than all of them in the Pareto sense. The method SiO marked with the fourth triangle is the tenth, and there are some more efficient (faster and more precious) methods.

In Fig. 10—the figure representing the third column—triangles are located in the first five places, only the method RiR slipped in between them in the fourth place. That is, the principle *haste makes waste* really applies here. Looking

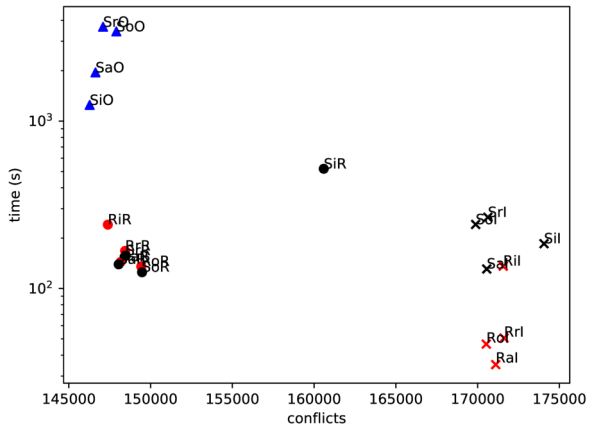

Fig. 10. Cumulative results on random relations where $q \in [0.59, 0.7]$: running time and efficiency of the variants of contraction

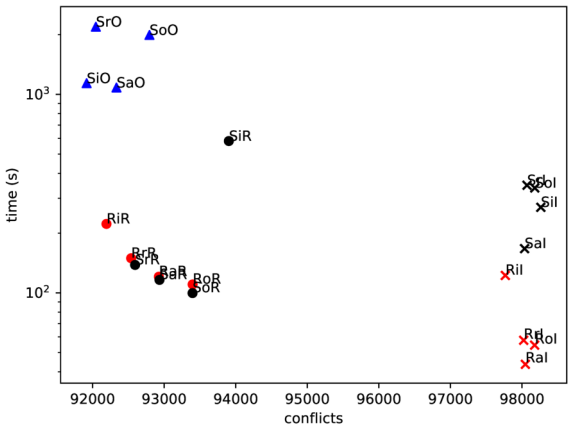

Fig. 11. Cumulative results on random relations where $q \in [0.71, 1]$: running time and efficiency of the variants of contraction

at the captions of the figure, we can see that most of the conflicts are here, so we have to handle the merging steps very carefully. The greedy independent merger performs poorly, all the crosses are on the right.

In the last column of Table I there is only one negative number and it belongs to the method RiR. The numbers belonging to methods RrR and SrR are small, but positive. If we narrow the last interval to $[-0.81.1]$, then all three numbers would turn negative (while the others would remain be positive), and their values would be -5.4, -2.8 and -2.2 respectively. These are the three methods in the last three rows of Fig. 6, and only these strips have significant non-white parts on the right. The method RiR is overcome by the methods SiO and SrO, but their runtime is nearly 5 and 10 times higher. On the other hand, RiR defeated our benchmark in the last interval, and the benchmark followed by the aforementioned methods RiR and SiR.

IV. Discussion

When wishing to use these optimisation methods in practice, it is possible to choose a precise or fast method depending on the desired outcome, along with the q value of the tolerance relation. The Pareto set is the set of methods that are optimal in that sense. That is, there is no such method which is better in both speed and precision than those in the Pareto set, so it is worth choosing a method contained in this set. Table II shows for each of the intervals which methods are optimal in this sense, and what are their ranks among the examined methods in accuracy and speed. For example, the method RaI in the first row of the table is the fastest of all, but if we disregard the first interval, then it performs with very poor accuracy.

Consider the building blocks of our methods and look at their roles. The *sequential* structure provides the most accurate solutions in each case, finding the closest equivalence relation for the tolerance relation of the problem. However, this often takes a lot of time. For a few tasks, a half-percent error can be sacrificed in accuracy for a fractional time calculation. That is why we may need a *repeated* structure.

TABLE II
Pareto Sets on different intervals

Method	$[0, 0.38]$	$[0.39, 0.58]$	$[0.59, 0.70]$	$[0.71, 1]$
RaI	4/1	14/1	18/1	16/1
RaR				8/7
RoI	2/2		14/2	
RoR		3/8	10/6	
RrI				14/3
RrR				5/10
RiI				
RiR			4/13	3/12
SaI		12/6		
SaR			6/8	9/6
SoI	1/7			
SoR		4/7	11/4	10/4
SrI				
SrR				6/9
SiI				
SiR				
SaO		2/18		
SoO		1/19		
SrO				
SiO			1/17	1/18

The implementation *recalculate* of the merging step can be interpreted as a compromise between the two structures, as this implementation is practically a cycle around the interpretation *one*. The interpretation *recalculate* is indicated by a circle in our figures. Except for the first interval, these circles are shown below the triangles (*one*) and to left on the crosses (*independent*), so they can also be considered as a compromise in this sense, they try to balance between the speed and the accuracy.

Apart from the first interval, the triangles (*one*) occupy the left upper part of the figures, resulting in precise solutions and

even the most accurate ones.

The crosses only perform well in the first interval, but then there are dominant.

When we turn to the implementation of motions, the implementation of *ordered* performed very nicely as a new player. The fact that we start by first processing the low-ranking objects—that cause the small changes—has clearly proven to be efficient in the first two intervals, reaching two-two podium positions. If we compare the efficiency in the lower left corner of the figure (ignoring the speed) we get the order of o-a-r-i, i.e. by fixing the structure and merging step we go from the most powerful method to the weakest.

In the third and fourth intervals, we cope with orders i-a-r-o and i-r-a-o, respectively. Partly this is the reason that in the third interval the method SaR is also a Pareto optimum beside the method RiR, and the method of RaR is just behind SaR. In the fourth interval, for similar reasons we can discover the Pareto optimum chain of methods RiR-RrR-SrR-RaR-SaR-SoR. Therefore, by prioritising the speed of the implementations *all* and *reversed* can play a role.

The motion step's *independent* implementation is very divisible. The SiI and SiR methods perform poorly for every interval. However, RiR is considered to be the Pareto optimum in the last two intervals, and it is very close to it in the second interval too. Additionally, the most accurate method of the last two intervals is SiO. The latter may be due to the fact that larger clusters are formed here, so moving an object from cluster g to cluster h in this implementation means that elements of g and h will be excluded from moving in that step. So from the four moving methods, it moves the least amount of items. Implementation O also means a one-time merging, so we change our partition the least as possible with this method. Many of these small changes—together—give the closest partition.

If we summarise the above statements, we see that each implementation and structure has its own place, and it is worth using different methods under different circumstances (for different q values).

V. Conclusion and further work

We've reviewed our previously described methods and implemented two new ways to move our objects in order of their importance and process them accordingly. We carried out a much larger and more thorough experiment than before, which included the new methods too. We elected four intervals of $[0, 1]$ in which the methods perform differently. By restricting our experiments to these intervals we determined which methods should be used for a given tolerance relation.

Python implementations were suitable for prototyping, testing new and newer ideas. Now we wish to develop a faster (probably C++) implementation that constantly updates the values of forces $a(x, g)$ when changing the partition, and stores whether the information on objects that need to be moved or joined is valid or outdated.

This can eliminate unnecessary calculations, which can speed up our methods. It should be taken into account how much energy such an administration needs, and whether it makes the software faster or slower.

References

[1] Agarwal, G., Kempe, D.: Modularity-maximizing graph communities via mathematical programming. The European Physical Journal B **66**(3), 409–418 (2008)

[2] Aszalós, L., Bakó, M.: Advanced search methods (in Hungarian). http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0103_13_fejlett_keresoalgoritmusok (2012)

[3] Aszalós, L., Bakó, M.: Contraction methods for correlation clustering: The order is important. In: Recent Advances in Computational Optimization, pp. 1–13. Springer (2019)

[4] Aszalós, L., Nagy, D.: Iterative set approximations based on tolerance relation. In: International Joint Conference on Rough Sets, pp. 76–88. Springer (2019)

[5] Bakó, M., Aszalós, L.: Combinatorial optimization methods for correlation clustering. In: D. Dumitrescu, R.I. Lung, L. Cremene (eds.) Coping with complexity, pp. 2–12. Casa Cartii de Stiinta, Cluj-Napoca (2011)

[6] Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Machine Learning **56**(1-3), 89–113 (2004). DOI 10.1023/B:MACH.0000033116.57574.95. URL http://dx.doi.org/10.1023/B:MACH.0000033116.57574.95

[7] Berend, D., Tassa, T.: Improved bounds on bell numbers and on moments of sums of random variables. Probability and Mathematical Statistics **30**(2), 185–205 (2010)

[8] Bonchi, F., Gionis, A., Gullo, F., Tsourakakis, C.E., Ukkonen, A.: Chromatic correlation clustering. ACM Transactions on Knowledge Discovery from Data (TKDD) **9**(4), 34 (2015)

[9] Demaine, E.D., Emanuel, D., Fiat, A., Immorlica, N.: Correlation clustering in general weighted graphs. Theoretical Computer Science **361**(2-3), 172–187 (2006)

[10] Emanuel, D., Fiat, A.: Correlation clustering–minimizing disagreements on arbitrary weighted graphs. In: European Symposium on Algorithms, pp. 208–220. Springer (2003)

[11] Giotis, I., Guruswami, V.: Correlation clustering with a fixed number of clusters. In: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 1167–1176. Society for Industrial and Applied Mathematics (2006)

[12] Néda, Z., Florian, R., Ravasz, M., Libál, A., Györgyi, G.: Phase transition in an optimal clusterization model. Physica A: Statistical Mechanics and its Applications **362**(2), 357–368 (2006). DOI 10.1016/j.physa.2005.08.008. URL http://dx.doi.org/10.1016/j.physa.2005.08.008

[13] Shi, J., Malik, J.: Normalized cuts and image segmentation. Departmental Papers (CIS) p. 107 (2000)

[14] Zahn Jr, C.: Approximating symmetric relations by equivalence relations. Journal of the Society for Industrial & Applied Mathematics **12**(4), 840–847 (1964). DOI 10.1137/0112071. URL http://dx.doi.org/10.1137/0112071