Speeding up Graph Edit Distance Computation with a Bipartite Heuristic[1]

(Extended Abstract)

Kaspar Riesen, Stefan Fankhauser and Horst Bunke[2]

1 Introduction

Graph edit distance is a dissimilarity measure for arbitrarily structured and arbitrarily labeled graphs. In contrast with other approaches, it does not suffer from any restrictions and can be applied to any type of graph, including hypergraphs [1]. Graph edit distance can be used to address various graph classification problems with different methods, for instance, k-nearest-neighbor classifier (k-NN), graph embedding classifier [2], or classification with graph kernel machines [3]. The main drawback of graph edit distance is its computational complexity which is exponential in the number of nodes of the involved graphs. Consequently, computation of graph edit distance is feasible for graphs of rather small size only. In order to overcome this restriction, a number of fast but suboptimal methods have been proposed in the literature (e.g. [4]).

In the present paper we aim at speeding up the computation of exact graph edit distance. We propose to combine the standard tree search approach to graph edit distance computation with the suboptimal procedure described in [4]. The idea is to use a fast but suboptimal bipartite graph matching algorithm as a heuristic function that estimates the future costs. The overhead for computing this heuristic function is small, and easily compensated by the speed-up achieved in tree traversal. Since the heuristic function provides us with a lower bound of the future costs, it is guaranteed to return the exact graph edit distance of two given graphs.

2 Graph Edit Distance

Graph edit distance defines the dissimilarity of two graphs by the minimum amount of distortion that is needed to transform one graph into another [1]. The distortions, or edit operations e_i, considered in the present paper consist of *insertions* $(\varepsilon \rightarrow v)$, *deletions* $(u \rightarrow \varepsilon)$, and *substitutions* $(u \rightarrow v)$ of nodes and edges. A sequence of edit operations $(e_1, \ldots, e_k)$ that transform a graph g_1 into a graph g_2 is commonly referred to as *edit path* between g_1 and g_2. In order to represent the degree of modification imposed on a graph by an edit path, a cost function is introduced measuring the strength of the change caused by each edit operation. Consequently, the edit distance of graphs is defined by the minimum cost edit path between two graphs. Note that the edit operations on edges can be inferred by edit operations on their adjacent nodes, i.e. whether an edge is substituted, deleted, or inserted, depends on the edit operations performed on its adjacent nodes.

[1] This work has been supported by the Swiss National Science Foundation (200021-113198/1).

[2] All authors are with the Institute of Informatics and Applied Mathematics, University of Bern, Neubrückstrasse 10, CH-3012 Bern, Switzerland, {riesen, bunke}@iam.unibe.ch

A widely used method for edit distance computation is based on the A* algorithm [5]. This algorithm explores the space of all possible mappings between two graphs by means of an ordered tree. Such a search tree is constructed dynamically at runtime by iteratively creating successor nodes linked by edges to the currently considered node in the search tree. In order to determine the node which will be used for further expansion of the actual mapping in the next iteration, a heuristic function is usually used. Formally, for a node p in the search tree, $g(p)$ represents the cost of the partial edit path accumulated so far, and we use $h(p)$ for denoting the estimated costs from p to a leaf node representing a complete solution. The sum $g(p) + h(p)$ gives us the total cost assigned to an open node in the search tree. Obviously, the partial edit path p that minimizes $g(p) + h(p)$ is choosen next for further expansion. Given that the estimation of the future costs $h(p)$ is lower than, or equal to, the real costs, an optimal path from the root node to a leaf node is guaranteed to be found [5].

3 Bipartite Heuristic

In the simplest scenario of an A* algorithm, the estimation of the lower bound $h(p)$ of the future costs for the current node p is set to zero for all p (Plain-A*). In the other extreme $h(p)$ would return the exact future costs in exponential time complexity which is unreasonable, of course. To solve the problem of estimating a lower bound $h(p)$ for the costs from the current node p to a leaf node, one maps the unprocessed nodes and edges of graph g_1 to the unprocessed nodes and edges of graph g_2 such that the resulting costs are minimal. This mapping should be done in a faster way than the exact computation and return a good approximation of the true future cost. Note that the smaller the difference between $h(p)$ and the real future cost is, the fewer nodes will be expanded by the A* algorithm. For our heuristic function we first define a cost matrix C_n which contains the individual costs of all possible node assignments.

Definition 1 (Cost Matrix) *Let us assume that n nodes $\{u_1, \ldots, u_n\}$ of graph g_1 and m nodes $\{v_1, \ldots, v_m\}$ of graph g_2 are unprocessed so far. The cost matrix C is defined by*

$$C_n = \left[\begin{array}{ccc|ccc}
c_{1,1} & \cdots & c_{1,m} & c_{1,\varepsilon} & & \infty \\
\vdots & \ddots & \vdots & & \ddots & \\
c_{n,1} & \cdots & c_{n,m} & \infty & & c_{n,\varepsilon} \\
\hline
c_{\varepsilon,1} & & \infty & & & \\
 & \ddots & & & 0 & \\
\infty & & c_{\varepsilon,m} & & &
\end{array}\right]$$

where $c_{i,j}$ denotes the cost of a node substitution $c(u_i \rightarrow v_j)$, $c_{i,\varepsilon}$ the cost of a node deletion $c(u_i \rightarrow \varepsilon)$, and $c_{\varepsilon,j}$ the costs of a node insertion $c(\varepsilon \rightarrow v_j)$.

Note that for the unprocessed edges of both graphs a cost matrix for edge assignments C_e can be constructed analogously. On the basis of the cost matrices C_n and C_e, Munkres' algorithm [6] given in Alg. 1 can be executed seperately for nodes and edges. This algorithm finds the optimal, i.e. the minimum cost, assignment of the elements (nodes or edges) represented by the rows to the elements represented by the columns of matrix C_n or C_e in polynomial time. That is, in the worst case the maximum number of operations needed is $O((n+m)^3)$, where $(n+m)$ is the dimensionality of the cost matrix[3]. Note that Munkres' algorithm, used in its original context, is optimal for solving the assignment problem. However, it provides us with a suboptimal solution for graph edit distance only since node edit operations are considered indiviudally and no implied operations on the edges can be inferred.

Algorithm 1 Computation of the minimum future cost by Munkres' algorithm

```
Input:    A cost matrix $C$ with dimensionality $k$
Output:   The minimum-cost node or edge assignment

For each row $r$ in $C$, subtract its smallest element from every element in $r$
For all zeros $z_i$ in $C$, mark $z_i$ with a star if there is no starred zero in its row or column
STEP 1:
for Each column containing a starred zero do
    cover this column
end for
if $k$ columns are covered then GOTO DONE else GOTO STEP 2 end if
STEP 2:
if $C$ contains an uncovered zero then
    Find an arbitrary uncovered zero $Z_0$ and prime it
    if There is no starred zero in the row of $Z_0$ then
        GOTO STEP 3
    else
        Cover this row, and uncover the column containing the starred zero GOTO STEP 2.
    end if
else
    Save the smallest uncovered element $e_{min}$ GOTO STEP 4
end if
STEP 3: Construct a series $S$ of alternating primed and starred zeros as follows:
Insert $Z_0$ into $S$
while In the column of $Z_0$ exists a starred zero $Z_1$ do
    Insert $Z_1$ into $S$
    Replace $Z_0$ with the primed zero in the row of $Z_1$. Insert $Z_0$ into $S$
end while
Unstar each starred zero in $S$ and replace all primes with stars. Erase all other primes and
uncover every line in $C$ GOTO STEP 1
STEP 4: Add $e_{min}$ to every element in covered rows and subtract it from every element in
uncovered columns. GOTO STEP 2
DONE: Assignment pairs are indicated by the positions of starred zeros in the cost-matrix.
```

Summing up all values of the minimum cost node and edge assignment results in an approximation of the real future costs. Obviously, this approximation does not consider any structure preserving constraints, whereas exact edit distance computation by means of a tree search considers the structure of the graphs under consideration. Hence, the distance found by Alg. 1 is lower than, or equal to, the exact edit distance, i.e. it provides us with a lower bound of the future costs. Consequently, using Alg. 1 in order to compute the function $h(p)$ in an A* implementation results in a procedure that is guaranteed to return the optimal edit distance. In the remainder of this paper we will refer to this method as Bipartite-A*, or BP-A* for short.

4 Experimental Results

To empirically show the speed-up of graph edit distance by means of the heuristic function used in BP-A* over the standard tree search (Plain-A*), both algorithms are applied to four different real world graph sets (Letter, Image, Fingerprint, and Molecule) measuring the mean computation time and the

[3] With a brute force algorithm a $O((n+m)!)$ complexity is required.

Table 1: Average running time and average number of open paths

	Plain-A*		BP-A*		Improvements	
Database	Time [ms]	Paths	Time [ms]	Paths	Speed-Up	Reduction
Letter	465	477.6	14	72.2	33.2	6.6
Image	0.5	8.8	0.5	3.7	-	2.4
Fingerprint	6140	2465.1	374	507.1	16.4	4.9
Molecule	3799	2195.4	2	18.0	1899.5	122.0

mean number of expanded paths during the search. Note that all databases have been used in graph classification experiments before [2].

In Table 1 the average running time per edit distance computation and the mean number of expanded paths are given. It turns out that the standard tree search algorithm can be substantially sped up by means of the BP-A* heuristic. On all datasets, except for the image graphs where BP-A* is not faster than Plain-A*, the speed-up factors are reported in Table 1. The strong reduction of the average number of open paths during the tree search by means of the novel heuristic function (note the reduction factors in Table 1) explains this speed-up plausibly.

5 Conclusions

Defining a heuristic $h(p)$ is a major task in A* based graph matching algorithms. A crucial point is the trade-off between the precision of the estimation of the true future costs and its fast computation. In the present paper we propose a novel heuristic which returns a good approximation of the future cost in polynomial time complexity. With several experiments, using real world data of quite diverse nature, we demonstrate that a significant speed-up of exact graph edit distance computation by means of our heuristic can be achieved. Consequently, it is potentially useful for various machine learning and pattern recognition tasks that use graph representations.

References

[1] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. Pattern Recognition Letters, 1:245–253, 1983.

[2] K. Riesen, M. Neuhaus, and H. Bunke. Graph embedding in vector spaces by means of prototype selection. Acc. for 6th Int. Workshop on Graph-based Representation in Pattern Recognition.

[3] M. Neuhaus. Bridging the Gap Between Graph Edit Distance and Kernel Machines. PhD Thesis, University of Bern, Institute of Informatics and Applied Mathematics, 2006.

[4] K. Riesen, M. Neuhaus, and H. Bunke. Bipartite graph matching for computing the edit distance of graphs. Acc. for 6th Int. Workshop on Graph-based Representation in Pattern Recognition.

[5] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions of Systems, Science, and Cybernetics, 4(2):100–107, 1968.

[6] J. Munkres. Algorithms for the assignment and transportation problems. In Journal of the Society for Industrial and Applied Mathematics, volume 5, pages 32–38, March 1957.