An algorithm for automatic assignment of reviewers to papers

Yordan Kalmukov

The latest version of the algorithm, better explained and accompanied with pseudo codes, examples and comprehensive experimental analyses proving its accuracy and complexity, is available here:
Kalmukov, Y. An algorithm for automatic assignment of reviewers to papers. Scientometrics 124, 1811-1850 (2020). https://doi.org/10.1007/s11192-020-03519-0

Full text of the author's accepted manuscript is available here:
https://www.researchgate.net/publication/342245629_An_algorithm_for_automatic_assignment_of_reviewers_to_papers

During the last years the conference management software has become widely popular. 5 years before it has been just an alternative of the traditional way of submitting papers and reviews by email, but now it tends to be the major way of organizing scientific conferences. The reasons for its popularity are quite obvious – reduced costs; convenience for all users; papers and reviews can be easily submitted and updated. But the biggest benefit is for the organizers of conferences as they have a powerful, centralized management tool that gives them an access to all papers and reviews from all over the world at any time [1]. Conference management software automatically calculates final marks, searches for conflict situations and prepares charts and reports. However as the number of papers and reviewers gets bigger, the process of manual assignment of reviewers to papers become harder and harder. That's why the functionality for automatic assignment is an important part of every conference management system. A correct and meaningful automatic assignment can be performed only if reviewers and authors provide detailed information about their interests, respectively papers. Keywords are commonly used for expressing areas of interest and describing papers in details. The easiest way of implementing this "keyword strategy" is giving users an ability to select keywords by using HTML checkboxes. In this case no string processing or semantic analysis is needed.

Good quality conference management systems, like The MyReview System [2], rely on well-known algorithms for automatic assignment of reviewers to papers that guarantee best possible accuracy. These algorithms, however, are very slow. The authors of The MyReview System propose to system administrators to run the C-based version of the algorithm if the number of submitted papers gets more that 200. Running the C-based version (as everything else is written in PHP) makes the system hard to install and maintain.

This paper suggests an alternative algorithm for automatic assignment of reviewers to papers that:

- Provides uniform distribution of papers to reviewers, i.e. all reviewers have an equal number of papers to review.
- Guarantees that if a paper has at least one keyword in common with a reviewer, then the paper will have a reviewer assigned to it (feasible if the number of papers that can be evaluated by that particular reviewer only, are not more than the maximum allowed number of papers per reviewer).
- Runs times faster comparing to similar algorithms.

An algorithm for automatic assignment of reviewers to papers

As mentioned the presented algorithm relies on keywords provided by authors and reviewers of papers. Using HTML checkboxes to select keywords suggests that there is a ***finite set*** of keywords defined by conference chairs. In most conferences 30 to 35 keywords are enough to describe different thematic fields within a larger category. For

example let's take conference on software technologies. Possible keywords are: databases, artificial intelligence, neural networks, image processing, data security & integrity and so on. Users are required to select at least *n* keywords. If the number of elements within the set is 30, then n = 2 or 3 keywords sounds reasonable. Authors usually select 4 to 5 keywords while reviewers select even more. Finding the most suitable reviewer for a paper is easy. Actually the automatic assignment should be very fast and easy if there were no restrictions in the number of papers per reviewer. As the number of papers per reviewer is automatically calculated by using (1) the algorithm has to provide a uniform distribution of papers to reviewers, i.e. all reviewers to have an equal number of papers to review. The ceil() function rounds the result up to the first integer.

number of papers per reviewer =
ceil((number of papers * number of reviewers per paper) / number of reviewers); (1)

A similarity factor is used to find the most suitable reviewers for each paper. The similarity factor is defined as:

$$SF_{PiRj} = \frac{count(KW_{Pi} \cap KW_{Rj})}{count(KW_{Pi} \cup KW_{Rj})} \quad (2)$$

where:

SF_{PiRj} - similarity factor between i-th paper and j-th reviewer

KW_{Pi} - set of keywords, describing the i-th paper

KW_{Rj} - set of keywords chosen by the j-th reviewer

count() - gives the number of elements within a set. When uniting the two sets the duplicates are ignored, i.e. the result set has unique values only.

$KW_{Pi} \subseteq KU$ and $KW_{Rj} \subseteq KU$, (3)

where KU is the set of all keywords defined by the conference chairs.

Obviously $SF_{PiRj} \in [0,1]$

SF_{PiRj} shows not only how suitable is the j-th reviewer to evaluate the i-th papers, but how suitable is the i-th paper to the j-th reviewer as well.

Here is an example:

Let's assume that conference chairs defined KU = {A, B, C, D, E, F, G, H, I}

KW_P = {A, C, F}; KW_{R1} = {B, C, F, G}; KW_{R2} = {A, F}

Reviewer 1 (R1) and reviewer 2 (R2) are ***equally capable*** of reviewing paper (P), as they cover 2 of the 3 keywords describing the paper. So, whom the paper should be assigned to? Let's turn to the similarity factors. SF_{PR1} = 2/5; SF_{PR2} = 2/3. Reviewer 2 (R2) is more suitable for paper (P) than reviewer 1 (R1). Why!? Because reviewer 1 has selected more keywords which means he is capable of reviewing more papers. In other words the probability of finding another paper that could be evaluated by reviewer 1 is bigger than the probability of finding a paper that could be evaluated by reviewer 2. That's why reviewer 2 is better choice for this paper. Assigning reviewer 1 to the paper is wasting of resources. Don't forget that there is a limit in the number of papers per reviewer! Think of what will happen if the next paper (P1) is described by {B, G} and reviewer 1 is busy evaluating paper P. Then there is nobody to review paper P1.

The algorithm relies on two data structures (PS and RS) loaded with the relevant data in advance. PS keeps information about all of the papers, while RS have data for all registered reviewers.

PS[index][PaperID]
[title]
[author]
[institution]
[country]
[reviewers][] // array of reviewers' usernames
[category]
[keywords][] // array of keyword identifiers

RS[username][name]
[institution]
[country]
[assignedPapers] // number of assigned papers
[categories][] // array of category identifiers
[keywords][] // array keyword identifiers

The suggested algorithm can be divided into four main steps. At the beginning it needs to know the similarity factor for every pair paper-reviewer. That's why it starts with building a matrix of similarity factors, called K matrix. This is the algorithm's first step. As shown on figure 1, the algorithm takes i-th paper and calculates all similarity factors between that paper and every reviewer, thus forming the K[i] column of the matrix. After finding all factors relative to i-th paper, the algorithm then sorts K[i] by similarity factor in descending order, so the most suitable reviewer for i-th paper is on top of the list. The algorithm then continues with the i+1 paper, forming K[i+1] and so on.

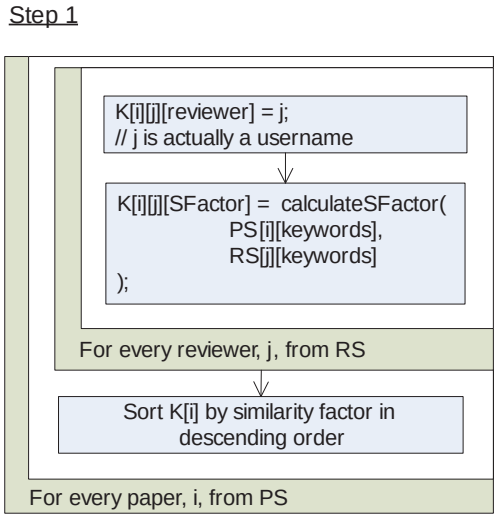


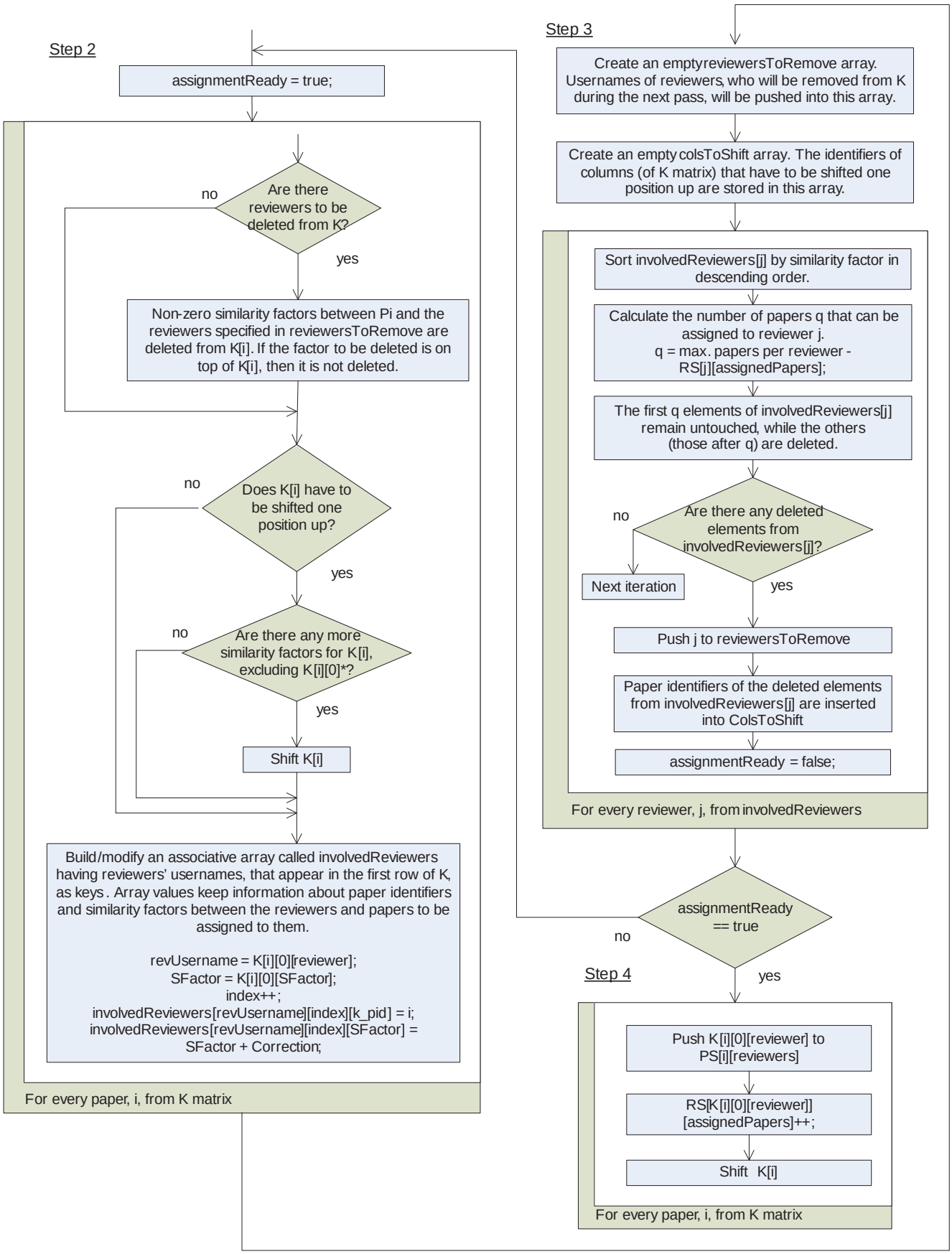
Figure 1. Step 1 – building a matrix of similarity factors

At the end of the first step the K matrix will look like this:

P1	P2	P3	P4	P5
R1 => 0.40;	R1 => 0.54;	R5 => 0.27;	R1 => 0.27;	R1 => 0.25
R5 => 0.38;	R5 => 0.42;	R4 => 0.25;	R5 => 0.22;	R4 => 0.14
R2 => 0.20;	R4 => 0.30;	R1 => 0.21;	R4 => 0.17;	R5 => 0.09
R3 => 0.18;	R3 => 0.20;	R2 => 0.14;	0	0
R4 => 0.17;	R2 => 0.10;	R3 => 0.13;	0	0

The column number points to the paper data stored within the PS structure. The real paper identifier can be obtained from PS[i]['PaperID'].

As a second step the algorithm processes the first row of the matrix (figure 2). The first row suggests the most suitable reviewer for each paper. In most cases it is impossible to assign the suggested reviewers directly to the papers, because there may be reviewers



* K[i][0] means the very first element of K[i]. The second index in fact is a string username, but by [0] we denote the very first element

Figure 2. Step 2 – building invlovedReviewers structure; Step 3 – Processing involvedReviewers and determining which papers to be assigned to reviewers who appear in the first row of K matrix; Step 4 – finalizing assignments.

who are suggested for too many papers, i.e. suggested for more papers than the maximal allowed number of papers per reviewer. To avoid this problem the algorithm builds a special data structure called ***involvedReviewers***. It is an associative array. The keys of this array are reviewers' usernames that appear in the first row of K. The array values keep paper identifiers and similarity factors between the reviewer and the papers suggested to be assigned to him. Here is an example:

```
involvedReviewers['revUsername'][index][k_pid]
                                        [SFactor]

involvedReviewers['R1'][1][k_pid] = 2;          // this is paper # 2 in K
                          [SFactor] = 0.54; // similarity factor between
                                            // reviewer R1 and paper # 2 (P2)
```

The similarity factors stored in *involvedReviewers* structure are modified with a correction C. The goal of this correction is to improve the assignment and mainly to guarantee that if a paper has at least one keyword in common with a reviewer, then the paper will have a reviewer assigned to it (that is possible only if the specified reviewer, or reviewers, has not exceeded the maximum allowed number of papers per reviewer). The correction can be divided into two fractions: C = C1 + C2, where C1 is calculated as:

If (number of non-zero SFs == 0) {

C1 = max; (4)

} elseif (number of non-zero SFs > 2*(revsPerPaper – numberOfRevs for P_i)) {

C1 = 0; (5)

} else {

$$C1 = \frac{\text{revsPerPaper - numberOfRevs for Pi}}{(\text{number of non - zero SFs})^3} \quad (6)$$

}

- number of non-zero SFs – number of non-zero similarity factors for P_i, excluding the first element of K[i].
- revsPerPaper – number of reviewers per paper, defined by conference chairs. Usually 2 or 3.
- numberOfRevs for P_i – current number of assigned reviewers to paper P_i.

The specified formulas are practically obtained. If the reviewer on top of K[i] is the only one reviewer for P_i, then he will be assigned to this paper regardless the similarity factors between him and the other papers. If there are plenty of reviewers suitable to review P_i, i.e. plenty of similarity factors in K[i] then the similarity factor stored in *involvedReviewers* is not modified with C1, thus C1 = 0.

The idea of C2 is to force the reviewer to be assigned to that paper whose second-suitable reviewer has much less similarity factor comparing to the first-suitable reviewer. So C2 is calculated as:

C2 = 2 * (SF of first-suitable reviewer for P_i – SF of second-suitable reviewer for P_i)

If the current pass through step 2 is not the first one there may be busy reviewers, i.e. reviewers who already have enough papers to review. These reviewers have to be removed from K as they will never be assigned to any more papers. If there are busy reviewers theirs usernames should be written in the ***reviewersToRemove*** array, formed during step 3 of the previous pass. So before forming the *involvedReviewers* structure, the algorithm first deletes all similarity factors between papers and the reviewers specified in *reviewersToRemove*. The reviewers are removed from the whole K matrix except from its first row. The columns corresponding to papers that will not be assigned to the busy

reviewers are shifted one position up. The identifiers of these columns should be inserted into *ColsToShift* array during the previous pass through step 3. If K[i] has to be shifted one position up, but the reviewer on top of K[i] is the only one suitable to review P_i, who left after deleting the busy reviewers from K, then K[i] is not shifted. Thus during the formation of *involvedReviewers* structure the similarity factor between that reviewer (the only one who left to review P_i) and P_i will be modified with the biggest possible correction, so at step 3 P_i will be assigned to him.

During the 3-rd step (fig. 2) the algorithm processes the *involvedReviewers* structure and determines which papers to be assigned to the reviewers who participate in the first row of K. Here is the explanation of this step by example. If restrictions are "papers per reviewer = 2" and "reviewers per paper = 2" and the K matrix has the values showed on the previous page, then after the second step the *involvedReviewers* will look like this (similarity factors are modified by C1 + C2):

```
involvedReviewers['R1'] =
                                {
                                  P1 => 0.40 + 0.04;  / /  0.44
                                  P2 => 0.54 +0.24;  // 0.78
                                  P4 => 0.27 + 0.25 + 0.10;  // 0.62
                                  P5 => 0.25 + 0.25 + 0.22; // 0.72
                                }
```

Then the algorithm sorts involvedReviewers['R1'] by similarity factor in descending order, i.e. involvedReviewers['R1'] = { P2, P5, P4, P1 }

As the allowed number of papers per reviewer is 2, only the first two papers (P2 and P5) are assigned to R1. The identifiers of P4 and P1 are inserted into *colsToShift* array, thus the corresponding columns in K (K[4] and K[1]) will be shifted one position up during the next pass through step 2. Analogically the username of reviewer R1 is pushed to *reviewersToRemove* that cause R1 to be removed from K (except from its first row) during the next pass through step 2.

The same actions are done for the other reviewers as well. If there is at least one column from K to be shifted up, then the first assignment of reviewers to papers is not ready and the ***assignmentReady*** flag is set to false. The execution then goes back to step 2 until *assignmentReady* gets true.

At the forth step (fig. 2) the first row of K contains all of the reviewers who will be assigned to papers 1 to n. The algorithm then passes through all papers, completes the 'reviewers' field of PS structure, increments the number of assigned papers for the specified reviewers within the RS structure and finally deletes the first row of K as reviewers have been already moved to 'reviewers' field of PS.

At the end of the fourth step all papers have 1 or 0 reviewers assigned to them. If papers have to be evaluated by more reviewers (as they usually have to) step 2 to 4 should be repeated as many times as needed.

Complexity

By analyzing the algorithm it can be estimated at a glance [3] that it runs in O(P.R.lg(R)), where P – the number of papers; R – the number of reviewers. However the algorithm's complexity will be a subject of further theoretical and experimental study.

Experimental results

The algorithm has been tested dozens of times by using randomly generated test data and several times by using real data fetched from CompSysTech' 2006 database. All 3 features, given on page 1, have been practically proven.

Unfortunately the suggested algorithm is not directly compared in terms of accuracy and running time to any other algorithm yet. However there is a table published on the

official web site of The MyReview System [2] that shows the execution time for assigning 50 reviewers to 100 papers; 100 reviewers to 150 papers and 150 reviewers to 200 papers. It is written that these results are obtained on a laptop computer. For a comparison (which of course is not claimed to be correct enough as the two algorithms are tested on different machines) the suggested algorithm is also run on a laptop computer [4] - 2 GHz Celeron, 512 MB of RAM, Windows XP, Apache 1.3x, PHP 4.3.4, MySQL 3.23.45.

The results of this comparison experiment are summarized in table 1. As expected the suggested algorithm runs times faster as it has better asymptotic efficiency than the optimal-weighted matching algorithm implemented in [2].

Table 1

	Optimal weighted matching algorithm (The MyReview System)	Algorithm suggested by Yordan Kalmukov
Number of papers and reviewers	Execution time	Execution time
100 papers; 50 reviewers	50 sec	2.3 sec
150 papers; 100 reviewers	225 sec	5.5 sec
200 papers; 150 reviewers	300 sec	11 sec

CONCLUSIONS AND FUTURE WORK

The algorithm for automatic assignment of reviewers to papers suggested in this paper provides an uniform distribution of papers to reviewers; guarantees that if a paper has at least one keyword in common with a reviewer, then the paper will have a reviewer assigned to it; runs times faster comparing to other algorithms. All of these are practically proven.

In the next several months the suggested algorithm will be compared to the optimal weighted matching algorithm implemented in The MyReview System [2] in terms of accuracy and running time. Thus both algorithms will be integrated in a single conference management system in order to share the same input data. Accuracy criteria have to be defined and analyzed. A possible criterion is the average similarity factor (ASF) for the assignments. The ASF can be calculated as a sum of similarity factors of all ***assigned*** pairs <paper-reviewer> divided by the number of assignments. The algorithm having the highest average similarity factor is the most accurate one. But this is a subject of further study.

REFERENCES

[1] Kalmukov, Y., B. Rachev, A. Smrikarov, About the necessity of building a web-based conference management system, Annual conference of the University of Ruse, Ruse 2005

[2] The MyReview System – a conference management system - http://myreview.lri.fr/

[3] Kalmukov, Y., Exploring the asymptotic bounds of the algorithm for automatic assignment of reviewers to papers suggested by Yordan Kalmukov, http://ecet.ecs.ru.acad.bg/etndec/JKalmukov/ADWMAasymptoticBounds.doc

[4] Kalmukov, Y., Experimental study of the algorithm for automatic assignment of reviewers to papers suggested by Yordan Kalmukov, http://ecet.ecs.ru.acad.bg/etndec/JKalmukov/ADWMAexperimentalStudy.doc

[5] Cormen, T., C. Leiserson, R. Rivest, C. Stein, Introduction to algorithms, Second edition, The MIT press, England 2001

[6] Sedgewick, R., Algorithms, Addison-Wesley Publishing, USA 1984

ABOUT THE AUTHOR

Yordan Kalmukov, MSc Student, Department of Computing, University of Ruse, Phone: +359 82 459038, e-mail: JKalmukov@gmail.com