A New Algorithm for Inexact Graph Matching[1]

Adel HLAOUI and Shengrui WANG
Université de Sherbrooke, Faculté des Sciences / DMI
Sherbrooke, Québec, J1K2R1, Canada
{hlaoui, wang}@dmi.usherb.ca

Abstract

The graph is an essential data structure for representing relational information. When graphs are used to represent objects, comparing objects amounts to graph matching. Inexact graph matching is the process of finding the best possible matching between two graphs when exact matching is impossible. In this paper, we propose a new algorithm for the inexact matching problem. The new algorithm decomposes the matching process into K phases, where the value of K ranges from 1 to the minimum of the numbers of nodes in the two graphs to be matched. The efficiency of the new algorithm results from the use of small values of K, significantly reducing the search space while still producing very good matchings (most of them optimal) between graphs. The algorithm is compared with the error-correcting subgraph isomorphism algorithm based on A.*

1. Introduction

The graph is one of the most important data structures in pattern recognition research. It is an essential structure for representing relational information. In our research, we are interested in using graphs to represent image content. Our goal is to develop efficient algorithms for indexing and retrieving images based on their graph representations. One basic requirement for achieving this goal is an efficient graph matching algorithm in order to evaluate the similarity between graphs. A well-known procedure for inexact graph matching, called the error-correcting subgraph isomorphism algorithm, is given in[1],[2]. The algorithm is based on the A* method and is able to find the best matching by exploring only the most promising avenues. However, the algorithm runs into combinatory explosion when the size of the graphs becomes large (e.g. over 10 nodes). Another well-known algorithm, reported by Ullman[4], utilizes the backtracking technique with forward checking. Ullman's algorithm also suffers from the combinatory explosion problem and is appropriate only for exact graph matching. Finally, graph matching is usually applied to two graphs at a time. Bunke and Messmer [5] proposed a new approach to solve the graph matching problem for a graph database. It involves decomposing the graph into subgraphs. A subgraph which appears multiple times will be compared only once to the input graph.[3],[5].

In this paper, we propose a new graph matching algorithm for computing the similarity between graphs. In the new algorithm, we make use of the similarity matrix as defined in Ullman's algorithm. The similarity matrix plays an important role in selecting match candidates. Like the error-correcting algorithm, the new algorithm is designed to perform inexact matching. Edit operations similar to those in Bunke-Messmer's algorithm are used.

With this new algorithm, we have proposed an efficient way to explore the best potential matchings, thus significantly reducing computation time relative to the error-correcting subgraph isomorphism algorithm based on A*, without any significant loss in matching results.
The outline of this paper is as follows. In Section 2, we describe the error-correcting subgraph isomorphism algorithm based on A*. In Section 3, we introduce our algorithm and present experimental results to demonstrate the performance of the new algorithm. Finally, Section 4 concludes this report.

2. The error-correcting subgraph isomorphism algorithm based on A*

In this work, we consider graphs made up of labeled nodes and edges. A graph contains a set of nodes and a set of relations between nodes called edges. Mathematically, a graph is represented by a 4-tuple $G = (V, E, \mu, \nu)$, where V is a set of nodes, E is a set of edges, μ is a function assigning labels to nodes and ν is a function assigning labels to edges. Classical algorithms for graph matching employ the concepts of graph and subgraph isomorphisms. A graph isomorphism is any bijection which transforms a graph G_1 into another graph G_2, and vice versa. If such a function exists, the two graphs are isomorphic; in other words, the two graphs are considered to be the same. If one of the graphs is larger than the

[1] The completion of this research was made possible thanks to Bell Canada's support through its Bell University Laboratories R & D program.

1051-4651/02 $17.00 © 2002 IEEE

other, then the matching process is performed by looking for subgraph isomorphisms. Given two graphs G_1 and G_2, finding a subgraph isomorphism involves finding a subgraph G_3 of G_2, such that G_1 and G_3 are isomorphic. Finally, in practical applications, objects and the relationships between them are often affected by noise and distortion, making it impossible to match the objects. In such cases, the concept of error-correcting is called for. Error-correcting involves finding isomorphisms between two graphs by using edit operations, such as insertion, deletion or substitution of both nodes and edges, to transform one graph into the other.

Classical error-correcting subgraph isomorphism methods [1],[2] use particular versions of the A* search procedure to find the best matching. These methods are guaranteed to find the optimal matching, but require exponential time due to the NP-completeness of the problem. A brief analysis [5] of the algorithm's complexity shows that in the best case, the algorithm needs $O(n^2 m)$ steps to find the optimal matching. However, in the worst case, $O(n^2 m^n)$ steps are needed to find the optimal matching. For this reason, algorithms of this kind are not suitable for larger graphs. Nevertheless, the error-correcting subgraph isomorphism algorithm based on A* is still among the most efficient, and is widely used by many researchers to solve the problem of graph matching. In order to reduce time and complexity, they incorporate various heuristic look-ahead techniques which are often application-dependent.

The error-correcting subgraph isomorphism algorithm based on A* [1],[2] belongs to the class of optimal algorithms; that is, it is guaranteed to find the optimal subgraph isomorphisms. The new algorithm proposed in this paper is application-independent, in that it does not require the use of heuristics. It can be used to find the optimal subgraph isomorphisms. However, its major strength resides in its capacity to find good (usually optimal) matchings in a short time. In this sense, it can be categorized in the class of approximate algorithms.

3. The new matching algorithm

In this section, we present a new algorithm for the graph-matching problem[6]. Given two graphs, the goal is to find the best matching between their nodes that leads to the smallest matching error. This smallest error between the two graphs can be viewed as the distance between them. To compute this error, we must compute the dissimilarity between each pair of matched nodes, plus the dissimilarity between (corresponding) edges. The goal is to find the mapping that results in the smallest error.

The basic idea of the new algorithm is iterative exploration of the best possible node mappings and selection of the best mapping at each iteration phase. The advantage of this algorithm is that the iterative process can often find the optimal mapping within a few iterations (for instance 5), significantly reducing the run time. In the first phase, the algorithm selects the best possible mapping(s) that minimize the matching error due to nodes only. Of these mappings, those that also give the smallest error in terms of edge matching are retained. In the second phase, the algorithm examines the mappings that contain at least one second-best mapping between nodes and then again computes those mappings that give rise to the smallest error in terms of edge matching. This process continues through a predefined number of phases.

3.1. Algorithm description

We suppose that distance measures associated with the basic graph edit operations have been defined; i.e. costs have already been associated with substitution of nodes and edges, deletion of nodes and edges, etc. The technique proposed here is inspired by both Ullman's algorithm and the error-correcting subgraph isomorphism procedure. The new algorithm is designed for substitution operations only. It can easily be extended to deal with deletion and insertion operations by considering some special cases. For example, deletion of a node can be performed by matching the node to a special (non)node. The algorithm is designed to find a graph isomorphism when both graphs have the same number of nodes and a subgraph isomorphism when one has fewer nodes than the other.

Given two graphs $G_1 = (V_1, E_1, \mu_1, \nu_1)$ and $G_2 = (V_2, E_2, \mu_2, \nu_2)$, in order to detect the most promising mappings, a $n \times m$ matrix $P = (p_{ij})$ is introduced, where n and m are the numbers of nodes in the first and the second graph, respectively. Each element p_{ij} in P denotes the dissimilarity between node i in G_1 and node j in G_2. In order to detect the most promising mapping, we use a second $n \times m$ matrix $B = (b_{ij})$. The first step is to initialize matrix P by setting $p_{ij} = d(\mu_1(v_i), \mu_2(v_j))$. The second step then consists of initializing matrix B by setting $b_{ij} = 0$. In order to detect the promising mappings, the algorithm first sets some elements of B to 1. Specifically, for each row in matrix B, the elements corresponding to the minimum elements in the same row of matrix P are set to 1, $(b_{ij} = 1)$. Then, for each possible mapping extracted from B, the algorithm computes the error generated by nodes and the error generated by edges. The mapping that gives the smallest matching error will be recorded. This is the first phase $(Current\_Phase = 1)$ of the algorithm.

In the second phase $(Current\_Phase = 2)$, the algorithm will reset some elements in each row of matrix

B to zero. These elements correspond to the second-smallest elements in each row of matrix P. The algorithm will extract those isomorphisms from matrix B that contain at least one node-to-node matching added to matrix B at this phase. Of these isomorphisms and the isomorphisms obtained in the first phase, those with the smallest cost are retained. The algorithm then proceeds to the next phase, $(Current\_Phase=3)$, and so on.

A direct implementation of the above ideas would result in redundant extraction and testing of isomorphisms, since any matching extracted from matrix B at a given time will also be extracted from any subsequent matrix B. To solve this problem, a smart procedure has been designed. First, matrix B' is introduced to keep a copy of all possible node-to-node matchings that have been considered by the algorithm so far. B is used as a 'temporary' matrix. At each phase (except the first), each of the n rows of B is examined successively. For each row i of B, all of the previous rows of B will contain all of the possible node-to-node matchings examined so far. Row i contains only the possible node-to-node matching in the present phase. Finally, all of the following rows of B will contain only the possible node-to-node matchings examined in the previous phases. Such a matrix B guarantees that the isomorphisms extracted as the algorithm progresses will never be the same and that all of the isomorphisms that need to be extracted at each phase will indeed be extracted.

A Simple Example. Two graphs G_1 and G_2 are shown in Figure 1. In this figure, the numbers inside the circles denotes the node labels and the numbers near the connecting lines denote the edge labels. The corresponding matrices P and B are shown in Table 1 and Table 2, respectively.

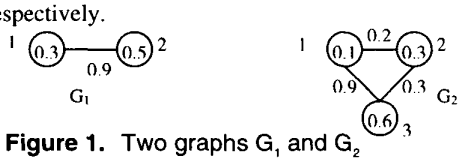

Figure 1. Two graphs G_1 and G_2

0.2	0	0.3
0.4	0.2	0.1

Table 1. Matrix P

0	1	0
0	0	1

Table 2. Matrix B / Phase 1

From the matrix in Table 2, the following matching $(1_{G1},2_{G2})$ and $(2_{G1},3_{G2})$ will be extracted and further examined.

At phase 2, matrix B' will contain

1	1	0
0	1	1

Table 3. Matrix B' Phase 2

while the extraction process uses only the following matrices

1	0	0
0	0	1

Table 4. Matrix B Step1/Phase 2

1	1	0
0	1	0

Table 5. Matrix B Step2/Phase 2

Thus there will never be redundant extraction of possible matchings. As the second matching $(1_{G1},2_{G2})$ and $(2_{G1},2_{G2})$ is not bijective, only the first matching will be considered. The Matching_Nodes process is applied only for a valid matching which has a bijective mappings.

3.2. Algorithm and Complexity

Input: two attributed graphs G_1 and G_2.
Output: matching between nodes in G_1 and G_2, from the smaller graph (e.g., G_1) to the larger (e.g., G_2)

1. Initialize P as follows:
   For each p_{ij}, set $p_{ij}=d(\mu_1(v_i),\mu_2(v_j))$.
2. Initialize B as follows:
   For each b_{ij}, $i=1,...,n$ and $j=1,...,m$, set $b_{ij}=0$.
3. While $Current\_Phase < K$
   If $Current\_Phase=1$,
   Then for all $i=1,...,n$
   select the element with the smallest value in P that is not marked 1 in B and set it to 1 in B;
   call Matching_Nodes(B).
   Else for all $i=1,...,n$
   set $B'=B$
   for all $j=1,...,m$ set $b_{ij}=0$
   select the element with the smallest value in P that is not marked 1 in B' and set it to 1 in B and B';
   call Matching_Nodes(B);
   set $B=B'$.
   If all elements in B are marked 1,
   Then set $Current\_Phase=K$
   Else add 1 to $Current\_Phase$.

Matching_Nodes(B)
For each valid mapping in B
1. Compute the matching error generated by nodes.
2. Add the error generated by the corresponding edges to the matching error.
3. Save the actual matching if the matching error is minimal.

The complexity of the algorithm depends on the number of phases K. For a given K, to find the best

matching we need $O(n^2K^n)$ steps, where n is the number of nodes in the smaller graph. Details of the deduction are given in [6].

4. Comparison with the error-correcting subgraph isomorphism algorithm

From a theoretical point of view, it is clear that the error-correcting subgraph isomorphism algorithm needs more steps to find the optimal matching than the new algorithm. The number of steps is reduced by $(m/K)^n$ when the new algorithm is performed.

In order to study the behaviour of the new algorithm in practice, we performed a number of experiments with randomly generated graphs. Comparisons were made with the error-correcting (**ec**) subgraph isomorphism algorithm based on A*[1],[2]. We generated 1000 pairs of graphs. Each graph was randomly assigned a number of nodes between 2 and 10. Each node and edge was randomly assigned a label value between 0 and 1. The new algorithm was tested with K varying from 1 to 5. The following indicators were used in the comparison: the average (accumulated) time needed by the new algorithm for each given number of phases, the number of optimal matchings found for each given number of phases and the average time needed by the error-correcting algorithm based on A* to compute the optimal matching. The experiments were performed on a SUN workstation, Ultra 60, with two 450 MHz CPUs and 512Mb internal memory.

Number of phases K	Optimal matchings reached by the proposed algorithm	Average time in seconds
1	609	2.14
2	827	3.69
3	940	6.14
4	971	11.04
5	1000	16.28
Error correcting(A*)	1000	186.57

Table 6. Comparison between the two algorithms.

Table 6 shows the performance of both the proposed algorithm and the error-correcting subgraph isomorphism algorithm based on A*. For the new algorithm, it is normal to find that the average time needed to detect the best matching increases with the number of phases. The number of optimal matchings also increases significantly with the number of phases. For this set of random graphs, the optimal matching is found for every pair of graphs when K=5. On the other hand, the error-correcting algorithm based on A* requires much more time than the new algorithm. We believe that the gain of the new algorithm in terms of execution time is due to its process of searching for the most promising candidates and to the subroutine for the extraction of node-to-node mappings (candidates).

A* is a state-space search algorithm. The number of states expanded by the algorithm grows significantly rapidly in the case of larger graphs. The algorithm is appropriate only for small graphs. In general, it cannot handle complete graphs with more than 10 nodes. In contrast, our algorithm can easily deal with graphs whose sizes exceed 10 nodes: Table 7 shows the time needed to find the best matching with K=5.

Size	8 ×12	10×15	13×18
Time in seconds	39	761	1345

Table 7. Time needed to find the best matching.

5. Conclusion

This paper has presented a new subgraph isomorphism algorithm which proposes a novel approach to the search for the best matching between two graphs. The search process is decomposed into K phases. The promising mappings in each phase are extracted and their matching errors are computed. The concept of the edit operation is used to compute the matching error. The new algorithm was compared with the error-correcting algorithm based on A*. Decomposition of the search process into phases allows it to find a high percentage of the optimal matchings in a small number of phases, considerably reducing the execution time. This algorithm is being applied to a content-based image retrieval system.

6. References

[1] W.H. Tsai and K.S. Fu. Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis. *IEEE Trans. on SMC*, vol. 9, no. 12. December 1979..

[2] A. Sanfeliu and K.S. Fu, A Distance Measure Between Attributed Relational Graphs for Pattern Recognition. *IEEE Trans. on SMC*, vol. 13, no. 3. May/June 1983.

[3] H. Bunke, Error Correcting Graph Matching: On the Influence of the Underlying Cost Function, *IEEE Trans. on PAMI*, vol. 21, no. 9. Sept. 1999.

[4] J.R. Ullman, An algorithm for subgraph isomorphism, *Journal of the ACM*, vol. 23, no. 1, January 1976, pp. 31-42.

[5] B.T.Messmer. Efficient Graph Matching Algorithms for Preprocessed Model Graphs, Thesis. University of Bern, 1996 http://citeseer.nj.nec.com/114601.html

[6] A. Hlaoui and S. Wang. Image Retrieval Systems Using Graph Matching. Rapport de Recherche, No. 275, Département de mathématiques et d'informatique, Univérsité de Sherbrooke, 2001.