Maximum Common Subgraph and Median Graph Computation from Graph Representations of Web Documents Using Backtracking Search

Kaushik K. Phukon

Department of Computer Science,
Gauhati University, Guwahati-14, Assam, India
kaushikphukon@gmail.com

Abstract

After constructing graph representations for a set of web documents, there are several techniques to determine the similarity between same-type objects. This is achieved by graph matching. The measure of similarity may be based on the size of the maximum common subgraph. In this paper, we are interested in the problem of maximum common subgraph(MCS) and median graph computation for the purpose of graph clustering using backtracking search. Median of a graph helps in the extension of prevalent term frequency based clustering algorithms to graph based clustering.

Keywords: *Graph, Subgraph, Median, Algorithm, Graph Distance*

1. Introduction

When representing text and web document content for the purpose of clustering and classification, a vector-space model is generally used. In this model, each possible term that can appear in a document becomes a feature dimension. The value assigned to each dimension of a document may indicate the number of times the corresponding term appears on it or it may be a weight that takes into account other frequency information, such as the number of documents upon which the terms appear. This model is simple and allows the use of traditional machine learning methods that deal with numerical feature vectors in a Euclidean feature space. However, it discards information such as the order in which the terms appear, where in the document the terms appear, how close the terms are to each other, and so forth. By keeping this kind of structural information we could possibly improve the performance of various machine learning algorithms. The problem is that traditional data mining methods are often restricted to working on purely numeric feature vectors due to the need to compute distances between data items or to calculate some representative of a cluster of items *(i.e.*, a centroid or center of a cluster), both of which are easily accomplished in a Euclidean space [8]. Thus either the original data needs to be converted to a vector of numeric values by discarding possibly useful structural information or we need to develop new, customized methodologies for the specific representation.

The composite model [1, 2] is an efficient and well organized method of web document representation which takes into account additional web-related content information which is not done in traditional information retrieval models. It can hold almost all the necessary information such as the order, proximity of word occurrence, markup information and location of a word within a document. The composite model is a combination of TSGM(Tag Sensitive Graph Model) and CSGM(Context Sensitive Graph Model).We will show that This model along with the enhanced distance measure (Eq.1) is giving increased effectiveness in the graph distance measure*.

$$dist_{MCS}(G_1, G_2) = 1 - \left(\frac{\sum_{SOM} d^{\pm}(MCS(G_1, G_2))}{\max(\sum d^{\pm}(G_1), (\sum d^{\pm}(G_2)))} \right) \quad \ldots(1)$$

where $\sum d^{\pm}$ represents the sum of in-degree and out-degree of the directed graph and $\sum_{SOM} d^{\pm}$ represents the sum of the minimum of the degrees generated by each common node of the graphs which are included in the MCS. In case when there are two or more MCS then we should consider max($\sum_{SOM} d^{\pm}$). max(x, *y)* is the usual maximum of two numbers *x* and *y*.

2. Graph Distance

The above distance measure express that, as the size of the maximum common subgraph of a pair of graphs becomes larger, the similarity between the two graphs will also increase. The larger the maximum common subgraph, the smaller $dist_{MCS}(G_1,G_2)$ becomes, indicating more similarity and less distance. If the two graphs are in fact identical, their maximum common subgraph is the same as the graphs themselves and thus the size of all three graphs is equal: $|G_1| = |G_2| = |MCS(G_1,G_2)|$. This leads to the distance, $dist_{MCS}(G_1,G_2)$, becoming 0. Conversely, if no maximum common subgraph exists, then $|MCS(G_1,G_2)| = 0$ and $dist_{MCS}(G_1, G_2) = 1$. This distance measure has been shown to be a metric, and produces a value in [0,1]. This distance measure has four important properties [8]. First, it is restricted to producing a number in the interval [0,1]. Second, the distance is 0 only when the two graphs are identical. Third, the distance between two graphs is symmetric. Fourth, it obeys the triangle inequality, which ensures the distance measure behaves in an intuitive way. For example, if we have two dissimilar objects *(i.e.*, there is a large distance between them) the triangle inequality implies that a third object which is similar *(i.e.*, has a small distance) to one of those objects must be dissimilar to the other. The advantage of this approach over the graph edit distance method is that it does not require the determination of any cost coefficients

*In [1] the distance measure was mistakenly printed as-

$$dist_{MCS}(G_1, G_2) = \left(\frac{\sum_{SOM} d^{\pm}(MCS(G_1, G_2))}{\max(\sum d^{\pm}(G_1), (\sum d^{\pm}(G_2)))} \right)$$

or other parameters. However, the metric as it is defined in Eq. 1 may not be appropriate for all applications.

Graph matching is known to be a computationally expensive procedure, which limits most of its applications [3]. A number of graph-matching algorithms, both optimal and approximate, have been proposed over the last three decades. Optimal algorithms are those that guarantee the best solution(s) to be found, while approximate algorithms offer nearly-best solutions usually at considerably lower computation cost. Most optimal algorithms for common subgraph isomorphism (CSI), used in various applications, are based on maximal clique detection in the association graph, as was first proposed in [4, 5]. In contrast, backtracking CSI algorithms [6] are rarely used, presumably because of the extra

computational cost involved. It is widely accepted, however, that the problem of exact subgraph isomorphism (ESI, a special case of CSI when the maximal common subgraph coincides with one of the input graphs) is much more effectively solved by the backtracking algorithm due to Ullman (UA) [7]. This fact gives an implication that the backtracking approach may be efficient for CSI at least in cases that are close to ESI.

In the next sections of the paper, we are going to discuss an algorithm for determining the MCS based on CSIA. The CSIA algorithm was initially derived from UA by modification of parts related to the expansion of partial solutions and rejection of unsuitable branches of the search tree [3]. The algorithm was not designed to calculate the MCS of graphs representing web documents. It was a general CSI algorithm capable of handling much more complex situations of graph isomorphism than detection of MCS in case of well organized graphs created for the purpose of matching. We did the modification needed considering the fact that we only need the MCS and some other data from a well organized set of graphs.

3. Steps for Extracting the MCS

Calculating the distance between two graphs (Eq 1) requires the computation of the maximum common subgraph of the pair of graphs. The determination of the maximum common subgraph in the general case is known to be an NP-complete problem [8]. However, with our graph representation for documents [1] each node in a graph has a unique label (representing a unique term) that no other node in the graph has. Thus the maximum common subgraph, G_{MCS}, of a pair of graphs, G_1 and G_2, can be created by the following procedure:

(1) Find the nodes V_{MCS} by determining the subset of node labels that the original graphs have in common with each other and create a node for each common label.

(2) Find the edges E_{MCS} by examining all pairs of nodes from step (1) and introduce edges that connect pairs of nodes in both of the original graphs.

(3) Select the MCS which have the highest no of nodes and
$\sum_{SOM} d^{\pm} = \max(\sum_{SOM} d^{\pm} MCS(G_1,G_2))$.

We observe that the complexity of this method is $O(|V_1| \bullet |V_2|)$ for step (1) , since
we need only to compare each node label from one graph to each node label of the other and to determine whether there is a match or not. Thus the maximum number of comparisons is $|V_1| \bullet |V_2|$, and since each node has a unique label we only need to consider each combination once. The complexity is $O(|V_{MCS}|^2)$ for step (2), since we have $|V_{MCS}|$ nodes and we look at all combinations of pairs of nodes to determine if an edge should be added between them or not:

$$\left(|V_{MCS}|C_2\right) = \frac{|V_{MCS}|!}{2!(|V_{MCS}|-2)!} = \frac{|V_{MCS}|.(|V_{MCS}|-1)}{2} < |V_{MCS}|^2$$

Thus overall complexity is $O(|V_1|.|V_2|+|V_{MCS}|^2) \leq O(|V|^2 + |V_{MCS}|^2) = O(|V|^2)$, if we substitute $V=\max(|V_1|, (|V_2|)$.

In this paper we are presenting an algorithm for MCS detection, which will be referred to as AMCSI (Algorithm for Maximum Common Subgraph Isomorphism). The AMCSI algorithm is based on backtracking CSI algorithm (CSIA) due to E. B. Krissinel and K.

Henrick. The AMCSI algorithm has been proposed in view of determining the MCS of the graphs created according to the composite graph model for web page representation [1]. The computational complexity of the algorithm (AMCSI) is considerably lower than its parent algorithm (CSIA) designed for CSI in general [3]; the complexity of which (*i.e.*, CSIA) is also considerably lower than the complexity range shown by algorithms based on the maximal clique detection.

4. Backtracking Scheme of AMCSI based on CSIA

We shall represent graphs as 3-tuples $G = [V,E,n]$, where $V = \{v_i\}$ and $E = \{e_{ij}\}$ are sets of vertices and edges, respectively, and $n = |V|$ is the number of vertices. Each element v_i and e_{ij} has a set of properties, which may be called labels, assigned to vertices and edges.

For the identification of matching vertices and edges, there are two separate vertex and edge comparison functions $\mu(v_i, v_j)$ and $\nu(e_{ij}, e_{kl})$, respectively. These functions return true if vertices or edges compare, and false otherwise. A subgraph of graph $G = [V,E,n]$ is graph $H = [W,F,k]$ where $k \leq n$, $W \subseteq V$ and $F = E \cap (W \times W)$, or, in other words, F is a subset W of k vertices from V , connected in graph H by *all and only* edges that connect these vertices in graph G. Without loss of generality, we can consider that all unconnected vertices are connected by virtual null edges under the condition that null edges compare only with themselves.

Common subgraphs $H_1 = [V,E,k]$ and $H_2 = [W,F,k]$ of two given graphs G_1 and G_2 are those, of equal size k, that are isomorphic to each other. This means that there should be such numeration of subgraphs' vertices $x(i)$ and $y(i)$, that

$$\mu(v_{x(i)}, w_{y(i)}) = \text{true} \;\&\; \nu(e_{x(i),x(j)}, f_{y(i),y(j)}) = \text{true} \;\forall i, j \in \{1 \ldots k\} \qquad \ldots\textbf{(2)}$$

or, in other words, all pairs of matching vertices in the subgraphs are connected by matching edges (including the virtual null edges). The problem of finding the common subgraphs can now be rephrased as looking for numbered sets $X = \{x(i)\}(i=1\ldots k)$ and $Y = \{y(i)\}(i=1\ldots k)$, satisfying conditions (2). Figure 1 presents a recursive version of the backtracking AMCSI algorithm (in the first consideration, ignore the undefined 'VMM *D*, *D*1'). The algorithm essentially enumerates all possible mappings of vertices which satisfy the conditions of subgraph isomorphism (2) in sets X and Y . Initially X and Y are set empty (cf. step 2 of the main program). On each entry to *Backtrack*, X and Y index vertices of the partial common subgraphs found. The subgraphs may or may not be extended, which is checked by function *Extendable*. If the solution is not extendable, it is output in step 20, and the algorithm retreats. Otherwise, *Backtrack* picks a yet unmapped vertex v_i from graph G_1 (step 2) and identifies set Z of all vertices from graph G_2 (step 3) that may be mapped onto v_i without violation of subgraph isomorphism conditions (2) (candidate matchings). For each vertex w_j from set Z, the algorithm extends the solution by appending X and Y with indices of v_i and w_j , respectively (cf. steps 5 and 6 in Figure 1). Then *Backtrack* is called recursively (step 8) in order to make further extensions. After the recursive call, X and Y are restored (steps 9 and 10), which allows the algorithm to perform different mappings on each loopover. By going up and down the recursion, all possible extensions of common subgraphs, originating from the extension of subgraph G_1 by vertex v_i , are thus explored. It is clear, however, that common subgraphs do not have to contain any particular vertex. Therefore, the search must be complemented by exploring all extensions of X that do not index the chosen vertex (v_i). For that purpose, in steps 12–14 v_i is temporarily removed from graph G_1, *Backtrack* is called recursively again, and after it returns, v_i is put back into G_1.

Global graph $G_1 = [V,E,n]$, $G_2 = [W,F,m]$; set X, Y ; integer n_{max},d_{som}

1. **call** Initialize(D)
2. $X := \phi$, $Y := \phi$
3. $n_{max} := 0$, $d_{som}:=0$
4. **call** Backtrack(D)
5. stop

procedure *Backtrack* (VMM *D*)

1. **if** Extendable(D) **then**
2. v_i := PickVertex(D)
3. Z := GetMappableVertices(v_i ,D)
4. **for** all $w_j \in Z$ **do**
5. $X := X + \{i\}$
6. $Y := Y + \{j\}$
7. D_1 := Refine(D)
8. **call** Backtrack(D_1)
9. $X := X - \{i\}$
10. $Y := Y - \{j\}$
11. done
12. $V := V - \{v_i\}$
13. **call** Backtrack(D)
14. $V := V + \{v_i\}$
15. else
16. $n_{max} := \max(n_{max}, |X|)$
17. **for** all $v_i \in X$ **do**
18. $d_{som} = \min\left(\sum_i v_i, \sum_j w_j\right)$
19. done
20. Output((X, Y),d_{som})
21. end if

Figure 1. The Backtracking Scheme of the AMCSI Algorithm. Capital letters denote numbered sets, and small letters their elements, so that $X = \{x_i\}$. Please see details in the text.

It is obvious that the efficiency of a recursive algorithm depends exponentially on the number of recursive calls it makes, while the number of operations, performed within a single call, makes the base of that exponent. As may be seen from Figure 1, the number of recursive calls may be decreased if Extendable is able to determine whether the search leads to a desirable result on further recursion levels, and to return false if it does not.

Indeed, backtracking algorithms for graph matching use different techniques for terminating undesirable branches of the recursion tree. For example, the UA algorithm [7] looks for exact subgraph isomorphism only, therefore any recursion branch not leading to a common subgraph not coinciding with the lesser of the input graphs is rejected as soon as it is identified as such. In the CSI algorithm of McGregor [6], undesirable branches are defined as those not leading to subgraphs that have more than the maximal number of edges in common with subgraphs already found.

5. The AMCSI Algorithm

In most applications, only sufficiently large common subgraphs are considered as a useful result of graph matching. For example, clustering techniques based on graphs normally require only the maximal common subgraphs (MCSs) to be found. The CSIA algorithm has introduced a new parameter into the CSI problem, namely the *minimal* size of common subgraphs to be found, n_0. Using this parameter, AMCSI is able to reject branches of the recursion tree not leading to acceptable results, without spending time on finding them.

The central idea of UA [6] is based on using the vertex matching matrix (VMM) P. P_{ij} is true if vertex v_i of graph G_1 can be matched to vertex w_j of graph G_2, and false otherwise. Initially $P_{ij} = \mu(v_i,w_j)$, however as the solution extends, P_{ij} takes into account that not all label-like vertices can be matched due to possible conflicts between edges connecting them (cf. conditions (2)). Using matrix P, UA concentrates only on those same-label vertices that do not have edge conflicts with vertices of *already* found subgraphs. Naturally, in most cases the number of such candidate mappings decreases sharply, and P becomes very sparse with the size of common subgraphs found.The MSIA is therefore adopting a more efficient version of VMM, namely the matrix$M$, such thatM_{ij} gives the *index* of the vertex of graph G_2 that is mappable onto vertex v_i of G_2, $j = 1 \ldots L_i$. The 2-tuple $D = [M,L]$ is referred to as 'VMM D' in Figure 1, and its initialization is demonstrated by procedure *Initialize* in Figure 2. The matrix$M$ created by the procedure *initialize* will be a column matrix only with L=1.

Global integer n_0

procedure *Initialize* (VMMD = [M,L])

1. **for** all $v_i \in V$ **do**
2. $k := 0$
3. **for** all $w_j \in W$ **do**
4. **if** $\mu(v_i,w_j) =$ true **then**
5. $k := k + 1$, $M_{ik} := j$
6. **endif**
7. $L_i := k$
8. **done**

function *PickVertex* (VMMD = [M,L])
return v_i such that for all v_i, $v_j \in V - \{v_{x(k)}\}_k$, $0 < L_i \leq L_j$

function *GetMappableVertices* (vertex v_i, VMMD = [M,L])
return $\{w_{Mij}\}$ j=1….. L_i

function *Refine* (VMMD = [M,L])

1. VMM $D1$ = [T,N]
2. q := |X|
3. **for** all $v_i \in V - \{v_{x(i)}\} i$=1…q **do**
4. l := 0
5. **for** all $j \in \{M_{ik}\} k$=1….L_i **do**
6. **if** $v_{(ei,x(q)}$, $f_{j,y(q)})$ = true **then**
7. $l := l + 1$
8. $T_{il} := j$
9. **endif**
10. $N_i := l$
11. **done**
12. **return** D_1

function *Extendable* (VMMD_1 = [M,L])
1. $q := s :=$ |X|
2. **for** all $v_i \in V - \{v_{x(i)}\} i$=1… q **do**
3. **if** $L_i > 0$ **then** $s := s + 1$
4. **if** $s \geq \max(n_0, n_{\max})$ and $s > q$ **return** true **else return** false

Figure 2. Procedures and Functions used in the AMCSI Algorithm shown in Figure 1. Global data from Figure 1 are used. Please see details in the text.

The compact form of VMM (*i.e.*, the column matrix) drastically simplifies the identification of mappable vertices in procedure *Backtrack*. The function *GetMappableVertices* merely returns all vertices of graph G_2 indexed by the i^{th} row of matrix M (cf. Figure 2). Since the VMM M includes only mappings that do not violate conditions (2) for already mapped vertices, extension of X with index i allows for extension of Y with any index $j = M_{ik}$, $k \in \{1 \ldots L_i\}$. However, once a particular extension is done, the VMM must be refined in order to exclude mappings that are not compatible with the newly mapped vertices v_i and w_j.

VMM is refined in step 7 of the backtracking scheme (cf. Figure 1). The refinement is done by function *Refine*, shown in Figure 2. The procedure is based on the comparison of edges, connecting the last mapped vertices ($v_{x(q)}$ and $w_{y(q)}$, where $q = |X| = |Y|$ is the size of common subgraphs) with all candidate mappings in both graphs. If the edge between (unmapped) vertices v_i and $v_{x(q)}$ in graph G_1 is not compatible with the edge between its mapping candidates $w_{M(i)(k)}$ and $w_{y(q)}$ in graph G_2, the candidate is removed from the list (that is, removed from the i^{th} row of the VMM). It may be verified that checking only edges between candidate mappings and last mapped vertices is equivalent to checking the conditions of subgraph isomorphism (2) in full. Indeed, the initialization of VMM (procedure *Initialize*)

guarantees that only vertices with compatible labels are listed as mappable, so we never check vertex labels in *Refine*. Then, the refinement is done at each extension of the solution, which guarantees that *Refine* receives a VMM, in which conditions (2) hold true for all edges between unmapped and mapped vertices, except only for just mapped $v_{x(q)}$ and $w_{y(q)}$. It is therefore evident that using VMM drastically reduces the number of comparison operations performed on each recursion level.

As mentioned above, *Extendable* could always return true if there are unmapped vertices left in both graphs. Then each branch of the recursion tree would be explored to the very end of it. In order to terminate branches not leading to desirable solutions (cf. above) and thus save computational efforts, *Extendable* estimates the maximum possible size of common subgraphs that may be achieved by further continuation of the search. This is done on the basis of consideration that if, for yet unmapped vertex v_i from graph G_1, the number of mapping candidates from graph G_2 is zero, v_i will never be mapped (the mapping candidates are only removed as the search proceeds, cf. *Refine*). Therefore, the maximal possible size of AMCSI s is given by the size of the currently identified subgraphs $q = |X|$ plus the number of non-empty rows in VMM, $D = [M,L]$ (*i.e.*, those for which $L_i > 0$, see function *Extendable* in Figure 2). If s falls below n_0 (the minimal size of common subgraphs to be looked at) or is equal to q, *Extendable* returns false and the branch is abandoned. In the case of looking for maximal common subgraphs only, *Extendable* should also return false if s is less than the size of the largest common subgraphs found so far, $n_{\max}$ ($n_{\max}$ may be updated each time a common subgraph is output, cf. Figure 1, step 16). Then maximal common subgraphs will be found without finding *all* smaller subgraphs.

This procedure is illustrated by Figure 3, 4 and 5. Both of the two graph representations of the Figure 3 are taken from [1].The figure shows only one branch of the recursion tree starting with mapping (v_3,w_3), and includes only connected components. On the very first level of recursion $r = 1$, AMCSI removes vertex v_1,v_7,v_{10},v_{11} and v_{12} from graph G_1 (v_1,v_7 is not mappable because there is no vertex in G_2 with the same label, vertex v_{10},v_{11} and v_{12} are unmappable because of connectivity reasons). At r =2 AMCSI maps (v_2,w_2).On the next recursion level, $r = 3$, AMCSI explores two sub branches, *A* and *B*, which correspond to different paths from the vertex v_2. In sub branch *A*, mapping (v_6,w_6) makes vertex v_4,v_5 and v_8 unmappable because of connectivity reasons, and the subbranch results in a MCS of size 4. Branch *B* leads to the largest CSI of size 6. If parameter n_0 is set to 5, then sub branch *A* can be terminated on recursion level $r = 3$, because the sum of mapped and potentially mappable vertices in graph G_1 on that level is less than $n_0 = 5$. Thus, in this example, AMCSI abandons subbranch *A* before detecting MCS of size 4 (highlighted in Figure by shading). Although in this particular example the profit of avoiding level $r = 4$ of branch *A* does not look impressive, it is immense in most actual cases. Finally, consider function *PickVertex* (cf. Figure 2), which decides which vertex from graph G_1 should be chosen for mapping on a particular level of recursion.

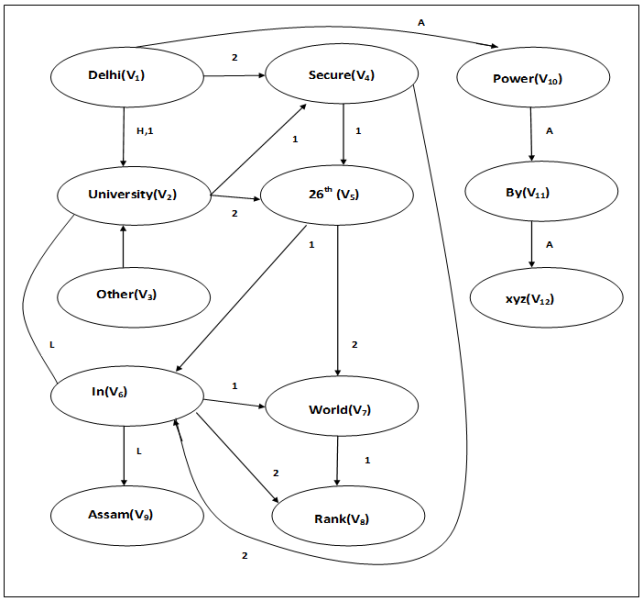

Figure 3(a)

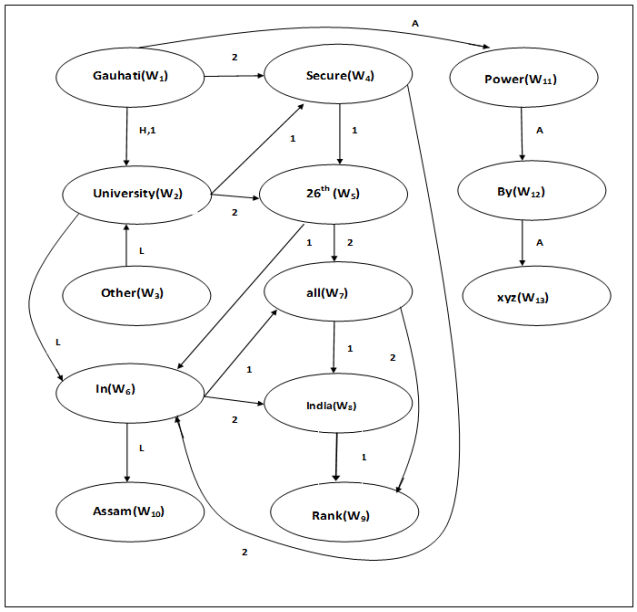

Figure 3(b)

Figure 3(a,b). The graph representations(G1,G2) of two similar kind of web pages taken from [1]. The document represented by Figure 3(a) has the title "Gauhati University", text containing "Gauhati University Secures 26th In All India Ranking", a link whose text reads "Other Universities In Assam", an address that contain "Powered by xyz". The document represented by Figure 3(b) has the title "Delhi University", a link whose text reads "Other Universities In Assam", an address that contain "Powered by xyz" and text containing "Delhi University Secures 26th In world Ranking".

Step 1: (r=1)

	Mapped	Candidates	Removed and Unmappable
X	Other (V_3,1)	University(V_2,5),In(V_6,6),Assam (V_9,1),Secure(V_4,4),26^{th}(V_5,4), Rank(V_8,2)	Delhi(V_1),World(V_7), Power(V_{10}),By(V_{11}),xyz(V_{12})
Y	Other (W_3)	University(W_2),In(W_6),Assam (W_{10}),Secure(W_4),26^{th}(W_5), Rank(W_9)	Guwahati(W_1),All(W_7),India (W_8)Power(W_{11}),By(W_{12}), xyz(W_{13})

Step 2: (r=2)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1), University (V_2,5)	In(V_6,6),Assam(V_9,1),Secure (V_4,4), 26^{th}(V_5,4), Rank(V_8,2)	Delhi(V_1),World(V_7), Power(V_{10}),By(V_{11}),xyz(V_{12})
Y	Other(W_3), University (W_2)	In(W_6),Assam(W_{10}),Secure(W_4), 26^{th}(W_5), Rank(W_9)	Guwahati(W_1),All(W_7), India(W_8), Power(W_{11}), By(W_{12}),xyz(W_{13})

Step3(A): (r=3)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1),University(V_2, 5), In(V_6,6)	Assam(V_9,1)	Delhi(V_1),World(V_7),Power(V_{10}), By(V_{11}), xyz(V_{12}), Secure(V_4), 26^{th}(V_5), Rank(V_8)
Y	Other(W_3),University(W_2), In(W_6)	Assam(W_{10})	Guwahati(W_1),All(W_7),India(W_8), Power(W_{11}), By(W_{12}),xyz(W_{13}), secure(W_4),26^{th}(W_5), Rank(W_9)

Step 3(B): (r=3)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1),University (V_2,5),Secure(V_4,4)	In(V_6,6),Assam (V_9,1),26^{th}(V_5,4), Rank(V_8,2)	Delhi(V_1),World(V_7), Power(V_{10}),By(V_{11}), xyz(V_{12})
Y	Other(W_3),University (W_2),Secure(W_4)	In(W_6),Assam (W_{10}),26^{th}(W_5), Rank(W_9)	Guwahati(W_1),All(W_7), India(W_8),Power(W_{11}), By(W_{12}),xyz(W_{13})

Step 4: (r=4)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1),University (V_2,5), Secure(V_4,4), 26^{th}(V_5,4)	In(V_6,6),Assam (V_9,1),Rank(V_8,2)	Delhi(V_1),World(V_7), Power(V_{10}),By(V_{11}),xyz(V_{12})
Y	Other(W_3),University (W_2),Secure(W_4), 26^{th}(W_5)	In(W_6),assam(W_{10}), Rank(W_9)	Guwahati(W_1),All(W_7),India(W_8)Power(W_{11}),By(W_{12}),xyz(W_{13})

Step5: (r=5)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1),University (V_2,5), Secure(V_4,4),26th (V_5,4), In(V_6,6)	Assam(V_9,1)	Delhi(V_1),World(V_7),Power(V_{10}), By(V_{11}), xyz(V_{12}), Rank(V_8)
Y	Other(W_3),University(W_2), Secure(W_4), 26th(W_5), In(W_6)	Assam(W_{10})	Guwahati(W_1),All(W_7),India(W_8), Power(W_{11}),By(W_{12}),xyz(W_{13}), Rank(W_9)

Step 6: (r=6)

	Mapped	Candidates	Removed and Unmappable
X	Other(V_3,1),University(V_2,5), Secure(V_4,4), 26th(V_5,4),In(V_6,6), Assam(V_9,1)		Delhi(V_1),World(V_7),Power(V_{10}), By(V_{11}), xyz(V_{12}), Rank(V_8)
Y	Other(W_3),University(W_2), Secure(W_4), 26th(W_5), In(W_6), Assam(W_{10})		Guwahati(W_1),All(W_7),India(W_8), Power(W_{11}),By(W_{12}),xyz(W_{13}), Rank(W_9)

Figure 4. Example of the recursive AMCSI algorithm functionality. *r* stands for the recursion level, *X* and *Y* list vertices of graphs G_1 = [*V,E,n*] and G_2 = [*W, F,m*], respectively. Mapped vertices are aligned vertically in 'mapped' cells. Different patterns of graph vertex symbols denote different vertex labels. Only one branch of the recursion tree, starting from mapping (v_3,w_3), is shown, and only connected common subgraphs are considered. At minimal requested size of CSI, n_0 = 5, the last mapping on the recursion level *r* = 3 in sub-branch *A* (highlighted by shaded background) is terminated by AMCSI because the current size of graph G_1 plus the number of candidate matchings on level *r* = 3, sub-branch *A*, is less then n_0, see text for more details.

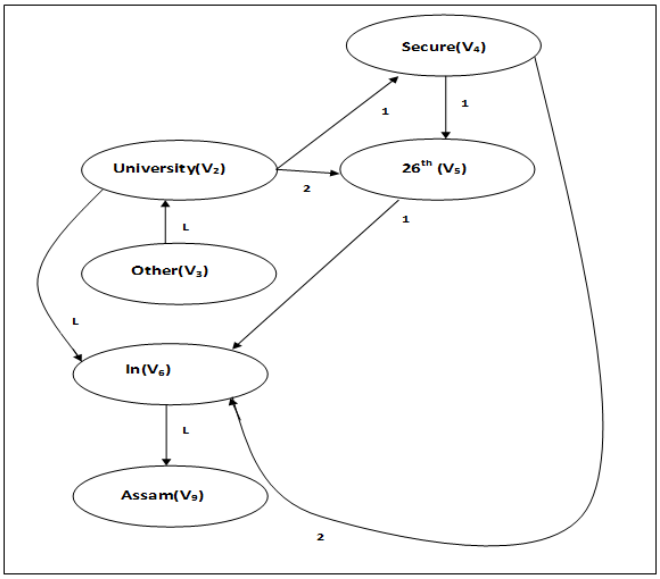

Figure 5. The MCS Obtained

6. Calculation of the Distance between the Two Graphs

To calculate the distance between the two graphs by using the above mentioned (Eq.1) distance measure which was proposed by us [1], we must have the numerical values of the following:

(i) $\max(\sum_{SOM} d^{\pm} MCS(G_1,G_2))$

(ii) $\max(\sum d^{\pm}(G1),(\sum d^{\pm}(G_2))$

To calculate the distance between the two graphs by using the prevalent distance measure as stated below

$$dist_{MCS}(G_1,G_2) = 1 - \frac{|mcs(G_1,G_2)|}{\max(|G_1|,|G_2|)} \quad \ldots\textbf{(3)}$$

we must have the numerical values of the following:

(iii) $\max(|G_1|,|G_2|)$

(iv) $|MCS(G_1,G_2)|$

From Figure 4 and 5 we obtain:

Table 1. Calculation of the Distance between G_1 and G_2

Method of Calculation	(i)	(ii)	(iii)	(iv)	$dist_{MCS}(G_1,G_2)$
Proposed	21	38			0.44737
Prevalent			6	13	0.53846

From Table 1 it is clear that the distance between two web pages (i.e. graphs G_1 and G_2) is found to be grater (|Prevalent - Proposed|=0.09103) in case of the prevalent method. The two web pages have an actual difference of only 3 words out of a maximum of 13 significant words of the second web page (represented by graph G_2). Also, it is seemed to be obvious that the two web pages are of very similar kind.

Hence it may be inferred that the proposed method for calculating the difference between graphs representing web documents is capable of yielding batter result (16.9% approx in case of the above example) then the prevalent one. However to establish this inference statistically, we must perform experiments in large scale which is not possible for us within the scope of our research. In one of our article [1] we showed this usefulness by arbitrarily determining the different numerical values required to calculate the distance. In this article we are considering the same example to calculate the distance by following a concrete procedure and are getting slight different but more desired result.

7. Median of a Set of Graphs

To form clusters from a set of data items it is necessary to compute the distance between data items and cluster centers. In our approach it follows that the cluster centers (centroids) must also be graphs. Therefore, we have to compute the representative "centroid" of a cluster as the median graph of the set of graphs in that cluster (Eq. 4).

The median of a set of graphs S is a graph $g \in S$ ($S = \{G_1, G_2, ..., G_n\}$) such that g has the lowest average distance to all elements in S [8]:

$$g = \arg\min_{\forall s \in S} \left(\frac{1}{|s|} \sum_{i=1}^{|S|} dist(s, G_i) \right) \quad \ldots \textbf{(4)}$$

since $g \in S$, it is straightforward (and relatively inexpensive) to simply compute the average distance to all graphs for each graph in *S*.

8. Conclusion

In this article we have introduced a new approach for graph distance measure which is useful to cluster data sets (in this case web documents) when utilizing more descriptive graphs in lieu of the usual case of vector representations. Our first contribution is presenting an efficient model by which web document content can be modeled as graphs. These graph representations retain information that is usually lost when using a vector model, such as term order and document section information. We have demonstrated how with the composite model of graph representation, we can perform the graph similarity task in $O(n^2)$ time ,*n* being the number of nodes. In general, graph similarity using maximum common subgraph is an NP-complete problem, so this is an important result that allows us to forgo sub- optimal approximation approaches and find the exact solution in polynomial time.

While AMCSI has been designed for use in web document management, the algorithm is conceptually general and is directly applicable to any graph-based similarity application. The new algorithm presented in this paper is highly recommended for applications where we are faced with large labeled and directed graphs. By contrast, the proposed algorithm is not suitable to be applied to problems dealing with unlabeled, undirected, highly disjoint graphs.

The experimental verification remains for future work on the algorithm.

Acknowledgments

The author would like to extend his gratitude to Prof. H.K. Baruah, Gauhati University for his valuable support in preparing this research article. The author would also like to thank the reviewers for their helpful comments and suggestions for improving the manuscript.

References:

[1] K. K. Phukon, "A Composite Graph Model for Web Document and the MCS Technique", International Journal of Multimedia and Ubiquitous Engineering, vol. 7, no. 1, SERSC, **(2012)** January.

[2] K. K. Phukon, "The Composite Graph Model for Web Document and its Impacts on Graph Distance Measurement", International Journal of Energy, Information and Communications, vol. 3, Issue 2, SERSC, **(2012)** May.

[3] E. B. Krissinel and K. Henrick, "Common subgraph isomorphism detection by backtracking search", Softw. Pract. Exper., vol. 34, **(2004)**, pp. 591–607 (DOI: 10.1002/spe.588).

[4] G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs", Calcolo., vol. 9, **(1972)**, pp. 341–354.
[5] C. Bron and J. Kerbosch, "Algorithm 457—finding all cliques of an undirected graph", Communications of the ACM, vol. 16, **(1973)**, pp. 575–577.
[6] J. McGregor, "Backtrack search algorithms and the maximal common subgraph problem", Software—Practice and Experience, vol. 12, **(1982)**, pp. 23–34.
[7] J. R. Ullman, "An algorithm for subgraph isomorphism", Journal of the Association of Computers and Machines, vol. 23, no. 1, **(1976)**, pp. 31–42.
[8] A. Schenker, H. Bunke, M. Last, A. Kandel, "Graph Theoretic Techniques for Web Content Mining", Series in Machine Perception and Artificial Intelligence, vol. 62, Copyright © 2005 by World Scientific Publishing Co. Pte. Ltd., **(2005)**.

Author

Kaushik Kishore Phukon, Assistant Professor (IT), Department of Commerce, Gauhati University, India received MCA degree from Jorhat Engineering College (Under Dibrugarh University), India in 2009. He is currently pursuing doctoral research at Department of Computer Science, Gauhati University, Assam, India. His research interests include representation of web documents using graphs and graph based clustering and classification.