Algorithm Of Dynamic Programming For Paper-reviewer Assignment problem

Nguyen Dinh Dung[1], Nguyen Huu Cong[2], Nguyen Tuan Anh[3]

[1,2,3] *Thai Nguyen University, Thai Nguyen, Viet Nam*

Abstract - *In this paper, we give paper-reviewer assignment problem based on paper keywords and database of reviewer keywords. We consider string matching algorithms as Brute Force Algorithm and KMP Algorithm to solve this problem, these algorithms seem to be effect less in case changing of phrase form. So, our algorithm was developed using dynamic programming to find suitable reviewers for papers. We give an expertise distance that is assignment criteria for paper-reviewer assignment and determined by edit distance.*

Key Words: Pattern matching, Dynamic programming, Edit distance, paper reviewer assignment.

1. INTRODUCTION

Peer review is a common task to people such as software developers, conference organizers, journal editors, grant administrators and educators. The paper-to-reviewer-assignment process aims to find the most expert reviewers for each submission. Obtaining high quality reviews is of great importance to the quality and reputation of an journal or a conference. The assignment of each paper to a set of suitable reviewers requires knowledge about both the topics studied in the paper and reviewers' expertise. This problem isn't a new problem, it has been faced and solved many times It has attracted considerable interest from different domains. Several works have been made for conference paper-reviewer assignment by using methods such as mining the web [2], latent semantic indexing [3], probabilistic topic modeling [4], integer linear programming [5], minimum cost flow [6] and hybrid approach of domain knowledge and matching model [7]. Recently, Tayal, Saxena, Sharma, Khanna, and Gupta put forward a new method for solving reviewer assignment problem in government funding agencies [8], Li and Watanabe proposed an automatic paper-to-reviewer assignment approach based on the matching degree of the reviewers [9], Long, Wong, Peng, and Ye solved a conference paper assignment problem by maximizing the topic coverage of the paper-reviewer assignment [10].

A correct and meaningful 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. In this paper, we study the problem of assigning a single journal paper *P* to a reviewer who is chosen from a set of reviewers. We propose the problem of expertise matching and presented algorithm relies on keywords provided by authors and reviewers of papers. We have applied the proposed algorithm to support the assignment of papers to reviewers for an journal.

The goal of our Reviewer Assignment Problem (RAP) is to find the best assignment such that the objective is minimized subject to the constraints. RAP is formally defined as follows:

Let $R_i = \{R_{i1}, R_{i2}, \ldots, R_{im_i}\}$, is a set of keywords chosen by the R_i – reviewer *(i = 1,2,..., M)*; $P = \{P_1, P_2, \ldots, R_n\}$ is a set of keywords, describing the *P* - paper; an expertise distance is used to find the most suitable reviewers for each paper. The expertise distance is defined as: if *P* - paper has at least one keyword in common with a R_i -reviewer, then the expertise distance *D(R$_i$,P)* is defined to be $d - \dfrac{c_i - 1}{Max}$.
Where, c_i is number of keywords that *P* - paper has in common with R_i - reviewer; *Max* is a positive arbitrary large constant, *Max* is choosen such that *Max >>* c_i; *d* is a positive constant *(0<d<1)*. If a paper has at least one keyword in common with a reviewer, then the paper will have a reviewer who can be assigned to it.

Definition 1:

Given $R = \{R_1, R_2, \ldots, R_M\}$, $P = \{P_1, P_2, \ldots, P_n\}$. *RAP finds* R_k *– reviewer such that:*

$$\begin{aligned} &D(R_k, P) = \min_{1 \le i \le M} D(R_i, P); \\ &D(R_k, P) \le d; R_k \in D_k \end{aligned} \quad (1)$$

Where, D_k is a domain that R_k satisfies the actual constraints (actual constraint may be busy reviewers, reviewers who already have enough papers to review i.e.).

There are some methods of finding paper keyword in common with a reviewer as exact string matching algorithms and approximate string matching algorithms (often colloquially referred to as fuzzy string searching). We will consider algorithms based on different approaches, including Brute Force [11], Boyer-Moore approach [15, 16], Knuth-Morris-Prat string matching [12] and dynamic programming [11,13,14].

2. EXACT STRING MATCHING ALGORITHMS

In computer science, string searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.

Given a pattern string $S=S_0S_1...S_m$ and a text string $T=T_0T_1...T_n$ ($m \le n$), both the pattern and searched text are arrays of elements that are on an alphabet Σ (e.g. the set of ASCII characters, the set of bytes [0..255], etc.).

When we come to string matching the most basic approach is what is known as brute force [11], which means just to check every single character from the text to match against the pattern. The principles of brute force string matching are quite simple. We must check for a match between the first characters of the pattern with the first character of the text. If they don't match we move forward the second character of the text. Now we compare the first character of the pattern with the second character of the text. If they don't match again we move forward until we get a match or until we reach the end of the text. In case they match we move forward the second character of the pattern comparing it with the "next" character of the text, If case a character from the text match against the first character of the pattern we move forward to the second character of the pattern and the next character of the text. This algorithm is slow that its complexity is *O(nm)*. In brute force matching we checked each character of the text with the first character of the pattern. In case of a match we shifted the comparison between the second character of the pattern and the next character of the text. The problem is that in case of a mismatch we must go several positions back in the text. Well in fact this technique can't be optimized. In 1977 James H. Morris and Vaughan Pratt described their algorithm [12], which by skipping lots of useless comparisons is more effective than brute force string matching. The only thing is to use the information gathered during the comparisons of the pattern and a possible match. Implementing Morris-Pratt, first we have to preprocess the pattern and then perform the search. The following algorithms show how to do that.

Algorithm 1: Processing the pattern

```
        Input: S
        Output: Table KMP
1.1     void Creat_Table_KMP(){
1.2     KMP[0]=-1;
1.3     KMP[1]= 0;
1.4     i=2;
1.5     j=0;
1.6     while (i <m)
1.7           if(S[i-1]==S[j]){
1.8                KMP[i]= j + 1;
1.9                i = i + 1;
1.10               j = j + 1;
1.11          }
1.12          else  if( j > 0) j= KMP[j];
1.13          else {
1.14               KMP[i]= 0;
1.15               i = i + 1;
1.16          }
1.17    }
```

Algorithm 2: Performing the search

```
        Input: S, T
        Output: pos, matched
2.1     void search(){
2.2     i=0; l=0;
2.3     pos=l;//Position found
2.4     matched=0 ;
2.5     while (l + i <= n){
2.6       if (S[i] == T[l + i]){
2.7           i:= i + 1;
2.8           if (i ==m) {
2.9              matched=1;return {pos, matched};
2.10          }
2.11      }
2.12      else if (KMP [i] > -1){
2.13          i= KMP[i]; l= l + i - KMP[i];pos=l;
2.14      }
2.15      Else{
2.16        i= 0; l= l + 1; pos=l;
2.17      }
2.18    }
```

Thus the preprocess of the pattern can be done in *O(m)*, while the search itself needs *O(m+n)*.

Morris-Pratt algorithm is a very good improvement of the brute force string searching. However, if we have to find whether a single character is contained into a text we need at least *n+1* steps. Once we have to find whether a pattern with the length of *m+1* is contained into a text with length of *n+1* the case is getting a little more complex. In order to reduce the time consumption in searching there is such algorithm that is faster and more suitable than Morris-Pratt, that is the Boyer-Moore string searching [15, 16]. The Boyer-More algorithm successively aligns patter *S* with Text *T* and checks if *S* matche with corresponding tokens in *T* as in the case of the naive algorithm. Further after the check is complete *S* is shifted right relative to *T* just as in the naive algorithm. Further more, apples some intuite tricks to avoid unnecessary shifts and comparisons. The worst case running time of the algorithm is *O(n)*, The best case running time is *O(n/m)*.

3. EDIT DISTANCE AND APPROXIMATE STRING MATCHING FOR PAPER-REVIEWER ASSIGNMENT

In above section, we consider exact string matching algorithms to find paper keyword in common with a reviewer. These algorithms seem to be effectless in case changing of phrase form or appearance of errors in keywords but the meaning of these keywords remain the same. Example: paper keyword is "paper keyword" and reviewer keyword is "paper's keyword", they different at "'s", paper keyword does not appear in reviewer keyword but their meaning remain the same. Overcome this drawback, in this section, we propose a approximate searching algorithm base on computing the lengths of the longest common sub words of two strings [11,13,14]. The closeness of a match is measured in terms of the number of primitive operations necessary to convert the string into an exact match. This number is called the edit distance between the string and the pattern. The usual primitive operations are the removal, insertion, or substitution of a character in the string.

3.1. Edit distance

One possible definition of the approximate string matching problem is the following: Given a pattern string *S* and a text string *T*, find a substring in *T* which of all substrings of *T*, has the smallest edit distance to the pattern *P*.

A Brute-Force approach would be to compute the edit distance to *P* for all substrings of *T*, and then choose the substring with the minimum distance. However, this algorithm would have the running time $O(n^3m)$. A better solution that have the running time $O(nm)$, which was proposed by Levenshtein [11], relies on dynamic programming. A matrix is initialized measuring, the edit distance between the prefix of P with the prefix of *T*. The matrix can be filled from the upper left to the lower right corner. Each jump horizontally or vertically corresponds to an insert or a delete, respectively. The cost is normally set to 1 for each of the operations. The diagonal jump can cost either one, if the two characters in the row and column do not match or 0, if they do. Each cell always minimizes the cost locally. This way the number in the lower right corner is the edit distance between both words. The following algorithm shows how to do that

Algorithm 3: Edit distance

```
         Input: S, T
         Output: Edit distance
3.1      int Edit_distance(S, T){
3.2      for (int i = 0; i <= m; i++)
3.3              d[i, 0] = i;
3.4      for (int j = 0; j <= n; j++)
3.5              d[0, j] = j;
3.6      for (int i = 0; i <= m; i++)
3.7          for (int j = 0; j <= n; j++)
3.8          {
3.9           if (S[i] == T[j])
3.10              cost = 0;
3.11          else
3.12              cost = 1;
3.13            d[i + 1, j + 1] = Min(d[i, j + 1] + 1, d[i + 1, j] + 1,
                                       d[i, j] + cost);
3.14         }
3.15       return d[i,j];
3.16     }
```

3.2. Approximate string matching for paper – reviewer assignment

In order to calculate the expertise distance between reviewer and paper, we first calculate the edit distance between reviewer keywords and paper keywords. A paper keyword P_k in common with a reviewer keyword R_{ij} if edit distance between P_k and R_{ij} is not larger than *e* (*e* is a positive integer constant), the task is to find all approximate occurrences of the pattern in the text with at most *e* differences. The following algorithm calculates expertise distance and find the most suitable reviewers for a paper. First, we create a table that is named *Reviewer Keyword* with two columns of data named *Reviewer ID* and *Keyword*, then, we create a table that is named *Expertise Distance* with two columns of data named *ReviewerID* and *Distance*. The *Reviewer Keyword* stores the reviewer ID and Keyword, *Expertise Distance* stores the reviewer ID and expertise distance between paper and reviewer. An example: Paper keyword is *Matching*. Let *d* is 0.5.

Table -1: Table *Reviewer Keyword*

ReviewerID	Keyword
1	Matching
2	Random sequence
1	Paper reviewer assignment
3	Machanics
2	Machining
1	Dynamic programming

Table -2: Table *Expertise Distance*

ReviewerID	Expertise distance
1	0.5
2	x
3	x

The operational process of algorithm includes two phases. In first phase which is called expertise distance computing phase. *Expertise Distance* stores the values of expertise distance that determined during the keyword matching. The second part of the algorithm is the searching phase. During this phase, the algorithm is searching reviewer whose expertise distance value is minimum and satisfying the constraints in (1). We summarize our searching process in Algorithm 4.

Algorithm 4: Approximate searching algorithm

```
         Input: P, R,e,d, Max, $Sum=\sum_{i=1}^{M} m_i$
         Output: Return reviewer $R_k$ that satifies (1)
4.1      struct ReviewerKeyword {
4.2                  int ReviewerID;
4.3                  char Keyword [Length];
4.4           };
4.5     struct ExpertiseDistance {
4.6              int ReviewerID;
4.7              double Distance;
4.8           };
4.9     struct ReviewerKeyword RK[Sum];
4.10    struct ExpertiseDistance ED[M];
4.11    int ApproximateSearching(P, R,e,d, Max){
4.12     for (int i = 1; i <= M; i++)
4.13       ED(i). Distance=Max;
4.14     for (int i = 1; i <= Sum; i++)
4.15       for (int j = 1; j <= n; j++){
4.16         $\delta$ = Edit_distance($P_j$, RK(i). Keyword);
4.17         if ( $\delta$ <=e){
4.18           Finding k such that ED(k). ReviewerID is equal
               RK(i). ReviewerID;
4.19           if (ED(k). Distance>d)
4.20               ED(k). Distance=d;
4.21           else
4.22               ED(k). Distance= ED(k). Distance-1/Max
4.23           Break;
4.24         }
4.25       }
4.26    Finding k such that ED(k).Distance
        =$\min_{1\le i\le M}${ED(i).Distance} and satisfying the
        constraints in (1).
4.27     return ED(k).ReviewerID;
4.28    }
```

In case the keyword morphology changes but still has the same meaning or has the same topic, we set up the procedure for separating paper keywords into single words, then applying algorithm 5 for the original keyword set and single words. We conduct an example to demonstrate the effectiveness of algorithm 4: Let a paper keyword is "string matching" and a reviewer keyword is "pattern matching". They have he same content that is matching. However, if we calculate the expertise distace of reviewer and paper by using edit distace in algorithm 4 then edit distace is 5. If e <5 then the results of algorithm 4 show that there is not relevance between reviewer and paper. To avoid the above drawback of the retrieval based methods, we propose a modified version of algorithm 4.

Algorithm 5: Modified version of algorithm 4 for Approximate searching

```
5.1     int ApproximateSearching(P, R,e,d, $\mu$, Max){
5.2      for (int i = 1; i <= M; i++)
5.3        ED(i). Distance=Max;
5.4      for (int i = 1; i <= Sum; i++)
5.5        for (int j = 1; j <= n; j++){
5.6          $\delta$ = Edit_distance($P_j$, RK(i). Keyword);
5.7          if ( $\delta$ <=e){
5.8            Finding  k such that ED(k). ReviewerID is equal
               RK(i). ReviewerID;

5.9            if (ED(k). Distance>d)
5.10               ED(k). Distance=d;
5.11           else
5.12               ED(k). Distance= ED(k). Distance-1/Max;
5.13           Break;
5.14         }
5.15         else{
5.16           If($P_j$ is substring of RK(i). Keyword
                 or RK(i). Keyword is substring of $P_j$){
5.17             if (ED(k). Distance> $\mu$)
5.18                 ED(k). Distance= $\mu$;
5.29             Else
5.20                 ED(k). Distance= ED(k). Distance-1/Max
5.21             Break;
5.22           }
5.23       }
5.24    Finding k such that ED(k).Distance
        =$\min_{1\le i\le M}${ED(i).Distance} and satisfying constraints
        $D(R_k,P)\le\mu$ and $R_k\in D_k$;
5.25     return ED(k).ReviewerID;
5.26    }
```

In this algorithm 5, we consider parameter μ that can replace d in case the keyword morphology changes but still has the same meaning and $0<d<\mu<1$. When finding occurrence of P_j in *RK(i). Keyword* or occurrence of *RK(i). Keyword* in P_j, we use exact string matching algorithms that are presented at section 2.

4. EXPERIMENTS

In this section, We give an example to illustrate the process of our proposed algorithm 5. we first set e=2, μ = 0.5, d=0.1 and Max=10000. Let's paper keywords are {Pattern matching; Dynamic programming; Edit distance; Reviewer keyword; paper reviewer assignment}. Table RK is defined as follows:

Table -3: Example to illustrate algorithm

Reviewer	ReviewerID	Keyword
R1	1	Machine learning
	1	Dynamic programming
	1	Random sequence
R2	2	Clinical Document Architecture
	2	Drug bills
	2	String matching
R3	3	Soft computing
	3	Artificial neural network
R4	4	String matching
	4	Paper reviewer assignment
	4	Dynamic programming
	4	Neural network
	4	Reviewer's keyword

The results of searching are summarized in table 4.

Table -4: Results of searching

ReviewerID	Expertise distance	Relevance keywords
4	0.0998	Paper reviewer assignment; Dynamic programming, Reviewer's keyword
1	0.1000	Dynamic programming
2	0.5000	String matching

After calculating, we find three candidate reviewers: R_4, R_1, R_2. R_4 covers 3 of the 5 keywords describing the paper, expertise distance is 0.0998 that is smallest expertise distance in set of expertise distance. So, paper should be assigned to R_4. If R_4 is busy or R_4 already have enough papers to review then algorithm considers R_1 ($R_4 \notin D_4$). If $R_1 \notin D_1$ then algorithm considers R_2, If $R_2 \notin D_2$ then there is nobody to review paper. In this case, the action editor can select from this database using keywords, search the bibliography of the manuscript for appropriate reviewers, or use a feature in system to search for similar paper to ensure that the reviewers have the appropriate expertise. One issue is that in case of two or more reviewers that have expertise distance are equal, paper should be assigned reviewer that has selected less keywords which means he is capable of reviewing less papers, probability of finding another paper that could be evaluated by this reviewer is smaller than the probability of finding a paper that could be evaluated by other.

5. CONCLUSIONS

In this paper, we studied the case of journal Reviewer Assignment and proposed an effecient algorithm that finds the most suitable reviewers for each paper. Experimental results demonstrate that the proposed approach can effectively and efficiently match experts with the queries. More importantly, we showed an assignment algorithm that could lead to efficiency improvements in approximate searching. Based on the proposed method, we have built a system to suggest reviewer assignments for Science and Technology Journal of Thai Nguyen University. In the future we are also going to apply the proposed method to several real-world applications.

6. ACKNOWLEDGEMENT

This work was partially done in the scope of project "Development of software for management and support of selecting reviewers for Science and Technology Journal of Thai Nguyen University, Viet Nam", financially supported by This work was supported by the Thai Nguyen University Foundation of Science and Technology Development.

REFERENCES

[1] H. K. Biswas and M. Hasan (2007), "Using publications and domain knowledge to build research profiles: An application in automatic reviewer assignment", in ICICT.

[2] C. B. Haym, H. Hirsh, W. W. Cohen, and C. Nevill-manning (1999), Recommending papers by mining the web, In IJCAI'99, pp. 1–11.

[3] S. T. Dumais and J. Nielsen (1992), "Automating the assignment of submitted manuscripts to reviewers", In SIGIR'92, pp. 233–244.

[4] M. Karimzadehgan, C. Zhai, and G. Belford (2008), "Multi-aspect expertise matching for review assignment", In CIKM'08, pp. 1113–1122.

[5] M. Karimzadehgan and C. Zhai (2009), "Constrained multi-aspect expertise matching for committee review assignment". In CIKM'09, pp. 1697–1700.

[6] D. Hartvigsen, J. C. Wei, and R. Czuchlewski (1999), "The conference paper-reviewer assignment problem", Decision Sciences, 30(3): pp. 865–876.

[7] Y.H. Sun, J. Ma, Z.P. Fan, and J. Wang (2007), "A hybrid knowledge and model approach for reviewer assignment", In HICSS'07, pp. 47–47.

[8] D.K. Tayal, P.C. Saxena, A.Sharma, G. Khanna, S. Gupta (2014), New method for solving reviewer assignment problem using type-2 fuzzy sets and fuzzy functions, Applied intelligence, 40, 1, pp. 54-73.

[9] X. Li, T. Watanabe (2013), "Automatic paper-to-reviewer assignment based on the matching degree of the reviewers", Procedia Computer Science, 22, pp.633-642.

[10] C. Long, R. C. W.Wong, Y.Peng, L.Ye (2013), "On good and fair paper-reviewer assignment", In IEEE 13th International Conference on Data Mining (ICDM'2013), pp.1145-1150.

[11] Navarro, Gonzalo (2001), "A guided tour to approximate string matching", ACM Computing Surveys, 33 (1), pp. 31-88.

[12] Knuth, E. Donald, H. Morris, Jr,James and R. Pratt Vaughan (1977), "Fast pattern matching in strings", SIAM Journal on Computing 6.2, pp. 323-350.

[13] P. Sellers (1980), "The theory and computation of evolutionary distances: Pattern recognition", Journal of Algorithms, 1, pp.359–372.

[14] E. Ukkonen (1985), "Finding approximate patterns in strings", Journal of Algorithms, 6, pp.132–137.

[15] R. Boyer and S. Moore (1977), "A fast string searching algorithm", Communcations of the ACM, 20, pp.762–772.

[16] Tarhio and E. Ukkonen (1993), "Approximate Boyer-Moore string matching", SIAM Journal on Computing, 22, pp.243–260.