A Graph-Decomposition Algorithm for Graph Optimal Monomorphism

Yasser El-Sonbaty
Dept. of Elect. & Computer Eng.
Arab Academy for Sc. and Tech.
Alexandria 1029, EGYPT

M. A. Ismail
Dept. of Computer Sci.
Faculty of Engineering
Alexandria 21544, EGYPT

Abstract

In this paper a new heuristic algorithm for graph optimal monomorphism is proposed. The main idea of the new algorithm is to decompose the graphs to be matched into smaller subgraphs. The matching process is then done at the level of the decomposed subgraphs based on the concept of error-correcting transformations. The cost of matching two graphs is defined as the minimum of a weighted bipartite graph constructed from the decomposed subgraphs. The average computational complexity of the proposed algorithm is found to be $O(N^4)$ which is better than many similar existing techniques.

1- Introduction

Describing structured objects in terms of their subpatterns, primitives and mutual relations, plays an important role in many applications in pattern recognition, image analysis and computer vision [1].

Attributed relational graphs *ARGs* are one of the most powerful tools in describing structured objects. In this representation, nodes represent primitives or subpatterns of structured objects and branches between nodes represent relations between primitives or subpatterns [1].

One way to recognize the structure of an unknown pattern is to transform it into an *ARG,* then match it with other *ARG*s representing structures of prototype patterns. This process of matching is called graph isomorphism. Formally, two graphs G and G' are said to be isomorphic (to each other) if there is a one-to-one correspondence between their vertices and between their edges such that incidence relationship is preserved [2]. If the isomorphism is encountered between a graph and a subgraph of another larger graph, then the problem is called graph monomorphism. The notion of optimality becomes important in graph monomorphism when a perfect matching between the attributed values of the two graphs may not be available [3].

Inexact subgraph matching of weighted relational graphs has been investigated by Shapiro and Haralick [4]. Relaxation methods were used in Kitchen [5], Davis [6], Li [7], Christmas *et al.* [8] and Berthod *et al.* [9]. Ghahraman *et al.* [10] proposed an algorithm based on getting subgraphs of the Cartesian graph product. Subgraph error-correcting isomorphism was introduced by Tsai and Fu [11] and Eshera and Fu [12,13]. An algorithm for graph optimal monomorphism was proposed by Wong *et al.* [3]. The approach of neural networks was also used for solving the problem of graph isomorphism like in [14-16]. Some recent techniques for graph matching were introduced by Gold and Rangarjan [17] and by Depero *et al.* [18].

Most of the techniques mentioned above and which deal with the problem of graph optimal monomorphism for attributed relational graphs have the problem of computational complexity as it may grow exponentially when increasing the sizes of matched graphs.

The core of this paper is to introduce a new heuristic algorithm for solving the problem of graph optimal monomorphism based on the idea of decomposing the graphs into smaller subgraphs and perform the matching between the graphs at the level of their decomposed subgraphs. The main advantage of the proposed algorithm is its average computational complexity which is $O(N^4)$.

The description of graph decomposition and how to match the decomposed subgraphs are shown in section 2. Section 3 introduces the proposed algorithm for graph optimal monomorphism with analysis of its computational complexity. Experimental results and conclusions are presented in sections 4 and 5 respectively.

2- Graph Decomposition

In this section the description of how to decompose a graph into smaller subgraphs, and how to match these decomposed subgraphs are proposed.

2-1- Decomposition of Attributed Relational Graphs.

The main idea of decomposing the *ARGs* is that, the smaller the graph structure, the easier the matching process, and this well result in lower complexity than matching the original *ARGs* which is the main contribution of this paper.

The graphs resulting from decomposing an *ARG* are call*ed Basic Attributed Relational Graphs* (*BARGs*) and this notion is adopted from [13]. The *BARG* is in the form of one level tree which consists of a node and all nodes connected to that node whether the connected branches are in or out of that node. *BARG*s are different from those in [13] where only emanating branches and nodes on which these branches terminate are encountered. The new definition gives the matching process more stability and preserves the structure of the original graph.

2-2- Matching BARGs

Matching *BARG*s is much easier than matching *ARG*s and this is because the structure of *BARG*s is simpler and easier in processing than *ARG*s. In this paper, we use the concept of error-correcting transformation used before in matching *ARG*s for matching *BARG*s. The cost of matching two *ARG*s is defined as the cost of the sequence of transformations that possesses minimum total cost and that must be performed on one of the two *ARG*s in order to produce the other *ARG*. The operations permitted to transform one *ARG* to the other are : *node insertion*, *node deletion*, *branch insertion*, *branch deletion, node label substitution* and *branch label substitution*. A cost (weight) is associated with each operation and its value is determined by an optimization procedure or heuristically. The costs corresponding to each operation are : w_{ni}, w_{nd}, w_{bi}, w_{bd}, w_{ns}, and w_{bs} respectively. This type of distance is logical and intuitive but its main problem is the complexity that may grow up exponentially when increasing the graph size. Some research in this area can be found in [1,11-13].

A new operation *structure preservation* is added here with cost w_{sp}. The main function of the new operation is to help preserving the global structure of the original *ARG* after performing the operation of graph decomposition and its use will be declared later in this section.

Given two *BARG*s *U* and *V*, as shown in Fig. 1, the cost of matching *U* and *V* is

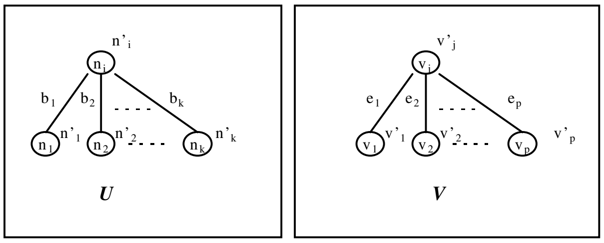


Fig. 1. Two *BARG*s; *U* and *V*

$$\text{Distance}\ (U,V) = w_{ns}\text{*dist}(n_i, v_j) + \min(w_{ni}, w_{nd})\text{*abs}(k\text{-}p) + \min(w_{bi}, w_{bd})\text{*abs}(k\text{-}p) + \text{dist}(b\text{'s},e\text{'s}) \quad \textbf{(1)}$$

where

- dist(n_i,v_j) is the distance between node labels of n_i and v_j and node attributes of n'_i and v'_j, and is computed depending on their data types,
- *k* and *p* are the number of branches connected to the root node of the *BARG*s being matched, and.
- dist(*b*'s,*e*'s) is calculated as the minimum of a weighted bipartite graph constructed from *b*'s and *e*'s as their nodes and its structure is as shown in Fig. 2.

There are many algorithms [19-21] for computing the minimum of a weighted bipartite graph which is defined as the complete matching with the sum of the weights of matching edges is minimum. Doing so is known to have $O(N^2)$ average case complexity, and $O(N^3)$ worst case complexity [21].
the weight of any branch connecting two nodes, say b_f and e_h in the bipartite graph is the distance between b_f and e_h and is calculated as follows :

$$\text{distance}(b_f, e_h) = w_{bs} * \text{dist}(b_f, e_h) + w_{sp} * \text{dist}(n_f, v_h) \quad \textbf{(2)}$$

where dist(*x*,*y*) is the distance between *x* and *y*, and their attributes, which is computed based on their data type.

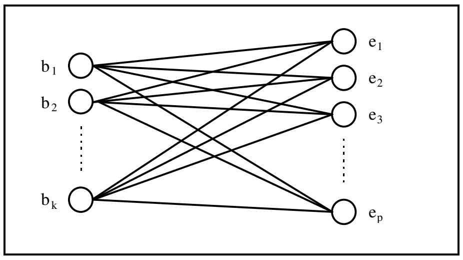
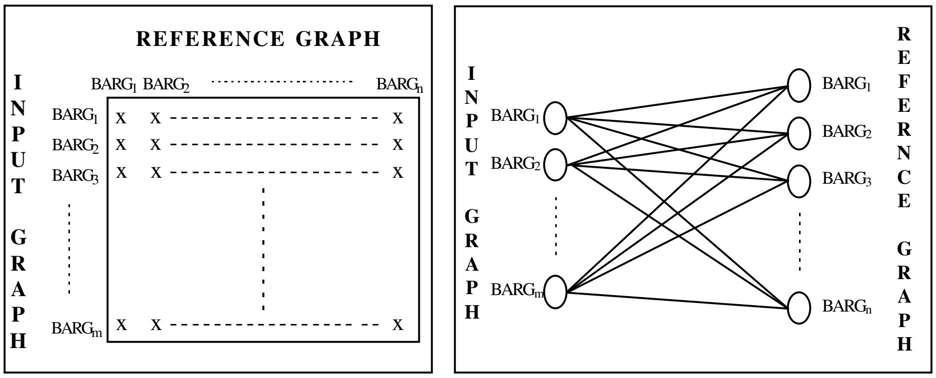


Fig. 2. Weighted bipartite graph corresponding to branches distance

3- The Algorithm

The main idea of the proposed algorithm is to decompose both reference and input graphs; say G_r and G_i respectively; to be matched into *BARG*s as previously mentioned in section 2-1. A distance matrix "*D*" between reference and input graphs is then constructed and is equivalent to the distances between the *BARG*s of both reference and input graphs. The labels of the rows in the distance matrix represent the *BARG*s of the input graph, while the labels of the columns represent the *BARG*s of the reference graph. $D_{i,j}$ represents the distance between the i^{th} *BARG* in the input graph and the j^{th} *BARG* in the reference graph and is calculated as mentioned in section 2-2.

After calculating the distance matrix *D*, a weighted bipartite graph is constructed which is equivalent to the distance matrix *D* where each branch connecting two nodes, one represents a *BARG* from the input graph and the other node represents a *BARG* from the reference graph, has a weight equivalent to the distance between the two *BARG*s connected by this branch as shown in Fig. 3.



(a) (b)

Fig. 3. (a) the distance matrix between input and reference graphs, (b) weighted bipartite graph corresponding to the distance matrix.

The minimum of the weighted bipartite graph which is equivalent to the distance matrix is considered as the cost of matching the input and the reference graphs.

Every pair of *BARG*s (one *BARG* from input graph and the other is from reference graph) in the weighted bipartite graph connected by a branch whose weight is taken in calculating the minimum of the weighted bipartite graph, is considered to be matched, i.e., the node rooted the *BARG* of input graph matches the node rooted the *BARG* of reference graph.

3-1- Proposed Algorithm

The proposed algorithm can be summarized in the following steps :

[1] : **Read** both *reference* and *input* graphs.

[2] : **Decompose** (*reference* graph)

[3] : **Decompose** (*input* graph)

[4] : **For** i=1 to No_of_nodes(*input* graph)

  For j=1 to No_of_nodes(*reference* graph)

    Distance_matrix $[i,j]$ = distance between i^{th} *BARG* of *input* graph and j^{th} *BARG* of *reference* graph { review equations 1,2 }

[5] : **Construct** the weighted bipartite graph equivalent to the *Distance_matrix*.

[6] : *Cost_of_matching* = the minimum of the weighted bipartite graph.

3-2- Analysis of computational complexity

It is well known that the graph matching problem belongs to the class of *NP* problems and any conventional search algorithm will require exponential time. The proposed cost of matching is defined as the minimum of the weighted bipartite graph constructed from the *BARG*s of both input and reference graphs as its nodes and the weight associated with a branch connecting two *BARG*s is defined as the minimum total-cost sequence of error transformations that is required to transform one *BARG* to the other. In this section we investigate the computational complexity of the proposed approach.

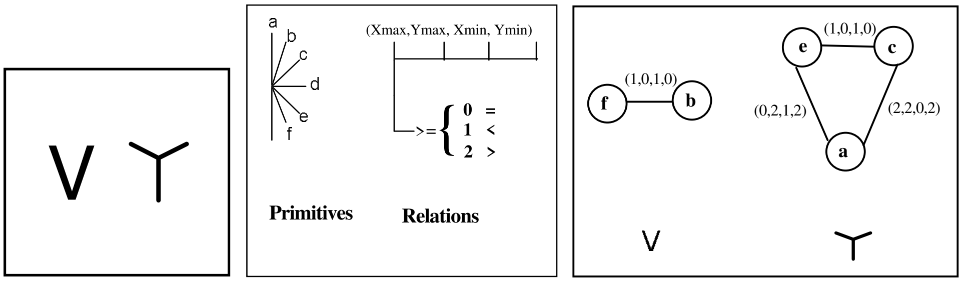
Suppose that we are given a reference graph R with M nodes and an input graph I with N nodes. The first step in the algorithm is to decompose both R and I into *BARG*s. It is obvious that this step is done in quadratic time for each graph, i.e., has time complexity of $O(M^2+N^2)$.

The second step in the algorithm is to calculate the distance between all the *BARG*s of input and reference graphs. The computational complexity of matching two *BARG*s mainly depends on the calculation of error-transformations between the two *BARG*s and the computational complexity of getting the minimum of a weighted bipartite graph. The computational complexity of calculating the error-correcting transformations is $O(\delta^2)$, where δ denotes the maximum number of branches connected to any node in the matched *BARG*s. The average computational complexity of getting the minimum of a weighted bipartite graph is $O(M*N)$ [21], assuming that any node is connected to all other nodes in the graph. so we can consider the computational complexity of matching two *BARG*s is $O(M*N)$,. The calculation of the distance between the *BARG*s is repeated for all the *BARG*s of both input and reference graphs, and that means, the average computational complexity of second step is $O(M^2*N^2)$, where $M*N \geq \delta^2$.

The third step is to calculate the cost of matching input and reference graphs and which is defined as the minimum of the weighted bipartite graph constructed in step 2 and the computational complexity of this step is $O(M*N)$.

In summary the average computational complexity of the proposed algorithm in calculating the cost of matching two *ARG*s with M and N nodes is $O(M^2*N^2)$. It can be easily proven that the average computational complexity of the proposed algorithm with respect to the number of edges is $O\ (\ l\ \ p\ )$, where l and p are the number of edges in the two graphs. The computational complexity of the new algorithm is much better than the graduated assignment algorithm [17] that has a low order complexity of $O\ (l\ \ p)$ and also than algorithms [1, 3, 11-13] which have exponential complexity.

Illustrative example: assume we need to match the two letters shown in Fig. 4-(a). The primitives and relations used in representing these letters are shown in Fig. 4-(b), where (X_{max}, Y_{max}, X_{min}, Y_{min}) shows the relations between the *x*-*y* coordinates of two primitives. The graphs equivalent to the letters are shown in Fig. 4-(c). Fig. 5 shows the distance matrix equivalent to the graphs shown in Fig. 4-(c) where we assumed the distance between two primitive is equal to the absolute difference of their order and the distance between any two relations is equal to the sum of the binary distance (0 when identical, 1 otherwise) between the four relations X_{max}, Y_{max}, X_{min} and Y_{min}. the values of w_{ni}, w_{nd}, w_{bi}, w_{bd}, w_{bs}, w_{ns} and w_{sp} are all assumed to equal 1. In getting the minimum of the weighted bipartite graph equivalent to the distance matrix, the cost of matching the two letters is 3+3 = 6. Fig. 6 shows the mapping between matched strokes in the two letters obtained from the minimum weighted bipartite graph in Fig. 5.



(a) (b) (c)

Fig. 4. (a) letters to be matched, (b) primitives and relations, (c) graph representation of the two letters.

	e: (0,2,1,2) c, (1,0,1,0) a	c: (1,0,1,0) e, (2,2,0,2) a	a: (0,2,1,2) e, (2,2,0,2) c
f (1,0,1,0) b	3	7	11
b (1,0,1,0) f	7	3	6

Fig. 5. Distance matrix of graphs in Fig. 5-(c)

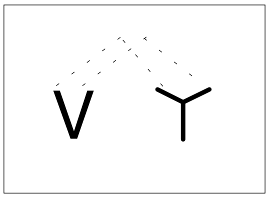
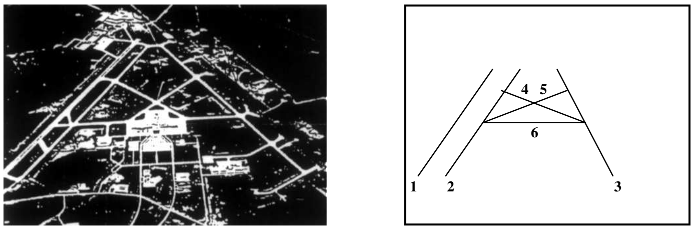


Fig. 6. Matched strokes

4- Experimental Results

In this section, the proposed algorithm is tested using some experiments that show its efficiency and capability of handling various *ARG*s with different structures.

Experiment 1: The problem is to identify an image graph of the prominent runways of the Jacksonville international airport from the picture shown in Fig. 7.[3]. The image graph has to be matched with three runway models of the open-V runway configuration shown in Fig 8- (a)-(c). Each airport is represented by an attributed graph where the vertex representing the runway with an attribute equivalent to the runway length. The attributed values of an edge are the central-to-center distance between connected runways and the cute distance angle between them [3].



(a) (b)

Fig. 7. (a) Photograph of Jacksonville International Airport [3], (b) Image graph of Jacksonville International Airport.

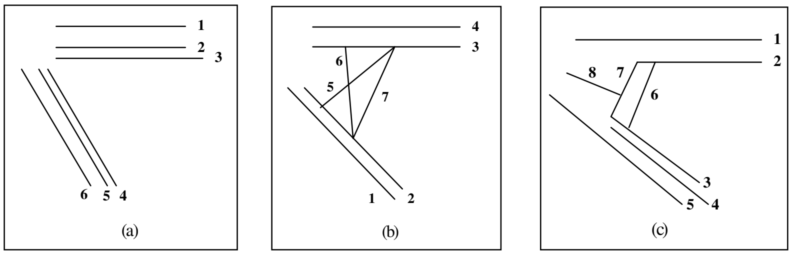


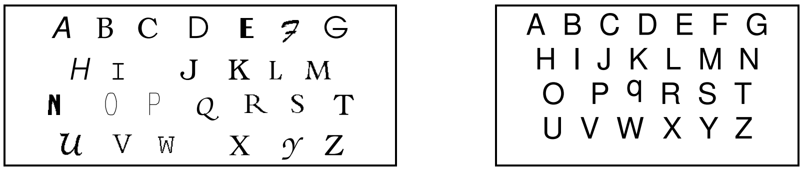
Fig. 8. (a) Runway model A., (b) Runway model B., (c) Runway model C

In this example, we find the optimal matching between the image graph and each of the three models. Table I shows the output of the proposed algorithm, which indicates that the image graph of the Jacksonville runways and runway model B incur the lowest cost of matching. The results of matching are consistent with the results published in [3].

Base Graph	**Matching Cost**	**Vertex Mapping**
Model A	5660	(1,6),(2,5),(3,1),(4,2),(5,4),(6,3)
Model B	3847	(1,1),(2,2),(3,4),(4,5),(5,6),(6,7)
Model C	4430	(1,5),(2,4),(3,1),(4,2),(5,3),(6,8)

Table I. Results of graph optimal monomorphism.

Experiment 2: In this experiment, an application from character recognition is used. The problem is to match a group of multi-font characters shown in Fig. 9-(a) with a set of normal-font characters shown in Fig 9-(b).



(a) (b)

Fig. 9. (a) Multi-font characters, (b) normal-font characters

Each character is represented as an *ARG*, where nodes represent primitives which are shown in Fig. 10. The attributes of an edge, connecting nodes n_i and n_j, are in the form of (X_{max}, Y_{max}, X_{min}, Y_{min}, *Connection*), where X_{max}, Y_{max}, X_{min}, and Y_{min} have the values 0, 1 or 2 when *X-Y* coordinates of n_i and n_j are in the relation =, <, or > respectively. *Connection* has the value of 1 when n_i and n_j are connected, 0 otherwise.

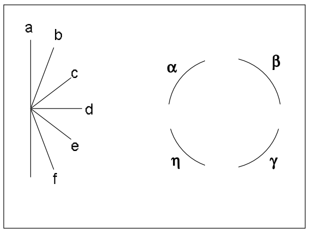


Fig. 10. Primitives used in experiment 2.

(A,A)	(B,B)	(C,C)	(D,D)	(E,E)	(F,Z)
(G,G)	(H,H)	(I,T)	(J,J)	(K,K)	(L,L)
(M,M)	(N,N)	(O,O)	(P,P)	(Q,O)	(R,R)
(S ,S)	(T,T)	(U,U)	(V,V)	(W,W)	(X,X)
(Y, Y)	(Z,Z)				

Table II- Results of experiment 2.

Each character in Fig. 9-(a) is matched with all characters Shown in Fig. 9-(b) and the character with the minimum distance is selected as the matched character. Table II shows the results of the matching process.

From Table II, the multi-font characters are correctly mapped to their equivalent normal-font characters except F, I and Q because these characters are closer in structure to Z , T and O than F , I and q.

Experiment 3: In this experiment we use simulation method to test the performance of the new algorithm in handling sparsely connected graphs. Attributed relational graphs of size 100 are generated with 10% degree of connectivity. The weights of edges are real numbers and are produced randomly in the range of {0 - 1}. Nodes have five random binary valued attributes. After the generation of the graphs, 60% or 80% nodes are deleted and uniform noise in the range of {0% - 20%} is added to the edges. One hundred trials are run at each noise level. The results of the proposed algorithm in comparison with the results obtained from the algorithm of graduated assignment [17] are shown in figures 11-12 where (a) is the proposed algorithm and (b) is the graduated assignment algorithm. The percent of incorrect matches is calculated as the average percent of mismatched nodes over the 100 trials. From figures 11-12, it can be concluded that the proposed algorithm has lower percentage of incorrect matches than the recent technique of graduated assignment [17].

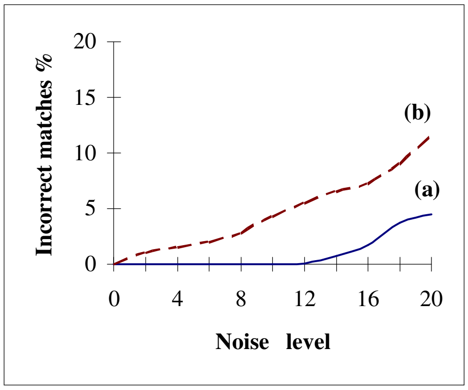


Fig. 11 Results of matching *ARG*s with 5 binary attributes and 60% deleted.

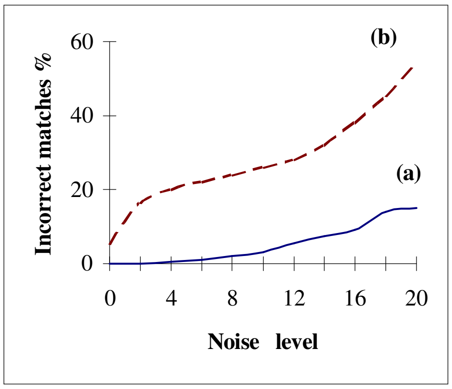


Fig. 12 Results of matching *ARG*s with 5 binary attributes and 80% deleted.

5- Conclusions

In this paper, a new heuristic algorithm for graph optimal monomorphism is proposed. The proposed algorithm decomposes the reference and input graphs into smaller subgraphs called *BARG*s. A weighted bipartite graph is then constructed with the *BARG*s as its nodes and the weight associated with a branch is calculated as the distance between the two *BARG*s connected by this branch. The distance is calculated based on the concept of error-correcting transformations. The graph optimal monomorphism is found to be the minimum of this weighted bipartite graph. From experimental results and complexity analysis, it can be concluded that the new algorithm proved its superiority over other recent techniques reported in the literature in matching attributed relational graphs, and it has lower complexity as the average computational complexity of the proposed algorithm is found to be $O(M^2*N^2)$. Also, the proposed algorithm is parallel in nature and can take advantage of hardware parallel architecture.

References

1- A. Sanfeliu & K. S. Fu, A Distance Between Attributed Relational Graphs for Pattern Recognition, IEEE Trans. on Sys., Man and Cybernetics, Vol. 13, No. 3, P. 353- 362, 1983 .

2- Narsingh Deo, Graph Theory with Applications to Engineering and Computer Science, Printice Hall, Inc., 1995.

3- A. K. C. Wong., M. You & S. C. Chan, An Algorithm for Graph Optimal Monomorphism, IEEE Tran. on Sys., Man and Cybernetics, Vol. 20, No. 3, P. 628-636, 1990.

4- L. G. Shapiro & R. M. Haralick, Structural Descriptions and Inexact Matching, Tech. Rep. CS-79011-R, Virginia Polytech. Inst. and State Univ., Blacksburg VA, Nov. 1979.
5- L. Kitchen, Relaxation Applied to Matching Quantitative Relational Structures, IEEE Tran. on Sys., Man and Cybernetics, Vol. 10, No. 2, 1980.
6- L. S. Davis, Shape Matching Using Relaxation Techniques, IEEE on PAMI, Vol. 1, PP. 60-72, 1979.
7- S. Z. Li, Matching: Invariant to Translation, Rotation, and Scale Changes, Pattern Recognition, Vol. 25, PP. 583-594, 1992.
8- W. J. Christmas, J. Kittler and M. Petrou, Structural Matching in Computer Vision Using Probabilistic Relaxation, IEEE Trans. on PAMI, Vol. 17, No. 8, PP. 749-764, 1995.
9- M. Berthod., M. Kato & J. Zerubia, DPA: A Deterministic Approach to the MAP Problem, IEEE Trans. on Image Processing, 1996.
10- D. E. Ghahraman, A. K. C. Wong & T. Au, Graph Optimal Monomorphism Algorithms, IEEE Tran. on Sys., Man and Cybernetics, Vol. 10, No. 4, 1980.
11- W. Tsai & K. S. Fu, Subgraph Error-Correcting Isomorphisms for Syntactic Pattern Recognition, IEEE Tran. on Sys., Man and Cybernetics, Vol. 13, No. 1, P. 48-62, 1983.
12- M. A. Eshera & K. S. Fu, An Image Understanding System Using Attributed Symbolic Representation and Inexact Graph Matching, PAMI, Vol. 8, No. 5, P. 604-617, 1986.
13- M. A. Eshera & K. S. Fu, A Graph Distance Measure for Image Analysis, IEEE Tran. on Sys., Man and Cybernetics, Vol. 14, No. 3, P.398-408, 1984.
14- S. S. Young, P. D. Scott & N. M. Nasrabadi, Object Recognition Using Multilayer Hopfield Neural Networks, Proc. IEEE Conf. Computer Vision & Pattern Recognition, PP. 417-422, 1994.
15- T. W. Chen and W. C. Lin, A Neural Networks Approach to CSG-Based 3D Object Recognition, IEEE Trans. on PAMI, Vol. 16, PP. 719-725, 1994.
16- P. Suganthan, E. Teoh & D. Mital, Pattern Recognition by Graph Matching Using the Potts MFT Neural Networks, Pattern Recognition, Vol. 28, PP. 997-1009, 1995.
17- S. Gold and A. Rangarajan, A Graduated Assignment Algorithm for Graph Matching, IEEE Trans. on PAMI, Vol. 18, No. 4, PP. 377-388, 1996.
18- F. Depiero, M. Trivedi & S. Serbin, Graph Matching Using a Direct Classification of Node Attendance, Pattern Recognition, Vol. 29, No. 6, PP. 1031-1048, 1996.
19- H. W. Kuhn, The Hungarian Method for the Assignment Problem, Naval Res. Log. Quart., No. 12, P. 83-97, 1955.
20- R. M. Karp, An Algorithm to Solve the m*n Assignment Problem in expected time O(m n log n), Networks, No. 10, P. 143-152, 1980.
21- S. Yamada and T. Hasai, An Efficient Algorithm for the Linear Assignment Problem, Elect. and Comm. in Japan, Part 3, Vol. 73, No. 12, P. 28-36, 1990.