Titre: Title:	A Google-inspired error-correcting graph matching algorithm
Auteurs: Authors:	Segla Kpodjedo, Philippe Galinier et Giuliano Antoniol
Date:	2008
Type:	Rapport / Report
Référence: Citation:	Kpodjedo, S., Galinier, P. & Antoniol, G. (2008). *A Google-inspired error-correcting graph matching algorithm* (Rapport technique n° EPM-RT-2008-06).

EPM–RT–2008-06

A GOOGLE-INSPIRED ERROR-CORRECTING GRAPH MATCHING ALGORITHM

Segla Kpodjedo, Philippe Galinier, Giuliano Antoniol
Département de Génie informatique et génie logiciel
École Polytechnique de Montréal

Juillet 2008

EMP-RT-2008-06

A Google-inspired error-correcting graph matching algorithm

Segla Kpodjedo, Philippe Galinier,
Giuliano Antoniol
Département de génie informatique et génie logiciel
École Polytechnique de Montréal

Juilet 2008

A Google-inspired error-correcting graph matching algorithm

Segla Kpodjedo, Philippe Galinier and Giuliano Antoniol
{segla.kpodjedo, philippe.galinier}@polymtl.ca, antoniol@ieee.org

Department of Génie Informatique
École Polytechnique de Montréal — Canada

Abstract. Graphs and graph algorithms are applied in many different areas including civil engineering, telecommunications, bio-informatics and software engineering. While exact graph matching is grounded on a consolidated theory and has well known results, approximate graph matching is still an open research subject.
This paper presents an error tolerant approximated graph matching algorithm based on tabu search using the Google-like PageRank algorithm. We report preliminary results obtained on 2 graph data benchmarks. The first one is the TC-15 database [14], a graph data base at the University of Naples, Italy. These graphs are limited to exact matching. The second one is a novel data set of large graphs generated by randomly mutating TC-15 graphs in order to evaluate the performance of our algorithm. Such a mutation approach allows us to gain insight not only about time but also about matching accuracy.

1 Introduction

Graph representations are well suited to modeling all kinds of real life objects or problems. When one represents two given objects or problems as graphs, one legitimate question is to determine how similar (quantitatively and qualitatively) those two objects are. Do they share common parts and if so, to what extent and at what level of detail? Given two graphs, an intuitive way to answer these questions is to match, with respect to some constraints, the nodes and edges of the first graph to the nodes and edges of the second graph. In many areas, the generated or observed graphs are subject to all kinds of deformations or modifications. Exact matching, which requires a strict correspondence among the two objects being matched or their subparts, often fails then to provide exploitable results. In this paper, we focus on approximate graph matching since real-life applications often fall into this category. Applications in different areas such as bioinformatics [8, 15], document processing [9] and video analysis [20] among others can be modeled as error tolerant graph matching problems.

Briefly, approximate graph matching algorithms allow matching two nodes that violate constraints such as the edge-preservation constraint - exact correspondence of edges - or any other characteristic such as node/edge labels, weights etc. Instead, a penalty is assigned to those constraint violations, depending on the specific problem and desired results. In most cases, the best matching is considered to be the one that minimizes the

overall penalty cost. The problem is known to be NP-hard, and optimal algorithms suffer from prohibitive computation times on medium and large graphs. In this paper, we propose a tabu algorithm to address the approximate graph matching problem. Given penalty costs, our algorithm will try to find the optimal matching between two graphs. A similarity measure between two nodes of two different graphs can be defined by "how closely" those nodes are likely to match in the optimal matching. In graph matching, similarity measures prevent random matching by providing the probability of any given match. Our local search procedure uses a similarity measure combining local information on nodes, such as the number of edges (incoming and outgoing), with "global" information on nodes provided by what we call "structural metrics" computed via the Google-inspired PageRank algorithm [3]. PageRank [3], one of the main components behind the first versions of Google, basically measures the relative importance of each element of a hyperlinked set and assigns it a numerical weighting. In essence, the more references (incoming arcs) an element (vertex) gets from other elements (preferably important), the more importance it deserves. PageRank is linked to random walks on Markov chains. Its main advantage is its outstanding efficiency in terms of computation speed on sparse matrices, which is very convenient as real-world graphs are often sparse.

Furthermore, the literature on error-correcting graph matching benchmarks is very limited; to overcome this we propose a mutation algorithm to generate a mutated version of a given graph. We define as a mutation path, the set of edit operations applied to the original graph to obtain the mutated version. The computed cost of this mutation path from a graph to its mutated offspring provides us with an expected cost which is useful to evaluate the performances of approximate graph matching algorithms.

In this paper, we formalize the graph matching problem as an Error-Correcting Graph Matching (ECGM) problem [4]; then we introduce a mutation mechanism to produce approximate graph matching benchmarks and propose an enhanced tabu search algorithm for ECGMs.

The rest of the paper is organized as follows. Section 2 introduces the background notions of this paper. Then, the section 3 briefly reviews some of the most important related works. The fourth section gives more insights about our algorithm. Section 5 presents the case studies and the results of our algorithm. The paper concludes with discussion and indications of future work.

2 Background Notions

In this section, we present the definitions for an error correcting graph matching problem as formulated in [4] and the basic principles of the PageRank algorithm.

2.1 Problem Statement

The following definitions, mostly adapted from [4], contextualize the error correcting graph matching in a theoretical framework.

Definition 1. Given two finite alphabets of symbols, $\sum_V$ and $\sum_E$, we define a graph as a triple (V, L_V, L_E) where V is the finite set of elements, called nodes or

vertices; $L_V : V \rightarrow \quad \sum_V$ is the node labeling function; $L_E : V \times V \rightarrow \sum_E$ is the edge labeling function.

To simplify problem formulation it is assumed graphs are fully connected; the special label *null* is assigned to *non-edges*. The set of *edges E* is then implicitly given by considering only edges with a label different from *null.* In the following, we will refer to *non-edges* as edges assigned the *null* label and *effective edges* as edges assigned with a label other than *null*

Def nition 2. Let $g_1 = (V_1, L_{V1}, L_{E1})$ and $g_2 = (V_2, L_{V2}, L_{E2})$ be two graphs. An *error-correcting graph matching (ECGM)* from g_1 to g_2 is a bijective function $f : \hat{V}_1 \rightarrow \hat{V}_2$ where $\hat{V}_1 \subseteq V_1$, $\hat{V}_2 \subseteq V_2$. We say $x \in \hat{V}_1$ is *substituted* by node $y \in \hat{V}_2$ if $f(x) = y$. Furthermore, any node from $V_1 - \hat{V}_1$ is *deleted* from g_1, and any node from $V_2 - \hat{V}_2$ is *inserted* in g_2 under f. We will use $\hat{g}_1$ and $\hat{g}_2$ to denote the subgraphs of g_1 and g_2 that are induced by the sets $\hat{V}_1$ and $\hat{V}_2$, respectively.

The mapping *f indirectly* implies edit operations on the edges of g_1 and g_2. If $f(x_1) = x_2$ and $f(y_1) = y_2$, then the edge (x_1, y_1) is substituted by edge (x_2, y_2). If a node x_1 is deleted from g_1, then any edge incident to x_1 is deleted, too. Obviously, any *ECGM* can be understood as a set of edit operations (substitutions, deletions, and insertions of both nodes and edges) that transform a given graph g_1 into another graph g_2).

Def nition 3. The *cost* of an *ECGM* $f : \hat{V}_1 \rightarrow \hat{V}_2$ from a graph $g_1 = (V_1, L_{V1}, L_{E1})$ to a graph $g_2 = (V_2, L_{V2}, L_{E2})$ is given by

$$c(f) = \sum_{x_1 \in \hat{V}_1} c_{ns}(L_{V1}(x_1), L_{V2}(f(x_1))) + \sum_{x_1 \in V_1 - \hat{V}_1} c_{nd}(L_{V1}(x_1)) + \sum_{x_2 \in V_2 - \hat{V}_2} c_{ni}(L_{V2}(x_2)) + \sum_{(x_1,y_1) \in \hat{E}_1} c_{es}(L_{E1}((x_1, y_1)), L_{E2}((f(x_1), f(y_1)))) + \sum_{(x_1,y_1) \in E_1 - \hat{E}_1} c_{ed}(L_{E1}((x_1, y_1))) + \sum_{(x_2,y_2) \in E_2 - \hat{E}_2} c_{ei}(L_{E2}((x_2, y_2))) \quad (1)$$

where $c_{ns}(L_{V1}(x_1), L_{V2}(f(x_1)))$ is the cost of substituting a node $x_1 \in \hat{V}_1$ by the label of $f(x_1) \in \hat{V}_2$; $c_{nd}(L_{V1}(x_1))$ is the cost of deleting a node $x_1 \in V_1 - \hat{V}_1$ from g_1; $c_{ni}(L_{V2}(x_2))$ is the cost of inserting a node $x_2 \in V_2 - \hat{V}_2$ in g_2; $c_{es}(L_{E1}((x_1, y_1)), L_{E2}((f(x_1), f(y_1))))$ is the cost of substituting an edge $e = (x, y) \in \hat{E}_1$ by $e' = (f(x), f(y)) \in \hat{E}_2$; $c_{ed}(L_{E1}((x_1, y_1)))$ is the cost of deleting an edge $e \in E_1 - \hat{E}_1$ from g_1 and $c_{ei}(L_{E2}((x_2, y_2)))$ is the cost of inserting an edge $e \in E_2 - \hat{E}_2$ in g_2.

The shorthand notations $E_1, \hat{E}_1, E_2, \hat{E}_2$ are used for $V_1 \times V_1, \hat{V}_1 \times \hat{V}_1, V_2 \times V_2, \hat{V}_1 \times \hat{V}_1$, respectively. Formula 1 present our cost function which is actually the sum of the cost of edit operations. As presented, those costs are def ned using nodes/edges labels. For instance, $C_{es}(l_1, l_2)$ is the cost of substituing an edge labeled l_1 by an edge labeled l_2, while $C_{ed}(l_1)$ is the cost for deletingan edge labeled l_1 from the f rst graph .

For simplif cation purposes, one can use single values for specif c edit operations such as a constant value for all node addition operations. Nevertheless, our approach can accomodate a higher level of detail if required. The term "*edge substitution*" in

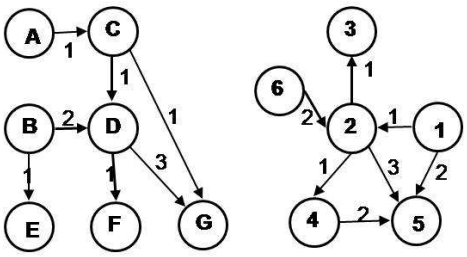

Fig. 1. Examples of graphs to be matched

the previous def nition is related to different operations one might want to distinguish. Thus, we identify

- c_{esi} Identical edge label substitution
- c_{ess} Structural error : edge substitution involving an effective edge and a non-edge
- c_{esl} Label error : edge substitution involving two effective edges with different labels

Let c_{ns} be the cost for substituing any node assigned a label l_1 by another assigned l_2, with $l_1 \neq l_2$; c_{nd} the cost of deleting any node from g_1; c_{ni} the cost of inserting any node in g_1; c_{ed} the cost of deleting any effective edge from g_1; c_{ei} the cost of inserting any effective edge in g_2. Any ECGM cost function could then be represented by the octuple (c_{ns}, c_{nd}, c_{ni}, c_{ed}, c_{ei}, c_{esi}, c_{ess}, c_{esl}). The cost of an identical edge label substitution should be *0*. Furthermore, if symmetry is wanted or irrelevant, one can use the same value for deletions/insertions. From the initial octuple, one can keep the quintuple (c_{ns}, c_{no}, c_{eo}, c_{ess}, c_{esl}) to def ne any ECGM cost function; c_{no} (respectively c_{eo}) being the cost of adding/removing any node (respectively any edge). For instance, (c_{ns}, c_{no}, c_{eo}, c_{ess}, c_{esl}) = (∞, 1, 0, ∞, ∞) corresponds to the maximum common subgraph problem. The values of those costs could be related to the probability of occurrence of the associated distortions. Therefore, one may want the cost of a structural error to be inferior to that of a label error, if changing the type of relation between two nodes is less likely than simply dropping / losing that relation.

Example: Given the cost function (c_{ns}, c_{no}, c_{eo}, c_{ess}, c_{esl}) = $(0, 2, 1, 12, 7)$ and the two labeled graphs on Figure 1, one may be interested in f nding the optimal matching. Notice here that we assumed a cost of modifying an edge with label modif cation (i.e., C_{esl}) substantially higher than the cost of inserting/deleting a node (i.e., C_{no}).

2.2 PageRank

PageRank allow us to assign a metric representative of global structure for each vertex of a given graph. This global metric is the outcome of a Random Walk (RW) on the graph which is a probabilistic model used to compute the probability $u_v(t)$ of being located in a given vertex v at time t. A random walker on a graph proceeds iteratively by moving from a vertex v_1 to a vertex v_2 following the arc (v_1, v_2) or by jumping directly

(with respect to a f xed low probability) to v_2. The probability distribution on all the vertices of a given graph G = (V,E) is represented by a vector $u(t) = [u_1(t); ...; u_{|V|}(t)]$. The vector of probabilities is updated at each step and is proven to converge to a stable solution. Further details can be found in [19] which proposes a very eff cient algorithm to compute $u(t)$. The point here is the use of an eff cient algorithm providing a global metric for each vertex of a graph. For instance, given a *jump probability* of *0.1*, we obtain $u(t) = [0.014; 0.014; 0.14; 0.16; 0.057; 0.046; 0.175]$, for the f rst graph on Figure 1. In that graph, the nodes G, D, C have the highest values.

3 Related Works

Conte et al. [10] in their review of graph matching algorithms classify the existing methods into three main categories: techniques based on tree search, techniques based on continuous optimization, spectral methods. This classif cation is completed by "other techniques" such as genetic algorithms etc.

In techniques based on tree search, the search - with backtracking - is directed by the cost of the partial solution obtained so far and various heuristics are used to prune paths which are estimated as unfruitful or, on the contrary, prioritize the most promising paths. The range and power of prediction of those proposed heuristics [1] are essential for reasonable computation times. Some authors investigated approaches aimed at redef ning or simplifying the graph matching problem. For example, decomposition techniques are presented in [13], while transformation models are the focus in [11].

Some tree-search based techniques lead to the def nition of optimal algorithms [12]; however, the major drawback of this category of techniques is the often prohibitive computation times required when the size of the graphs increases.

The literature is rich in fast and eff cient, if not optimal, algorithms designed to resolve continuous optimization problems. This explains the popularity of this family of techniques, even if it means, as in this case, casting an inherently discrete problem into a continuous, non linear problem, solving it with a continuous optimization technique, and eventually converting the - often non-optimal - obtained solution back into the initial discrete problem. Different methods are available from "simple" probabilistic relaxation framework [16] to the def nition of a *Bayesian graph edit distance* [18], or reformulation as *Weighted Graph Matching* [2, 26].

Considering two isomorphic graphs, their node-to-node adjacency matrices will have the same eigenvalues and eigenvectors. The converse may not be true and information gained may be only structural, thus lacking all kinds of other useful information about a graph, but this is still an interesting starting point for a graph matching problem and, among some others, Umeyama [24] pioneered the above idea. Of course, as far as approximate matching is concerned, the obtained results will gain in accuracy as the considered graphs are nearly isomorphic. In such cases, spectral techniques [6] robust to the distortions can be successfully applied. In the literature, spectral features are combined with other methods like continuous optimization techniques [17], clustering techniques [7, 6, 21] or are simply used to guide a greedy search procedure [22].

Closer to the algorithm presented in this work is [21], in which random [16]walks provide topological features further used with clustering techniques.

Among the ”other techniques” previously mentioned, we can underline the use of meta-heuristics such as genetic algorithms [5, 23], simulated annealing [13] and tabu search [25].

Overall, our review of the literature supports the opinion that most algorithms and results are application-driven; furthermore ECGM benchmarks are not publicly available. This prevents simple and clear comparisons between the different approaches and algorithms. Finally, most often, the graphs considered are usually quite small (less than 100 nodes).

4 The Algorithm

In the following, we consider an ECGM problem instance def ned by a triple (G_1, G_2, C) with G_1 being the f rst graph, G_2 the second graph and C the cost parameters as def ned in Section II. The choice of the cost parameters steers the type of matching found.

4.1 Overview of the tabu search

Our algorithm is a Tabu Search (TS) algorithm guided by global information on the nodes from the PageRank algorithm and local node features such as the number of edges. In the following paragraph, we summarize the key ideas behind a tabu algorithm.

Given a function *f* (*cost function*) to be minimized (or maximized) over some set `S` (the *Search Space*), a local search technique starts from some initial feasible point (*solution*) in the search space and proceeds iteratively (*moves*) from one point in *S* to another (a *neighbor*) until some termination criterion is met. There is no guarantee of obtaining an optimal solution as the search may get trapped in local optima, but some techniques are proven very helpful in avoiding local optima and f nding good solutions. For instance, to prevent cycles in the search, TS introduces one or several tabu lists (*short term memory*) used to exclude moves which would tend to make the search process go back to a previously visited solution. Other lists for intermediate and long-term memory may be used to intensify the search in a promising area of the search space or diversify the search to previously unexplored areas.

Given two graphs $G_1 = (V_1, L_{V1}, L_{E1})$ and $G_2 = (V_2, L_{V2}, L_{E2})$, we def ne as a *match* any pair $(n_{1i}, n_{2j}) \in (V_1 \times V_2)$. We def ne a solution S as a subset of $V_1 \times V_2$: $S \subset V_1 \times V_2$ and $(a, b), (c, d) \in S, (a, b) \neq (c, d) \Rightarrow a \neq c \; and \; b \neq d$.

Since we are interested in one-to-one matching, a legal solution excludes multiple matches, i.e. if S contains the pair (n_{1i}, n_{2j}), n_{1i} and n_{2j} cannot be in any other match of S. The search space is the set of all *legal* solutions. Given a current solution, the only moves permitted are *match insertion* + and *match removal* - i.e. one can only insert a match of nodes missing from any match of the current solution. Considering, as described in Section II, that each solution S def nes $\hat{V}_1$, the set of matched nodes of V_1 and $\hat{V}_2$, the set of matched nodes of V_2, the neighborhood $N(S)$ of a solution S is def ned by the following moves

$$S \begin{array}{ll} \nearrow & +(n_{1i}, n_{2j}) \in ((V_1 - \hat{V}_1) \times (V_2 - \hat{V}_2)) \searrow \\ & or \qquad\qquad\qquad\qquad\qquad\qquad\quad N(S) \\ \searrow & -(n_1, n_2) \in S \qquad\qquad\qquad\qquad\quad \nearrow \end{array}$$

We use two tabu lists in our algorithm : one for a match just inserted ($TABU_{OUT}$) and one for a match just deleted ($TABU_{IN}$). The $TABU_{OUT}$ list prohibits for a certain period the removal of a just inserted match, while the $TABU_{IN}$ list prevents a just deleted match to be reinserted before a certain number of moves.

4.2 Similarity Measure and moves

Given a solution *S* and a move *m*: $S \rightarrow S'$, we def ne delta cost $\delta_m = f(S') - f(S)$ as the differential of cost brought by the move *m*.

Generally, in a local search algorithm, given a solution S, the choice of a *move* m is essentially guided by its δ_m. However, in our problem, a choice based solely on this local information, on the search space, may lead to poor performance, both in terms of computation and matching accuracy.

At the beginning of the search, when the f rst matching is chosen, there is typically a very large number of possible choices with the same best possible performance. Therefore, erroneous choices are very likely to occur at the beginning of the search. and they may compromise the f nal solution. Our empirical results conf rm that assumption, as very poor results were obtained.

To make a search eff cient, our idea is to use available information for each of the two graphs. Local features of a node, such as its degree, are natural candidates for the implementation of this idea. In our algorithm, local information of a vertex v is a feature vector $loc(v) = (v_{in}, v_{out})$ with v_{in} being the number of incoming edges and v_{out} the number of outgoing edges. This is a f rst valuable information one can use to weight the match of two nodes $v_1 \in V_1$ and $v_2 \in V_2$. Given a node n_1 of a graph G_1 and a node n_2 of a graph G_2, their respective number of incoming edges e_{i1}, e_{i2} and number of outgoing edges e_{o1}, $e_{o2,}$ the local similarity metric $l(n_{1,}n_2)$ is computed as follows

$$l(n_{1,}n_2) = 1 - \frac{\sqrt{(e_{i,1} - e_{i,2})^2 + (e_{o,1} - e_{o,2})^2}}{\max(\sqrt{e_{i,1}^2 + e_{o,1^2}}, \sqrt{e_{i,2}^2 + e_{o,2}^2})} \in [0, 1]$$

We then resort to global information about a node and select a very fast algorithm : the PageRank algorithm [19]. Note that we generate the metrics on nodes without taking into account the edge labels. Thus, the obtained metrics are purely structural. The nodes are sorted in a descending order of that metric. Given a node n_1 of a graph G_1 and a node n_2 of a graph G_2, their respective ranks *rank(*n_1*)* and *rank(*n_2*)*, the global similarity metric $g(n_{1,}n_2)$ is computed as follows

$$g(n_{1,}n_2) = 1 - \left| \frac{rank(n_1)}{|V_1|} - \frac{rank(n_2)}{|V_2|} \right| \in [0, 1]$$

Note that the global similarity metric of a given node may change dramatically due to the insertion or removal of a single incoming edge. The local similarity metric will be particularly useful in those extreme situations. For each possible match $m = (n_{1i}, n_{2j})$, with $n_{1i} \in V_1$ and $n_{2j} \in V_2$, we compute as follows a weight w_{ij} designed to be a measure of the likelihood of this match. In order to combine the two metrics (local and global), we weight them with coeff cients. We assume that the higher the degree of a

node, the fewer are its possible *matches* in the other graph; this makes the similarity measure more trustable as high values of similarity will be observed for a restricted number of *matches*. Similarly, the higher the rank of a node, the more relevant its global similarity values; coeff cients are def ned as follows:

$$g_{coeff}(n_1,n_2) = 1 - \frac{1}{2} \times (\frac{rank(n_1)}{|V_1|} + \frac{rank(n_2)}{|V_2|}) \in [0,1]$$
$$l_{coeff}(n_1,n_2) = \frac{1}{2} \times (\frac{deg(n_1)}{deg_{max}(V_1)} + \frac{deg(n_1)}{deg_{max}(V_1)}) \in [0,1]$$

$deg_{max}(V_1)$ (resp. $deg_{max}(V_2)$) is the highest degree of a node found in V_1 (resp. V_2). $g_{coeff}(n_1,n_2)$ and $l_{coeff}(n_1,n_2)$ are the respective weights of the global and local similarity measures in the computation of the f nal similarity measure:

$$w_{ij} = 1 - \frac{l_{coeff}(n_1 n_2) \times l(n_1 n_2) + g_{coeff}(n_1 n_2) \times g(n_1,n_2)}{2} \in [0,1]$$

Given a move m_v = *operation*$((n_1,n_2))$, with $n_1 \in V_1$ and $n_2 \in V_2$ and its cost δ_m, we recompute the delta brought by a movement as follows. If m_v is a *match insertion*, we grant m_v a bonus to encourage the insertion of the match (n_1,n_2):

$$\delta_{m_v} = \delta_m - (w_{ij} \times MAX_{incentives})$$

Else, i.e. m_v is a *match removal*, we grant m_v a malus to discourage the removal of (n_1,n_2).

$$\delta_{m_v} = \delta_m + (w_{ij} \times MAX_{incentives})$$

$MAX_{incentives}$ represents the maximum of incentives given to a match. The weights w_{ij} are def nitively computed at the initialization of the algorithm. They are used to bias the δ_m of the moves.

Application to the example Given the cost function (c_{ns}, c_{no} c_{eo}, c_{ess}, c_{esl}) = $(0,2,1,12,7)$, we want to f nd the best matching for the two graphs in Figure 1. Initially, we have an empty solution, no matching, whose cost is 40. In this empty solution, we have to pay for all the nodes and edges deleted (7*2 + 7*1 = 21), and At Step 0, in our example, all moves have the same delta cost $\delta_m = c_{ns} - (c_{ni} + c_{nd}) = -4$. When we apply the bonus of the computed similarity metrics, we obtain, with $MAX_{incentives} = 4$, Table 4.2. (D,2) having the least cost is selected, and the δ_m are recomputed, if required. In this way, our similarity measure guides our local search. The process is iterated and, for that example, the best solution is S = {(D,2) (G,5) (C,1) (F,3) (B,6)} with $cost(S) = 17$ as shown in Figure 2.

5 Case Studies

To the best of our knowledge, only a few works have addressed the ECGM problem and no reference or standard database of large graphs for such problem exists. The existing literature does not report or mention benchmarks for ECGM algorithms. Most of the past contributions are application-driven and results on presented algorithms are

	1	*2*	*3*	*4*	*5*	*6*
A	-4	-4	-4	-4	-4	-5
B	-5	-5	-4	-4	-4	-4
C	-5	-6	-5	-5	-5	-4
D	-4	**-7**	-5	-5	-5	-4
E	-4	-5	-5	-5	-5	-4
F	-4	-4	-5	-5	-4	-4
G	-4	-5	-5	-5	-6	-4

Table 1. Modified δ_m of the moves at step 0

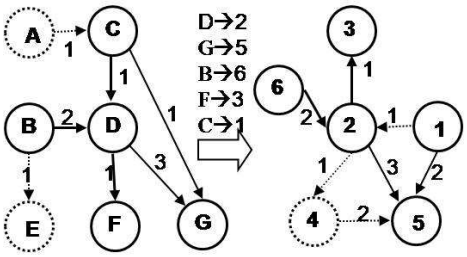

Fig. 2. Example of approximated graph matching.

reported on their own specific datasets. Moreover, the graphs are rarely publicly available. To overcome such a limitation we apply a twofold strategy. First, given that no specific dataset was found, we first applied our algorithm to a subsets of graph taken from the TC-15 Graph Database [14]. Then we applied a mutation strategy outlined in the following mutation subsection to generate a set of graph pairs with known distortion. In essence, by deleting, adding or modifying graph elements we generate graph pairs where a *near optimal* matching is known.

5.1 The Graph Database

The Graph Database of [14], also called TC-15 Database, is a large benchmark for exact graph matching problems such as graph isomorphism, subgraph isomorphism and maximum common subgraph. TC-15 contains three different kind of graphs: randomly connected, bound valence and mesh. We selected the first two categories as most relevant for our problem. In these two categories (i.e., randomly connected and bound valence graphs) TC-15 contains several subcategories. For each available subcategory we selected a large (about 1000 nodes) and a medium (about 100 nodes) graph. In particular we selected the categories of randomly connected graphs (*iso_r****) and modified bound valence graphs (*iso_b**m*). For each of their subcategories, we chose the first graphs (*.A00*) of order 100 and 1000. Table 2 presents our selection. In essence, we selected in TC-15 the subsets of large graphs likely to represent most encountered matching problems.

#	TC-15		Labeled graphs
#01	iso_r001_s100.A00		
#02	iso_r001_m1000.A00	#13	iso_r001_m1000.L00
#03	iso_r005_s100.A00		
#04	iso_r005_m1000.A00	#14	iso_r005_m1000.L00
#05	iso_r01_s100.A00		
#06	iso_r01_m1000.A00	#15	iso_r01_m1000.L00
#07	iso_b03m_s100.A00		
#08	iso_b03m_m1000.A00	#16	iso_b03m_m1000.L00
#09	iso_b06m_s100.A00		
#10	iso_b06m_m1000.A00	#17	iso_b06m_m1000.L00
#11	iso_b06m_s100.A00		
#12	iso_b06m_m1000.A00	#18	iso_b06m_m1000.L00

Table 2. Selected graphs from TC-15 and labeled graphs.

It is worth mentioning that all these graphs are unlabeled, while our algorithm is able to deal with labeled graphs. We annotated, with labels, edges of TC-15 graphs to obtain labeled graphs. Labels were drawn randomly from uniform distribution with labels in an alphabet of 4 letters $L_E = \{1, 2, 3, 4\}$. The graphs obtained are presented in Table 2.

5.2 Mutation

To obtain a set of ECGM specif c problems with controlled distortion we applied mutation to transform a given graph G_1 into another graph G_2.

The advantage is substantial since we keep track of the edit operations our mutation algorithm performs; these operations correspond to a path from G_1 to G_2. That means, knowing which node has been deleted, inserted, substituted from G_1 to G_2, we have, de facto, for our two graphs a very good known matching which we call $MM(G_1, G_2)$ for Mutation Matching with a known Exact Mutation Matching Cost (EMMC).

In essence, since G_2 is the result of a mutation algorithm we know the exact transformation applied and thus we know a nearly ideal matching and the expected cost against which results obtained by ECGM algorithms can be compared. EMMC is then a very good upper bound for our matching problem, especially when the original graph is lowly distorted. For highly distorted graph pairs where the original graph structure is severely disrupted, we cannot guarantee that there does not exist a matching cost lower than EMMC. This is not a surprise since as we introduce more and more edit operations, it is more likely that good alternatives to $MM(G_1, G_2)$ can be found. Overall, the lower the distortion and the noise introduced, the closer the EMMC value will be to the optimal cost. A theoretical study of relation between distributions of edit operations and relations with upper and lower bounds are beyond the scope of this paper and will be the subject of future works.

In this work we consider simple distortions such as : node/edge deletions, node/edge insertions and node/edge substitutions. The edit operations were assigned probabilities of occurrence and applied based on algorithms described below.

The mutation algorithm has a two steps beginning with what we called *node noise,* followed by *edge noise*. Note that all the edges inserted or modif ed are assigned labels with respect to the edge label distribution in G_1. Our mutation algorithm takes three

parameters, p_{nd} (probability of node deletion), p_{na} (probability of node addition) and p_{edge} (noise on edges). First it performs a shuffle of nodes with labels randomly assigned, then nodes are deleted, and in the subsequent steps, nodes are added before we finally apply noise on edges.

The node noise is applied using p_{nd} and p_{na}. More precisely, the node deletion algorithm parses all the nodes of the graphs and decides, according to p_{nd}, to delete or not the considered node. Similarly, the node addition algorithm parses all the nodes of the graphs and decides, according to p_{na}, to add or not a new node. When a node is deleted, its edges are deleted as well. After all nodes have been added, edges are added in an attempt to preserve the graph density. Given the number of new nodes $nodes_{new}$ and the density d_o of the original graph, we compute the number $edges_{new}$ of edges which should be linked to the set of new nodes $edges_{new} = nodes_{new} \times d_o = insertion_{trials}$. For $insertion_{trials}$ iterations, we randomly choose two nodes of the mutated graph and effectively insert an edge if at least one of the chosen nodes is a new one. When an edge is inserted, we assign it a label with respect to the edge label distribution in the original graph.

Once the node deletion and addition phases are completed, we apply the edge noise algorithm with the parameter p_{edge} which represents the probability of modifying an effective edge. An edge substitution can result in a simple edge label substitution or an edge deletion. After the effective edge substitution stops, we proceed to edge insertions. In order to preserve the overall density of the graph, we recompute the probability to insert a new edge as follows : $p_{ei} = \frac{deleted_{edges}}{m_{null}}$, with $deleted_{edges}$ the number of deleted *effective edges* in the previous step and m_{null} the number of *non-edges* (i.e., null edges) in the mutated graph. For each non-edge, an effective edge is inserted or not according to p_{ei}. Whenever we modify/insert/delete an edge we do it by using a biased wheel where the weight of each label is its percentage of occurences in the original graph.

Applying the above described graph mutation strategy, we generate a mutated graph G_2 from G_1 with a known set of applied transformation. The EMMC value is then computed by applying Formula 1 (Definition 3).

5.3 ECGM results

We apply our mutation algorithm to the selected graphs in the TC-15 dataset with different parameters. Considering the triple (p_{nd}, p_{na}, p_{edges}) it is possible to generate graph tailored for different kinds of graph matching problems.

1. $(0, 0, 0)$: graph isomorphism
2. $(0, p_{na}, 0)$: subgraph isomorphism
3. $(p_{nd}, p_{na}, 0)$: maximum common subgraph
4. $(p_{nd}, p_{na}, p_{edges})$: ECGM

Provided that the values of the parameters are kept small, the first three categories of mutation configurations can be used to obtain good approximations for optima values in corresponding exact matching problems while the last category is specific to ECGMs.

In this first set of experiments, we empirically selected the following weights for our cost function (c_{ns}, c_{no}, c_{eo}, c_{ess}, c_{esl}) = $(0, 2, 1, 12, 7)$. We applied our tabu algorithm to each of the pairs $(G, G_{mutated})$ with different noise parameters; we empirically fixed the length of the tabu lists as follows : $TABU_{IN}(10)$ and $TABU_{OUT}(20)$; the value $MAX_{incentives}$ is set to the highest edit cost : $c_{esl} = 12$; each experiment was replicated ten times. Results collected and summarized in Tables 3 and 4.

Tables are organized as follows. The *Mutation* column shows the three probability values, as percentages, applied to mutate the graph identified in column one. Thus the sequence "5_5_1" mean 5 % probability of deleting or adding a node and 1 % probability of adding noise to one edge. For each mutated graph, ECGM algorithm was run ten times and data was collected; columns *min, max* and *mean cost* report respectively the minimum, maximum and average cost in the ten trials. We indicate how far the best result of the algorithm is from the EMMC in the brackets of column *min Cost*, reporting this distance in terms of percentage. To obtain evidence of the dispersion of the matching cost, we computed the cost's standard deviation as reported in the fifth column *Std Cost*. The *Matched nodes* column is one of the key figures to judge the algorithm performance in that it represents the percentage of correctly matched nodes with respect to the $MM(G_1, G_2)$ transformation. The average time in the ten runs required to obtain a match is shown in column "Mean Time".

ECGM algorithms find optimal or near optimal solutions by minimizing some cost functions. Column *PEMMC* (percentage of EMMC) reports the percentage of times in the ten trials in which the exact value EMMC was attained. A naive approach would be to discard all nodes from the first graph and add nodes and edges to an empty graph so that the second graph is obtained; the cost of such a solution is reported in the penultimate column labeled *empty solution*. Finally the last column contains the *EMMC value*.

Results reported in the tables refer to different possible configurations: *unlabeled* versus *labeled* graphs. We report results for mutated unlabeled graphs in Table 3 and mutated labeled graphs in Table 4.

We report the percentage of nodes matched to their distorted versions (column *Matched Nodes*) and the number of times we reached the upper bound cost (*PEMMC*) as relevant accuracy indices.

On unlabeled graphs, we have an average of about 68 % of correctly matched nodes (see Table 3) attaining about 48 % of the time the value of EMMC. Labels on edges help the algorithm as shown in Table 4, indeed, the authors believe that a value of 76 % of correct matching with an average PEMMC of 36 % can be considered very good. In fact, more than two thirds of the distorted nodes are correctly retrieved. Also, those results are just an average of ten trials and a detailed analysis of Tables 3 and 4 showed that within the ten runs, our algorithm failed to reach the upper bound for only 7 of the 24 ECGM unlabeled datasets and 3 of the 10 ECGM labeled datasets. We believe that although the average PEMMC is not very high, we can find good perspectives in the high percentage of "correctly" matched nodes. This proves we generally hit the good search areas despite the empiric values of our cost function. Notice that different penalty costs may provide better or worse results for this performance indice. As shown in Table 3, some graphs like iso_r01_s100.A00 mutated with parameters (5_5_1) or iso_b03m_m1000.A00 mutated with (10_10_2) gave very poor results. This can be

explained by considering the graph density. We have found that given a graph with low density, adding and deleting edges can easily result in a drastic and deep structure change. For instance, graphs low both in density and in order can mutate in various sets of disconnected graphs with various f oating islands. These are isolated mini-graphs which are very hard to map back into the original structure. Computation time is generally low and acceptable for applications where the algorithm is run off-line, which is what we are considering, .

Dataset	Min Cost	Max Cost	Mean Cost	Std Cost	Matched Nodes%	Mean Time (s)	PEMMC	Empty solution	EMMC
iso_r001_s100.A00	262 (+274%)	378	332	47.50	18.2	0.06	0	682	70
iso_r001_m1000.A00	4523 (-0.1%)	22743	8369.40	7190.04	74.80	11.65s	60	23795	4527
iso_r005_s100.A00	189 (-5%)	189	189	0	100	0.08	100	1449	199
iso_r005_m1000.A00	19414 (0%)	104142	70290	41450.30	48.40	146.29	60	104572	19414
iso_r01_s100.A00	507 (0%)	507	507	0	100	0.13	100	2335	507
iso_r01_m1000.A00	35098 (0%)	35098	35098	0	100	609.67	100	205728	35098
iso_b03m_s100.A00	71 (-8%)	155	114.6	29.59	81.2	0.05	40	701	77
iso_b03m_m1000.A00	2338 (+151%)	3362	2655.6	379.43	48.8	12.01	0	6978	932
iso_b06m_s100.A00	189 (-1%)	343	225	59.85	86.6	0.05	70	991	191
iso_b06m_m1000.A00	1454 (+0.4%)	2948	2296.8	521.74	87.3	8.81	0	9978	1448
iso_b09m_s100.A00	113 (0%)	203	131	36	98.2	0.06	80	1349	113
iso_b09m_m1000.A00	2104 (-0.1%)	2560	2199.6	180.24	96.6	7.95s	90	12948	2108
iso_r001_s100.A00	333 (+92%)	357	345.4	8.14	8.4	0.05	0	653	173
iso_r001_m1000.A00	7299 (-0.2%)	23283	13697.4	7813.56	54.6	14.43	10	24197	7315
iso_r005_s100.A00	1001 (+168%)	1197	1137.8	70.43	12.4	0.1	0	1331	373
iso_r005_m1000.A00	36119 (0%)	102851	62810.2	32689.9	53.6	131.86	60	103279	36119
iso_r01_s100.A00	609 (0%)	2235	934.2	650.4	73.6	0.15	80	2365	609
iso_r01_m1000.A00	70667 (0%)	70667	70667	0	100	649.37	100	205689	70667
iso_b03m_s100.A00	201 (+51%)	305	267	48.13	50.4	0.05	0	713	133
iso_b03m_m1000.A00	3191 (+101%)	3633	3478.6	155.123	16.9	13.51	0	6921	1591
iso_b06m_s100.A00	209 (-2%)	209	209	0	88.4	0.06	100	977	215
iso_b06m_m1000.A00	2715 (0%)	3345	3041	243.95	83.1	9.61	20	10007	2715
iso_b09m_s100.A00	309 (0%)	327	316.2	8.82	83.8	0.06	60	1253	309
iso_b09m_m1000.A00	3419 (-2%)	11489	5823	2887.47	69.2	10.37	20	13065	3475
iso_*_*.A00	-	-	**68.10**	-	**47.92**	-	-	-	-

Table 3. Detailed results of selected graphs of TC-15 (ECGM) - Upper part mutation parameters 5_5_1; lower part 10_10_2.

6 Conclusion

We have presented an algorithm inspired by Google PageRank to f nd optimal or near optimal solutions to the class of problems of approximate, error tolerant graph matching where it is acceptable to match two nodes that violate constraints such as the edge-preservation constraint or any other characteristic such as node/edge labels or weights.

Inspired by the work of previous authors [3, 4, 21] we represented the approximate graph matching problem as an optimization problem modeled by cost matrices accounting for the cost of various edit operations (e.g., node insertion and deletion). Our algorithm relies on a tabu search; our search procedure combines local information on nodes (e.g., the number of incoming edges) with “global” information on nodes provided by “structural metrics” computed via the PageRank algorithm [3].

Dataset	Min Cost	Max Cost	Mean Cost	Std Cost	Matched Nodes%	Mean Time (s)	PEMMC	Empty solution	EMMC
iso_r001_m1000.L00	4330 (0%)	4354	4346	7.15	97.4	9.1628	20	24127	4330
iso_r005_m1000.L00	20841 (0%)	102948	53740.4	40173.3	66.2	128.29	60	103216	20841
iso_b03m_m1000.L00	1340 (+20%)	2068	1664	245.78	77.6	10.57	0	6972	1118
iso_b06m_m1000.L00	1279 (-2%)	7162	3392.4	2521.44	74.4	10.86	20	10015	1307
iso_b09m_m1000.L00	2541 (+29%)	3407	3094.6	331.137	92.5	8.35	0	13157	1975
iso_r001_m1000.L00	7742 (-0.6%)	23353	10871	6241	82.8	12.09	80	24271	7792
iso_r005_m1000.L00	36034 (0%)	101869	75528.2	32246.9	42.6	138.95	40	102188	36034
iso_b03m_m1000.L00	2133 (+48%)	2559	2348.8	150.85	67.2	11.55	0	7091	1443
iso_b06m_m1000.L00	2358 (-6%)	2496	2438.4	64.91	86.6	8.63	80	9998	2522
iso_b09m_m1000.L00	3822 (-4%)	11515	5701	2973.69	68.4	10.24	60	12850	3966
iso_*_*.A00	-	-	-	-	**75.57**	-	**36**	-	-

Table 4. Detailed results of labeled graphs (ECGM) - Upper part mutation parameters 5_5_1; lower part 10_10_2

To quantify performance of our algorithm we selected a subset of medium and large graphs from the TC-15 benchmark, which is designed for exact graph matching problems, and generated mutated graph versions with different level of distortions. These graphs are available for downloading at the SOftware Cost-effective Change and Evolution Research (SOCCER) laboratory[1].

We report accuracy and performance of our algorithm on the above mentioned set as well on the original TC-15 dataset. On an ECGM problem and labeled graphs, on average, we attain an average about 76 % of accurate node matching for graphs of about 1000 nodes. Computation time is in the order of a couple of minutes which we believe is acceptable for the off-line type of applications we foresee for our algorithm.

Future work will be devoted to assessing algorithm accuracy for larger graphs (e.g., 8,000 nodes and 80,000 edged). Also we believe it is important to better investigate more thoroughly the impact of different cost schema in the cost function as well as the inf uence of various kinds of distortions on algorithm performance.

References

1. M. Y. A. K. C. Wong and S. C. Chan. An algorithm for graph optimal monomorphism. *IEEE Trans. Syst. Man Cybern.*, 20:628–638, 1990.
2. H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(5):522–525, 1993.
3. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.
4. H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recogn. Lett.*, 18(9):689–694, 1997.
5. J. T. H. C. W. Liu, K. C. Fan and Y. K. Wang. Solving weighted graph matching problem by modif ed microgenetic algorithm. In *IEEE Int. Conf. Syst. Man Cybern.*, pages 638–643, 1995.
6. T. Caelli and S. Kosinov. An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):515–519, 2004.

[1] http://web.soccerlab.polymtl.ca/

7. M. Carcassoni and E. R. Hancock. Weighted graph-matching using modal clusters. In *CAIP '01: Proceedings of the 9th International Conference on Computer Analysis of Images and Patterns*, pages 142–151, London, UK, 2001. Springer-Verlag.
8. Cheng, Saigo, and Baldi. Large-scale prediction of disulphide bridges using kernel methods, two-dimensional recursive neural networks, and weighted graph matching. In *Proteins: Structure, Function, and Bioinformatics*, volume 62, pages 617 – 629, 2006.
9. D. Conte, P. Foggia, , C. Sansone, and M. Vento. Graph matching applications in pattern recognition and image processing. In *ICIP03*, pages II: 21–24, 2003.
10. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artif cial Intelligence*, 18:265–294, 2004.
11. L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An eff cient algorithm for the inexact matching of arg graphs using a contextual transformational model. In *ICPR '96: Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276*, pages 180–184, Washington, DC, USA, 1996. IEEE Computer Society.
12. A. C. M. Dumay, R. J. van der Geest, J. J. Gerbrands, E. Jansen, and J. H. C. Reiber. Consistent inexact graph matching applied to labeling coronary segments in arteriograms. In *Proc. Int. Conf. Pattern Recognition, Conf. C (1992)*, pages 439–442, 1992.
13. A. A. Eshera and K. S. Fu. A similarity measure between attributed relational graphs for image analysis. In *Proc. 7th Int. Conf. Pattern Recognition*, pages 75–77, 1984.
14. P. Foggia, C. Sansone, and M. Vento. A database of graphs for isomorphism and subgraph isomorphism benchmarking. In *Proc.Third IAPR TC-15 Intl Workshop Graph-Based Representations in Pattern Recognition*, pages 176–187, 2001.
15. I. Jonassen. Eff cient discovery of conserved patterns using a pattern graph. 13(5):509–522, 1997.
16. J. Kittler and E. R. Hancock. Combining evidence in probabilistic relaxation. *Int. J. of Patt. Recogn. Artif. Intell*, 3:29–51, 1989.
17. B. Luo and E. Hancock. Structural graph matching using the em algorithm and singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(10):1120–1136, 2001.
18. R. Myers, R. C. Wilson, and E. R. Hancock. Bayesian graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(6):628–635, 2000.
19. L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
20. M. Salotti. Topographic graph matching for shift estimation. In *Proceedings of the 3rd Conference on Graph Based Representations in computer vision (GBR 2001)*, pages 54–63, 2001.
21. L. Sarti. Exact and approximate graph matching using random walks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(7):1100–1111, 2005.
22. A. Shokoufandeh and S. J. Dickinson. A unif ed framework for indexing and matching hierarchical shape structures. In *IWVF-4: Proceedings of the 4th International Workshop on Visual Form*, pages 67–84, London, UK, 2001. Springer-Verlag.
23. M. D. Th. Brecke. Memetic algorithms for inexact graph matching. In *CEC: IEEE Congress on Evolutionary Computation,*, 2007.
24. S. Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10(5):695–703, 1988.
25. M. L. Williams, R. C. Wilson, and E. R. Hancock. Deterministic search stragtegies for relational graph matching. In *EMMCVPR '97: Proceedings of the First International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 261–275, London, UK, 1997. Springer-Verlag.
26. M. M. Zavlanos and G. J. Pappas. A dynamical systems approach to weighted graph matching. In *45th IEEE Conference on Decision and Control*, pages 3492–3497, 2006.

L'École Polytechnique se spécialise dans la formation d'ingénieurs et la recherche en ingénierie depuis 1873

École Polytechnique de Montréal

École affiliée à l'Université de Montréal

Campus de l'Université de Montréal
C.P. 6079, succ. Centre-ville
Montréal (Québec)
Canada H3C 3A7

www.polymtl.ca

ÉCOLE
POLYTECHNIQUE
MONTRÉAL