Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Graphics and Algorithms
TU Wien
Favoritenstr. 9/186-3
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801 - 18661
Fax: +43 (1) 58801 - 18697
E-mail: sek@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-138

183.151 Selected Chapters of Image Processing

SS 2016, Graphs: Matching and Distance
Student's contributions

Wen Chao Chen, Max Langer, Daniel Pucher,
Carmine Sansone, Domenico Verlotta,
edited by Walter G. Kropatsch
Institute of Computer Graphics and Algorithms
Pattern Recognition and Image Processing Group 186/3

Abstract

This technical report presents a collection of selected papers, submitted by students in the course "Selected Chapters of Image Processing" (VU 183.151) of the Pattern Recognition and Image Processing group during summer term 2016.

2016: Graphs: Matching and Distance

Basic document: Habilitation of Kaspar Riesen [1]

Most illustrations, algorithms and formulas are based on this basic document and are not explicitly cited in all cases.

1 New Working Mode 2016

Each lecture unit (except begin) will be subdivided into three parts:

1. Summary of discussion of last lecture unit.
2. Presentation of new book chapters and/or related scientific articles.
3. Discussion lead by opponent.

2 Student's Tasks

1. Select a topic and present in part 2 of the respective lecture unit;
2. as opponent: prepare a few initial critical statements (1-2 slides);
3. actively participate in the discussion;
4. Write a summary of the discussion of the preceding lecture unit.

Reports, presentations and slides are the basis for evaluation. Reports of presentations and discussions are part of this script.

3 Topics/Chapters 2016

KR refers to the Habilitation of Kaspar Riesen [1].

Date	Speaker Opponent report	Topic (2+3)	page
3. 3.2016	Walter Kropatsch	Introduction	
17. 3.2016	Wen Chao Chen Carmine Sansone Domenico Verlotta	(Sub-)Graph Matching (KR sect.1)	3 7
7. 4.2016	Max Langer Daniel Pucher Wen Chao Chen	Graph Edit Distance (GED; KR sect.2)	10 14
14. 4.2016	Daniel Pucher Wen Chao Chen Carmine Sansone	Benchmarking with Graph Data Sets (KR sect.A)	16 20
28. 4.2016	Carmine Sansone Domenico Verlotta Max Langer	Improvements by Search (KR sect.4)	27 41
23. 6.2016	Domenico Verlotta Max Langer Daniel Pucher	Improvements by Learning (KR sect.5)	43

A summary of selected chapters since 2002 completes this TR (p.45).

Graph Matching: Summary

Selected Chapters in Image Processing SS 2016

Wen Chao Chen (1129468)

April 13, 2016

1 Graph Matching

Graph Matching is a structural approach to pattern recognition which makes use of graph data structures to outline relationships or similarities between certain patterns through node representation and edge connectivity. Furthermore, by labeling nodes and edges, additional information for pattern matching is provided. It is also very important to note that by using graphs the pattern size and complexity are adaptive. The adaptive size often becomes an argument which decides whether one opts for structural pattern recognition, since feature vectors in the statistical approach are required to not only have a fixed size, but also need to have corresponding pattern features in order to be compared to each other. However, as the size of the feature vectors never change, the complexity for the statistical approach also never changes, such that a greater efficiency is given, compared to graph matching. The biggest worry is that the complexity of graph matching increases exponentially, as nodes and edges are inserted into a graph adaptively. In general, Graph Matching distinguished between two types: *Exact Graph Matching* and *Error-Tolerant Graph Matching*. The complexity dilemma becomes even more apparent in case of Exact Graph Matching, which is introduced next.

1.1 Exact Graph Matching

The objective of Exact Graph Matching is to provide information about the dissimilarity of compared graphs. Therefore, it is a necessity to use a method which determines whether or not two graphs or subgraphs correspond to each other such that a bijective function mapping one graph g_1 to another graph g_2 can be defined. In regard to bijective functions for graph matching, the concepts of *Graph Isomorphism* and *Subgraph Isomorphism* have been established. Graph Isomorphism for two graphs only holds, if the node and edge structure, as well as the labeling, are identical. *Subgraph Isomorphism* can be seen as an extension of Graph Isomorphism, such that the conditions hold if an induced subgraph of a graph g_1 is isomorphic to another graph g_2. Figure1 provides an illustrative explanation.

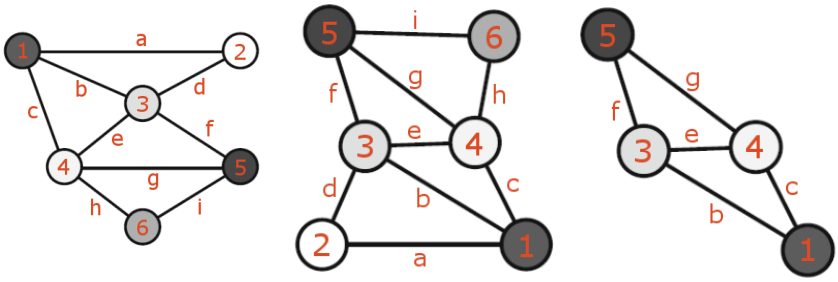


Figure 1: Graph (b) is isomorphic to graph (a), and graph (c) is isomorphic to a subgraph of (a).

Considering that for subgraph isomorphism all possible subgraphs of the larger graph have to be checked, it has been proven that subgraph isomorphism is NP-complete. That means that there is unclear, how long the algorithm will have to perform in order to find a solution or if there is a solution in the first place. Furthermore, the performance time increases even more, as the complexity of a pattern increases. In general, graphs come with a polynomial complexity, but if we were to restrict the problem of matching graphs to a special class of graphs, e.g, complete graphs, the complexity could be reduced, since the labeling would not have any significance anymore. The value resulting from the isomorphism evaluation is either 0 for similarity or 1 for dissimilarity, which aims to be an indication on the similarity of two graphs. However, this indication has no significance whatsover, since 0 is only returned, if and only if graph structure and label are identical. In all other cases, 1 is returned. A more significant dissimilarity is formulated, when the Maximum Common Subgraph or the Minimum Common Supergraph is considered. Roughly speaking, the Maximum Common Subgraph can be described as the greatest common denominator of two graphs, while the Minimum Common Supergraph represents the union of two graphs. Figure 2 gives an example for each case. By determining the subgraph, or respectively the supergraph, it is now possible to compute distance measures under consideration and in relation to the size of the graphs.

The introduced distance models based on the Maximum Common Subgraph are therefore,

$$d_{MSC}(g_1, g_2) = 1 - \frac{|msc(g_1, g_2)|}{\max\{|g_1|, |g_2|\}}. \tag{1}$$

$$d_{WGU}(g_1, g_2) = 1 - \frac{|msc(g_1, g_2)|}{|g_1| + |g_2| - |msc(g_1, g_2)|}. \tag{2}$$

$$d_{UGU}(g_1, g_2) = |g_1| + |g_2| - 2\,|msc(g_1, g_2)|\,. \tag{3}$$

where d_{WGU} and d_{UGU} can be reformulated in a way such that the Minimum Common Supergraph is taken into consideration. When the definition of Minimum Common Supergraph goes by,

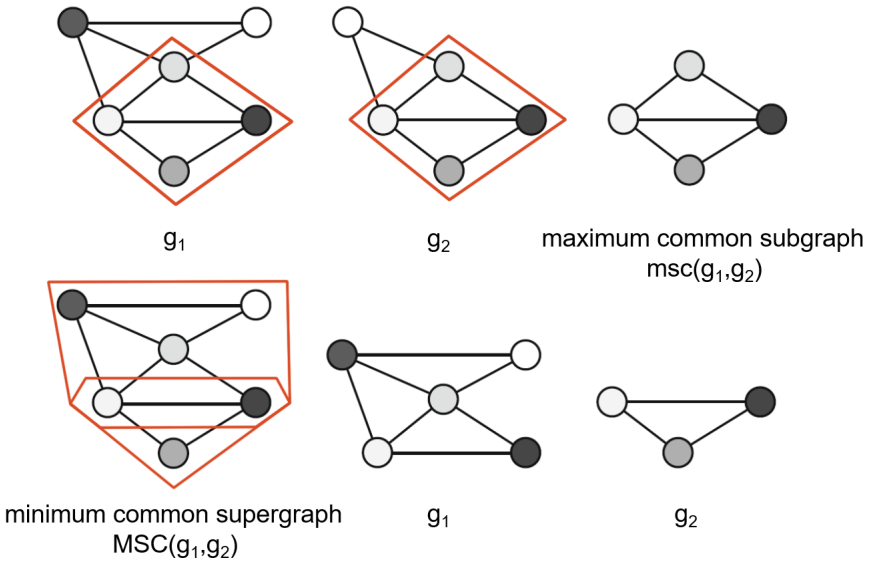


Figure 2: Example for a Maximum Common Subgraph as well as a Minimum Common Supergraph.

$$|MSC(g_1, g_2)| = |g_1| + |g_2| - |msc\,(g_1, g_2)|\,. \tag{4}$$

then the reformulated distance models,

$$d_{UGU}(g_1, g_2) = |MSC\,(g_1, g_2)| - |msc\,(g_1, g_2)|\,. \tag{5}$$

$$d_{WGU}(g_1, g_2) = d_{MMSCN}(g_1, g_2) = 1 - \frac{|msc\,(g_1, g_2)|}{|MSC\,(g_1, g_2)|}. \tag{6}$$

concludes that the Maximum Common Subgraph represents a lower bound on the similarity of two graphs, whereas and the Minimum Common Supergraph provides an upper bound.

1.2 Error-Tolerant Graph Matching

For real-world scenarios, it is nearly if not completely impossible to find two identical patterns due to noise. This, however, connotes that Exact Graph Matching is not practicable, since all or part of the graph structures as well as the labeling have to be preserved. Error-Tolerant Graph Matching takes a little different approach to graph matching such that slight differences are not only tolerated, but the structure of a graph itself is processed in order to provide information, whether or not two graphs are similar to each other. In general, the concept of the Error-Tolerant Graph Matching can be described as follows: The goal is to find a mapping from one graph to another, by processing the graph through editing operations such as node and edge insertion and

deletion. For every processed operation, a cost is then assigned and the the algorithms main aim is a minimization of the overall cost. A generalization of the overall cost is illustrated in Figure 3.

$$c(f) = \overbrace{\sum_{\substack{u \in V_1 \\ f(u) \neq \varepsilon}} c(u, f(u))}^{\text{node mappings}} + \overbrace{\sum_{\substack{u \in V_1 \\ f(u) = \varepsilon}} c(u, \varepsilon)}^{\text{node deletions}} + \overbrace{\sum_{\substack{v \in V_2 \\ f^{-1}(v) = \varepsilon}} c(\varepsilon, v)}^{\text{node insertions}} + \underbrace{\sum_{\substack{(u,v) \in E_1 \\ (f(u), f(v)) \in E_2}} c((u, v), (f(u), f(v)))}_{\text{edge mappings}} + \underbrace{\sum_{\substack{(u,v) \in E_1 \\ (f(u), f(v)) \notin E_2}} c((u, v), \varepsilon)}_{\text{edge deletions}} + \underbrace{\sum_{\substack{(f(u), f(v)) \in E_2 \\ (u,v) \notin E_1}} c(\varepsilon, (f(u), f(v)))}_{\text{edge insertions}}$$

Figure 3: The overall cost function of the Error-Tolerant Graph Matching paradigm.

Several approaches to Error-Tolerant Graph Matching have been proposed over the years, of which some use relaxation labeling techniques, artificial neural networks, spectral decomposition or Kernel machines. Another flexible and universal error-tolerant graph matching paradigm is, the Graph Edit Distance, which also uses the concept of cost minimization, and is discussed in the following chapter.

(Sub-) Graph Matching: Discussion Report.

Verlotta Domenico

March 7, 2016

Graph matching has applicationsin a variety of fields from computer vision to chemistry and molecular biology. In graph matching, patterns are modeled as graphs and pattern recognition amounts to finding a correspondence between the nodes of different graphs.

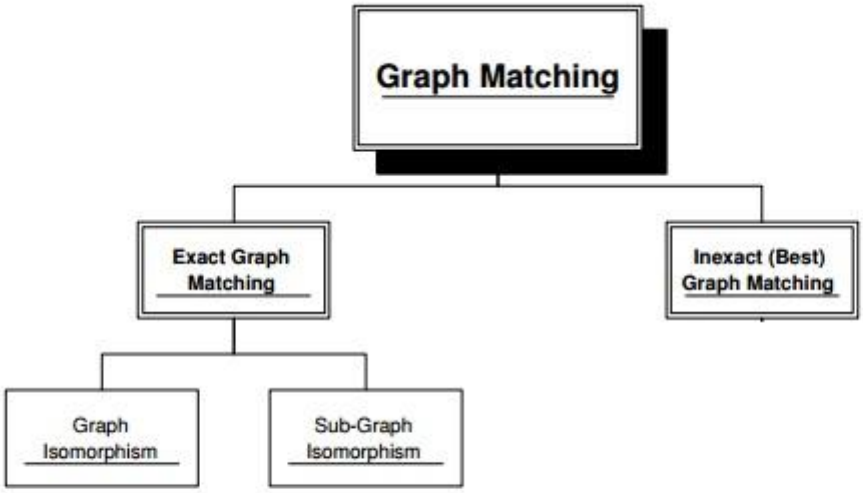
This processing is to be performed by a computer automatically without the assistance of a human expert, a useful way of representing the knowledge is by using graphsthat rappresent a good way of representing objects.

Definition *Graph* : Let *LV* and *LE* be finite or infinite label sets for nodes and edges, respectively. A *graph g* is a four-tuple $g = (V, E, \mu, \nu)$, where

- V is the finite set of nodes,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \rightarrow LV$ is the node labeling function, and
- $\nu : E \rightarrow LE$ is the edge labeling function.

The set of all graphs over the label alphabets *LV* and *LE* (also referred to as *graph domain*) is denoted by G . The size of a graph g is denoted by $|g|$ and is defined as the number of nodes, i.e., $|g| = |V|$.

Exact and inexact graph matching



In model-based pattern recognition problems, given two graphs graph G1 and the data graph G2 the procedure of comparing them involves to check whether they are similar or not. Generally speaking, we can state the graph matching problem as follows:

Given two graphs G1 = (v1, e1) and G2 = (v2, e2), with |v1| = |v2|, the problem is to find a one-to-one mapping $f : v1 \rightarrow v2$ with $(u, v) \in e2$ iff $(f(u), f(v)) \in e1$.

When such a mapping f exists, this is called an isomorphism, and G1 is said to be isomorphic to G2, this type of problems is said to be exact graph matching.

The term inexact applied to some graph matching problems means that it is not possible to find an isomorphism between the two graphs to be matched. This is the case when the number of vertices is different in both the model and data graphs.

This may be due to the schematic aspect of the model and the difficulty to segment accurately the image into meaningful entities. Therefore, in these cases no isomorphism can be expected between both graphs, and the graph matching problem does not consist in searching for the exact way of matching vertices of a graph with vertices of the other, but in finding the best matching between them. This leads to a class of problems known as inexact graph matching.

Questions

- *I'd like to implement an application to solve a determinate problem based on the subgraph matching, can I solve my problem also if the problem is NP-complete?*

Graph matching is considered to be one of the most complex problems in object recognition in computer vision, the NP-complete problem is solvable (but the computation time is exponential).

In the lecture there are different type of algoritm we can use for the graph matching for example, Random walks, Decision trees, VF2.

These type of the problem is impossible to resolve (we can find a solution but we don't know that solution in the optimal solution) with an general algorith for the elevate complexite but you can create a smart program that resolve a class of the problem (you can create an euristic that dipens for a class of the applycation in this case we can find the optimal solution because we considerat a class of the problem).

- *How can we built a function cost and what is the influence that the function cost has on the algorith that we chose to do the graph matching?*

- *A common way to make error-tolerant graph matching more efficient is to restrict considerations to special classes of graphs. Examples include the classes of ordered graphs, planar graphs, trees or graphs with unique node labels. If possible solve the problem of graph matching in a easy way restricting the class of the graph.*

- *No one of this algorithms (Spectral methods, Graph kernels) use semantic information. We have chosen to move our attention from the vector to the graph because with the graph we can represent relationships among different parts of the underlying patternand but we use algorithm that don't use this information.*

Graph Edit Distance

Max Langer

April 13, 2016

1 Definition

The basic idea of *Graph Edit Distance* (GED) is to measure the distance base on the operations needed to transform graph g_1 to graph g_2. In the book by Riesen [Rie15] three operations are considered: Insertion, Deletion and Substitution. By insertion (Figure 1a) a node is added to the graph. On the opposite, by deletion (Figure 1b) a node is removed from the graph. Substitution (Figure 1c) transforms one node into another. This is important with labeled graphs.

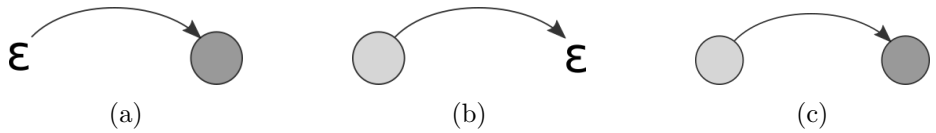


Figure 1: The three operations: (a) Insertion, (b) Deletion and (c) Substitution.

Before considering GED, the definition of an Edit Path is important (Definition 1).

Definition 1 (Edit Path)**.** *A set $e_1, \ldots, e_k$ of k edit operations e_i that transform g_1 completely into g_2 is called a complete edit path $\lambda(g_1, g_2)$ between g_1 and g_2.*

An Edit Path describes a set of operations, that transform one graph to another. As can be seen there are an infinite number of different edit paths. A next step to obtain a distance measure is to find an Edit Path that is as small es possible. This consideration motivates the Graph Edit Distance (Definition 2).

Definition 2 (Graph Edit Distance)**.** *Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ be the source and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ the target graph. The graph edit distance $d_{\lambda_{\min}}(g_1, g_2)$, or $d_{\lambda_{\min}}$ for short, between g_1 and g_2 is defined by*

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \Upsilon(g_1, g_2)} \sum_{e_1 \in \lambda} c(e_i)$$

$\Upsilon(g_1, g_2)$ is the set of all complete edit paths and $c(e)$ the cost for an operation e.

Figure 2 shows an example Edit Path, that also is the minimum Edit Path and therefore the base for the GED calculation. The path is $\lambda = \{(u_1 \rightarrow \varepsilon), (u_2 \rightarrow v_3), (u_3 \rightarrow v_2), (u_4 \rightarrow v_1)\}$. Because we know the resulting graph g_2 we can find the implicit edge operation necessary to transform the graph: $\{((u_1, u_2) \rightarrow \varepsilon), ((u_2, u_3) \rightarrow (v_3, v_2)), ((u_3, u_4) \rightarrow (v_2, v_1)), ((u_2, u_4) \rightarrow \varepsilon)\}$

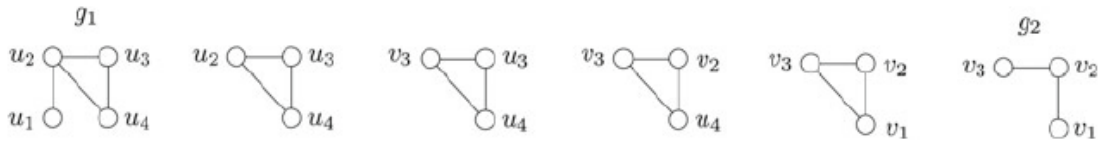


Figure 2: Example Edit Path.

2 Edit Cost Function

As can be seen in Definition 2 every operation is associated with a cost value. The idea behind the function $c(e)$ is that not every operation as equally costly.

To limit the possible Edit Paths for the GED calculation the cost function has to fulfill the following three conditions.

1. Non-negativity: $c(e) \geq 0$, for all edit operations e
2. Substitution over insert/delete: $c(e) > 0$, for all deletions and insertion operations e
3. Triangle inequality: $c(u \rightarrow w) \leq c(u \rightarrow v) + c(v \rightarrow w)$. This ensures, that only one operation is used if possible instead of using intermediate nodes.

All those conditions considered the number of Edit Paths to take into account for GED from graph g_1 with n nodes to graph g_2 with m nodes is m^n.

Riesen gives two examples of cost functions. For unlabeled graphs the simplest cost function is to define unit costs to insertion/deletion and have substitution free of cost, because one node can not be told from another.

A more common situation are labeled graphs. For symmetry insertion and deletion must have the same value τ. Substitution is calculated with respect to the node labels $(c(u \rightarrow v) = \|\mu_1(u) - \mu_2(v)\|_p)$.

To define good edit cost functions prior knowledge of the labels is important.

3 Computation

An exact computation can be done with an A*-based search method. This method uses a tree to map all possible edit paths and find the smallest GED in the leaf nodes.

In Figure 3 the tree for the example in Figure 2 is given. Each layer represents all nodes the node on the left side can be transformed to. The tree nodes show the current

cost for node and edge operations, leading to a cost of 4 in the shortest Edit Path marked with a bold line.

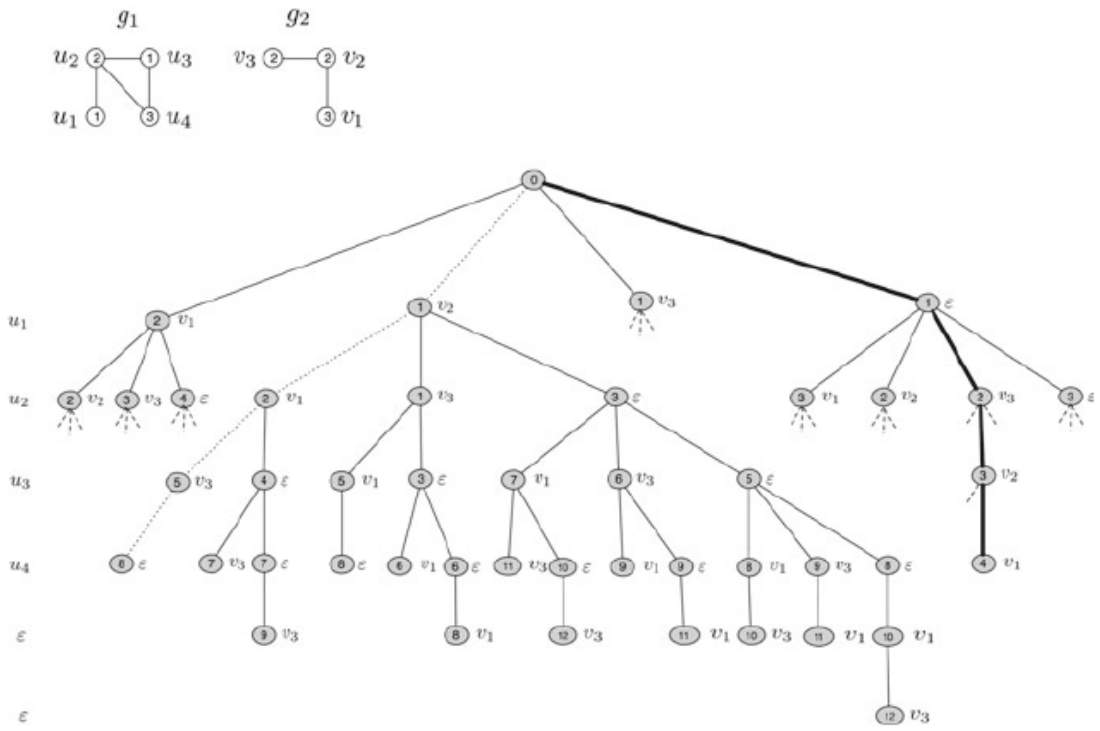


Figure 3: Exact GED computation using a tree search.

Because this algorithm is exponential and therefore to time consuming to calculate, approximations are needed. Riesen's book deals with many different approaches.

4 Pattern Recognition

In the following three methods that can use GED in pattern recognition are discussed.

4.1 Nearest-Neighbor

Nearest-neighbor is a simple technique that uses N training graphs with corresponding class labels. To find the correct class of a graph g the nearest training graph (GED is minimal) is searched for. Graph g is labeled with this nearest training graph's label. For a better result k nearest neighbors can be considered, resulting in the class most of them are in.

4.2 Kernel-Based

There exist many methods for statistical pattern recognition. To use these well established statistical tools with graphs we relate to the kernel trick.

The kernel trick states, that every dot-product can be rewritten as a function if this function is a positive definite kernel function. More formally that means that there exists a mapping *phi* so that $\kappa(x, x') = \langle \phi(x), \phi(x') \rangle$.

A positive definite kernel function is a function κ that fulfills the following condition:

$$\sum_{i,j=1}^{N} c_i c_j \kappa(x_i, x_j) \geq 0$$

$$\forall N \in \mathbb{N}, \forall \{c_1, \ldots, c_N\} \subseteq \mathbb{R}, \forall \{x_1, \ldots, x_n\}$$

Utilizing this kernel trick, we can use every statistical tool, that uses the dot product. Many statistical tools can be rewritten to do exactly that resulting in many available so called kernel machines. One example is SVM.

There exist multiple positive definite kernel functions based on the GED, like $-d(g_1, g_2)^2$.

4.3 Graph Embedding

As with the kernel based methods graph embedding allows the use of graphs with statistical methods. With graph embedding a graph is turned into a vector first and then the statistical tool is used the same way as with feature vectors.

First prototype graphs $\mathscr{P} = \{p_1, \ldots, p_n\}$ are defined. Those graphs have to be representative and cover all possible classes. One would for example use a prototype of every letter of the alphabet for letter classification.

In a next step a graph $g \in \mathscr{G}$ is mapped to a vector $v \in \mathbb{R}^n$:

$$\phi_n^{\mathscr{P}}(g) = (d(g, p_1), \ldots, d(g, p_n))$$

$d(g_1, g_2)$ is the GED of g_1 to g_2.

The resulting vector can be used in statistical applications.

References

[Rie15] Kaspar Riesen. Structural pattern recognition with graph edit distance. 2015.

Graph Edit Distance: Discussion Report

Selected Chapters in Image Processing SS 2016

Wen Chao Chen (1129468)

April 10, 2016

1 Regarding the general understanding

- The defined substitution operation is only relevant for labeled graphs, since, logically, for unlabeled graphs, label changes do not matter.
- In case of a linked structure, one concept is deleting nodes and edges through the adjacency matrix. Hereby, however, the edges are not automatically removed. It is necessary to look through the whole graph in order to delete incident edges .
- One question that came up during the discussion were the given similarity kernel, in particular $-d(g_1, g_2)^2$. Given the circumstance of requiring positive definite kernels, the provided kernel example does not seem to make sense,since the condition of a positive definite kernel function would not hold due to the to the minus-sign in front.

See answer next page!

2 Questions

- *Is there additional use for the information we gathered in the edit path selection procedure?*
- *Why don't we consider graph contraction in addition to removal when computing the graph edit distance?* Using graph contraction for the purpose of building graph pyramids also preserves graph connectivity. The graph can be shrinked that way, and the inverse would deconstruct and reinsert a new edge. So in principal, we can redefine the set of basic operations of the graph edit distance. As a matter of fact, the operation set introduced in the book does not consider planar graphs. However, with a proper definition using the redefined set of operations (including contraction), we can preserve the planarity.

4.3.4 *Using Non-Positive Definite Kernel Functions*

In previous sections, kernel machines have been introduced for positive definite kernel functions. The main results from the theory of kernel functions, such as Theorem 4.1 about the equivalence of valid kernels and inner products in kernel feature spaces, require kernel functions to be symmetric and positive definite. In fact, it seems to be surprising that the single condition of positive definiteness implies a huge number of mathematical properties related to the underlying kernel function. The general kernel trick discussed in Sec. 4.2.2, for instance, requires positive definite kernels as well. However, in the case of SVMs (Sec. 4.3.1), kernel PCA (Sec. 4.3.2), kernel FDA (Sec. 4.3.2), and some other translation-invariant problems, it turns out that kernels only need to be conditionally positive definite in order to be applicable [Schölkopf and Smola (2002)]. Hence, this opens up the class of kernel functions that are suitable for the kernel machines described in this chapter. Although the complete theory of kernel functions might not be available if conditionally positive definite kernels are used, the interpretation of the kernel machines considered in this book remains valid, and inner products can be replaced by conditionally positive definite kernels.

From the theoretical point of view, valid kernel functions are particularly interesting because of their associated vector space structure. Yet, in practice it seems to be difficult to design valid kernel functions for certain applications. In such cases, it may be tempting to deliberately plug an invalid kernel function into a kernel machine. Using SVMs, for instance, the training procedure is not guaranteed to terminate if invalid kernels are used. On the other hand, if the training succeeds, one obtains an empirical classifier for which the theory of SVMs does not hold true, but which may nevertheless perform well. Accordingly, a number of successful applications of indefinite kernels functions to kernel machines have been reported [Bahlmann *et al.* (2002); DeCoste and Schölkopf (2002); Shimodaira *et al.* (2002); Moreno *et al.* (2004); Jain *et al.* (2005)].

In a recent analysis of indefinite kernels [Haasdonk (2005)], it was observed that SVM learning with indefinite kernels offers a reasonable geometrical interpretation. The main conclusion is that using SVMs in conjunction with arbitrary symmetric kernels can be understood as hyperplane classification in pseudo-Euclidean spaces. Pseudo-Euclidean spaces are defined by the direct product of two Euclidean spaces. In the case of valid kernels, the objective of SVM learning is to find a maximum-margin hyperplane separating two classes. If indefinite kernels are used, the SVM procedure aims at

maximizing the distance between the convex hulls of two classes. This geometrical interpretation is possible as distance, orthogonality, hyperplanes, and convex hulls can be defined in reasonable terms in pseudo-Euclidean spaces. Hence, even if kernel functions are applied that violate the conditions of (conditional) positive definiteness, training SVMs may nevertheless be a viable procedure with an intuitive and reasonable interpretation.

Given a kernel matrix derived from an arbitrary symmetric kernel function, its suitability for application to an SVM can be predicted to a certain degree [Haasdonk (2005)]. An increasing number of negative eigenvalues of the kernel matrix indicates that the convex hull separation of the training data is difficult. Another criterion is the distance of the class means in the pseudo-Euclidean space. If the Euclidean distance is negative, which is possible in pseudo-Euclidean spaces, a solution of the convex hull separation problem need not exist. Hence, indefinite kernels that are suitable for SVMs are characterized by a small number of negative eigenvalues of their kernel matrix and a positive distance of the class means. These criteria allow us to quantify how appropriate a given invalid kernel function may be for SVM classification.

4.4 Nearest-Neighbor Classification Revisited

This section will return to the nearest-neighbor classification briefly introduced in Sec. 3.6 and shed new light on the nearest-neighbor paradigm. For the formal definition of nearest-neighbor classifiers, let us assume that a pattern space $\mathcal{X}$, a space of class labels $\mathcal{Y}$, and a labeled training set of patterns $\{(x_i, y_i)\}_{i=1\ldots m} \subseteq \mathcal{X} \times \mathcal{Y}$ is given. The *1-nearest-neighbor classifier* (1NN) is defined by assigning a test pattern $x \in \mathcal{X}$ to the class of its most similar training pattern. Accordingly, the 1NN classifier $f : \mathcal{X} \to \mathcal{Y}$ is defined by

$$f(x) = y_j, \text{ where } j = \arg\min_{i=1\ldots m} d(x, x_i) \ , \tag{4.12}$$

where $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a pattern dissimilarity measure. Clearly, the 1NN classifier assigns the same weight to each training pattern, no matter if a pattern constitutes an outlier or a true class representative. Its decision boundary is composed of piecewise linear functions, which makes the 1NN classifier very flexible and applicable to arbitrarily complex class distributions.

The training of a nearest-neighbor classifier consists of providing the classifier with a training set of prototypes — an actual process learning the

3 Additional annotations

- One elimination of a single node is more devastating for a graph with low number of vertices, compared to the deletion of a vertex from a graph with a large number of vertices. A small change on a small graph may change the whole graph structure, e.g change an unconnected graph into a connected graph.
- A major deficiency of the book is that there are no preconditions (e.g., for inserting an edge, it would be a necessary precondition to know the two endpoints). Let's take a look at the incidence matrix which is another data structure of the graph in order to illustrate the necessity of preconditions. The problem becomes apparent, if we try to delete a vertex in the incidence matrix, since it will result in a pending edge. Missing preconditions for the operations are a major deficiency. When deleting a vertex, the vertex has to be part of the graph. How vertices are connected plays a big role from a image perspective (e.g., closed and open envelope). However, this is apparently not noticeable in either the adjacency or the incidence graph. Another subject to consider is a series of operations, since not every operation is available everytime (e.g, sometimes nodes have to be inserted first in order to insert new edges). Riesen avoids this precondition by defining that the vertices are considered first before turning to the edges.
- Concerning the computation of sets of graphs, the idea is to define the median of the set of graphs to have statistical properties and a corresponding number to a set of graphs. Example Given, letter drawing: There is a set of prototype. Now, which is the graph that represents the mean of the set of graph. Use the Kernel trick to do graph matching to sets of graphs by linear combinations.
- 2.1 talks about the inversion of an operation. However, there is no specific definition of the inversion, since insertion of a deletion is complex. (e.g., deleting a vertex also deletes incident edges).

Experimental Evaluation of Sorted Beam Search

Daniel Pucher

Vienna University of Technology

daniel.pucher@tuwien.ac.at

April 13, 2016

Overview

Quadratic Assignment Problem

- The graph edit distance belongs to the family of quadratic assignment problems (QAPs) $\rightarrow$ NP-complete
- QAPs deal with the problem of assigning n entities of a first set S to n entities of a second set Q.
- Assignments formally represented with permutations $(\varphi_1, ..., \varphi_n)$ where the first entity $s_1 \in S$ is mapped to entity $q_{\varphi_1} \in Q$, the second entity $s_2 \in S$ is mapped to entity $q_{\varphi_2} \in Q$, and so on.

Quadratic Assignment Problem

- QAPs are generally stated on sets with equal cardinality.
- A solution to a QAP corresponds to a bijective assignment.
- Therefore a number of empty nodes ε is added to every node set of two graphs so that the number of nodes in every graph is equal. This gives new node sets V_1^+ and V_2^+.
- Adjacency matrices A and B also offer equal dimensions.

Quadratic Assignment Problem

$$\mathbf{C} = \begin{array}{c|cccc|cccc} & v_1 & v_2 & \dots & v_m & \varepsilon_1 & \varepsilon_2 & \dots & \varepsilon_n \\ \hline u_1 & c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\ u_2 & c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ u_n & c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\ \hline \varepsilon_1 & c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \varepsilon_2 & \infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \varepsilon_m & \infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0 \end{array}$$

Quadratic Assignment Problem

Optimization problem:

$$(\varphi_1, \dots, \varphi_{(n+m)}) = \underset{(\varphi_1,\dots,\varphi_{(n+m)}) \in \mathcal{S}_{(n+m)}}{\arg\min} \left[\sum_{i=1}^{n+m} c_{i\varphi_i} + \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c(a_{ij} \rightarrow b_{\varphi_i\varphi_j}) \right]$$

The linear term refers to the sum of node edit costs.

The quadratic term refers to the implied edge edit cost.

The minimum cost permutation $(\varphi_1, ..., \varphi_{n+m})$ and the minimum cost edit path λ_{min} are equivalent.

Bipartite Graph Edit Distance

- Approximates computation of graph edit distance.
- Reduce QAP to an instance of a Linear Sum Assignment Problem (LSAP).
- LSAPs optimize the permutation $(\varphi_1, ..., \varphi_{n+m})$ with respect to the linear term only.
- By omitting the quadric term, the structural relationships between nodes is neglected.

Linear Sum Assignment Problem

- LSAPs optimize the permutation $(\varphi_1, ..., \varphi_{n+m})$ with respect to the linear term only.
- By omitting the quadric term, the structural relationships between nodes is neglected.
- To integrate knowledge about the graph structure, the minimum sum of edge edit operation costs is added to the corresponding node operation.

$$c^*_{ij} = c_{ij} + \min_{(\varphi_1,\ldots,\varphi_{(n+m)}) \in \mathscr{S}_{(n+m)}} \sum_{k=1}^{n+m} \left(c(a_{ik} \to b_{j\varphi_k}) + c(a_{ki} \to b_{\varphi_k j})\right)$$

Linear Sum Assignment Problem

Optimization problem:

$$(\varphi^*_1, \ldots, \varphi^*_{(n+m)}) = \underset{(\varphi^*_1,\ldots,\varphi^*_{(n+m)}) \in \mathscr{S}_{(n+m)}}{\arg\min} \sum_{i=1}^{n+m} c^*_{i\varphi^*_i}$$

The permutation corresponds to a bijective assignment of entities in V_1^+ to entities in V_2^+.

$$\psi = \{(u_1 \to v_{\varphi^*_1}), (u_2 \to v_{\varphi^*_2}), \ldots, (u_{m+n} \to v_{\varphi^*_{m+n}})\}$$

Beam Search

- One major problem of the BP-GED is the over- or underestimation of the true edit distance.
- Due to a few incorrectly assigned nodes in the complete edit path ψ.
- Quality is improved by systematically varying the initial assignment ψ. $\to$ Search Strategies

Sorting Criteria

- Sorting of node edit operations $(u_i \to v_{\varphi_i}) \in \psi$.
- The terminology *Inverse* is used for sorting strategies that process less-evident edit operations first.
- Sorting strategies that process evident edit operations first are named without the suffix *Inverse*.
- Evident edit operations are those that are supposed to be correct with respect to the exact edit path.

Sorting Criteria 1 - Confident

- The source nodes u_i of the edit operations $(u_i \to v_{\varphi_i}) \in \psi$ are weighted according to $c_{i\varphi_i} \in \mathbf{C}$.

Sorting Criteria 2 - Unique

- The source nodes u_i of the edit operations $(u_i \rightarrow v_{\varphi_i}) \in \psi$ are weighted according to

$$\max_{\forall j=1,\ldots,m} c_{ij} - c_{i\varphi_i}$$

- The weight corresponds to the maximum difference between the cost of the actual edit operation and the cost of a possible alternative matching node for u_i.
- Can be negative $\rightarrow$ current edit operation suboptimal

Sorting Criteria 3 - Divergent

- Prioritize nodes u_i that have a high divergence among all possible node edit costs.
- For each row i in the cost matrix $\mathbf{C}$

$$\sum_{j=1}^{m-1} \sum_{k=j+1}^{m} |c_{ij} - c_{ik}|.$$

- High divergence $\rightarrow$ local node assignments that are easier to be conducted.

Sorting Criteria 4 - Leader

- Weights nodes u_i according to the maximum difference between the minimum cost assignment of node u_i and the second minimum cost assignment of u_i.

$$\frac{\text{min2}_i - \text{min1}_i}{\text{min2}_i}$$

- The higher the difference, the less difficult is the local assignment.

Sorting Criteria 5 - Interval

- Compute intervals for each row i and each column j of the upper left part of $\mathbf{C}$.
- Intervals δ_{r_i} and δ_{c_j} are the absolute difference between the maximum and minimum in row i (or column j).
- Compute mean of all row and column intervals: $\bar{\delta_r}$ and $\bar{\delta_c}$
- Weights
  - 1, if $\delta_{r_i} > \bar{\delta_r}$ and $\delta_{c_{\varphi_i}} > \bar{\delta_c}$
  - 0, if $\delta_{r_i} < \bar{\delta_r}$ and $\delta_{c_{\varphi_i}} < \bar{\delta_c}$ 0.5, otherwise
- When the intervals of rows and columns are larger than the mean intervals, the row and column of the edit operation are in general easier to handle than others.

Sorting Criteria 6 - Deviation

- Means $\bar{\theta_{r_i}}$ and $\bar{\theta_{c_j}}$ and deviations $\bar{\sigma_{r_i}}$ and $\bar{\sigma_{c_j}}$ are calculated for each row i and each column j of the upper left part of $\mathbf{C}$.
- Weights
  - Initially, the weight for an edit operation is 0.
  - If $c_{i\varphi_i} < \bar{\theta_{r_i}} - \bar{\sigma_{r_i}}$, add 0.25 to weight, compute total number p of assignments in row i that also fulfill this condition and add $0.5/p$ to the weight.
  - Repeat the previous step for column $j = \varphi_i$ using $\bar{\theta_{c_j}}$ and $\bar{\sigma_{c_j}}$.

Sorting Criteria 6 - Deviation

Given an edit operation $(u_i \rightarrow v_{\varphi_i}) \in \psi$ with cost $c_{i\varphi_i}$

- If the cost is lower than the mean minus the deviation, it is assumed that the assignment cost is low enough to be considered as evident.
- Weight increases if in the corresponding row or column only few or no other evident assignments are available.
- Assignments with small weights correspond to difficult assignments.

Experimental Evaluation

- Eight data sets with graphs representing molecular compounds, fingerprint images, distorted letter drawings and symbols from architectural and electronic drawings.
- Goal of the experimental evaluation is to research the effects of the novel reordering procedures on the distance accuracy.
- Beam search procedure is repeated ten times with random permutations of the node assignments in ψ.

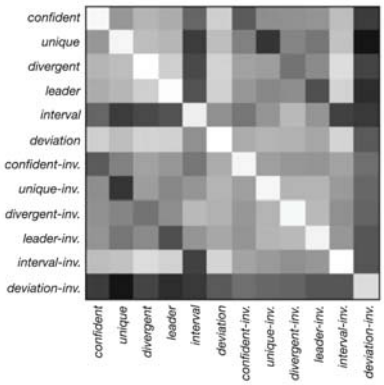
Experimental Evaluation

Table 8.1 Mean relative overestimation with respect to the exact distance (d_{GO}) (including rank) and mean run time for one matching (d_{GT}) in ms

Algorithm	k	AIDS		Fingerprint		GREC		LETTER		Acyclic		Alkane		MAO		PAH	
		d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}
BP-Beam		21.75	1.82	19.69	1.49	19.77	2.61	20.41	1.41	23.31	8.31	20.55	7.49	42.05	18.22	52.94	47.57
Confident	53	15.81 (5)	1.81	11.99 (2)	1.48	20.57 (10)	2.63	15.37 (8)	1.38	17.35 (7)	8.47	16.69 (10)	7.54	32.13 (4)	17.99	48.85 (7)	47.97
Confident Inv.	49	16.36 (7)	1.83	21.06 (12)	1.49	13.60 (1)	2.64	11.10 (3)	1.38	16.73 (6)	8.51	13.42 (3)	7.59	35.08 (7)	18.14	49.50 (10)	46.15
Unique	50	15.91 (6)	1.81	18.07 (10)	1.49	15.74 (7)	2.65	15.21 (7)	1.38	20.52 (10)	8.45	15.08 (7)	7.50	29.44 (2)	18.28	47.38 (1)	47.78
Unique Inv.	50	16.49 (8)	1.81	14.27 (6)	1.50	14.87 (5)	2.62	12.74 (4)	1.39	16.07 (5)	8.48	16.23 (9)	7.57	33.99 (5)	18.35	49.19 (8)	47.76
Divergent	68	14.03 (1)	1.80	16.29 (8)	1.50	16.68 (8)	2.64	15.92 (9)	1.39	23.40 (12)	8.40	17.84 (11)	7.46	40.95 (10)	18.01	49.19 (9)	48.70
Divergent Inv.	55	21.09 (12)	1.80	16.44 (9)	1.49	15.64 (6)	2.62	12.81 (5)	1.39	14.70 (2)	8.45	13.65 (4)	7.72	41.03 (11)	18.17	48.50 (6)	49.50
Leader	61	15.38 (2)	1.82	14.43 (7)	1.50	21.42 (11)	2.61	16.58 (11)	1.38	20.45 (9)	8.44	13.18 (2)	7.54	35.47 (8)	18.00	49.54 (11)	47.62
Leader Inv.	48	17.52 (9)	1.80	9.11 (1)	1.49	13.80 (4)	2.64	13.09 (6)	1.39	15.89 (4)	8.41	15.62 (8)	7.56	30.31 (3)	18.13	47.73 (3)	47.73
Interval	52	18.32 (10)	1.80	13.64 (5)	1.51	13.73 (3)	2.65	10.50 (2)	1.39	14.35 (1)	8.51	12.54 (1)	7.60	34.59 (6)	18.04	48.27 (4)	47.62
Interval Inv.	66	15.53 (4)	1.81	12.98 (3)	1.49	18.63 (9)	2.63	16.22 (10)	1.39	22.19 (11)	8.38	19.34 (12)	7.45	46.05 (12)	17.85	48.31 (5)	47.84
Deviation	74	20.94 (11)	1.81	13.24 (4)	1.50	21.65 (12)	2.59	17.15 (12)	1.39	20.33 (8)	8.39	14.67 (6)	7.48	39.89 (9)	18.05	52.00 (12)	48.46
Deviation Inv.	18	15.39 (3)	1.81	10.20 (1)	1.50	13.66 (2)	2.66	9.20 (1)	1.39	14.80 (3)	8.53	14.27 (5)	7.58	24.81 (1)	18.14	47.62 (2)	48.36

Experimental Evaluation

Fig. 8.1 Sum of ranks for all 144 combination pairs (the *darker* the *square* is, the *lower* the rank and thus the better the approximation)



Experimental Evaluation

Table 8.2 Mean relative overestimation with respect to the exact distance (d_{GO}) in % and mean run time for one matching (d_{GT}) in ms

Algorithm	AIDS		Fingerprint		GREC		LETTER		Acyclic		Alkane		MAO		PAH	
	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}
BP-Beam	21.75	1.82	19.69	1.49	19.77	2.61	20.41	1.41	23.31	8.31	20.55	7.49	42.05	18.22	52.94	47.57
Unique/Deviation Inv.	10.09	3.32	7.15	2.61	9.37	5.01	5.75	2.54	11.69	16.11	9.35	14.38	17.16	35.91	42.73	94.05
Leader/Deviation Inv.	9.87	3.33	5.99	2.64	10.96	4.97	5.35	2.55	11.78	16.11	8.69	14.45	18.12	35.77	43.69	93.66
Unique/Unique Inv.	9.87	3.32	7.93	2.61	8.92	4.98	6.28	2.52	11.54	16.09	9.44	14.37	19.12	35.67	43.27	93.97

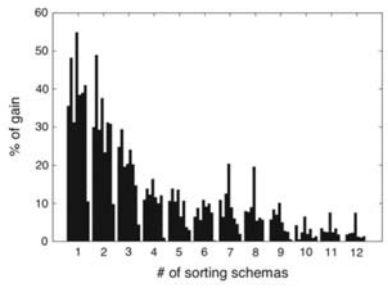
Experimental Evaluation

Table 8.3 Mean relative overestimation with respect to the exact distance (d_{GO}) in % and mean run time for one matching (d_{GT}) in ms

Algorithm	AIDS		Fingerprint		GREC		LETTER		Acyclic		Alkane		MAO		PAH	
	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}	d_{GO}	d_{GT}
A^*	0.00	25750.22	0.00	2535.42	0.00	7770.81	0.00	1.81	0.00	8924.17	0.00	812.77	0.00	13538.46	0.00	30495.54
BP	100.00	0.28	100.00	0.35	100.00	0.27	100.00	0.21	100.00	0.81	100.00	0.65	100.00	0.81	100.00	157.8
BP-Beam	21.75	1.82	19.69	1.49	19.77	2.61	20.41	1.41	23.31	8.31	20.55	7.49	42.05	18.22	52.94	47.57
SBP-Beam(1)	14.03	1.82	10.20	1.51	13.60	2.68	9.20	1.46	14.35	8.57	12.54	7.62	24.81	18.49	47.38	49.81
SBP-Beam(2)	9.83	3.35	5.21	2.68	9.62	5.08	5.74	2.66	10.99	16.43	8.63	14.50	17.16	35.72	42.73	97.42
SBP-Beam(3)	7.40	4.92	3.68	3.82	7.73	7.41	4.57	3.78	8.34	23.94	6.88	21.49	14.63	53.77	40.85	145.46
SBP-Beam(4)	6.59	6.35	3.17	4.98	6.78	9.85	3.82	5.00	7.37	31.34	6.19	28.59	12.86	70.95	40.46	187.76
SBP-Beam(5)	5.89	7.89	2.73	6.39	6.07	12.21	3.30	6.08	6.89	39.23	5.53	35.13	12.38	89.85	39.27	233.97
SBP-Beam(6)	5.51	9.42	2.49	7.21	5.73	14.54	2.94	7.35	6.26	47.70	4.98	42.17	11.44	106.79	39.19	281.09
SBP-Beam(7)	4.91	10.96	2.33	8.28	5.01	16.93	2.34	8.48	5.70	54.63	4.68	49.50	10.92	124.22	38.42	328.01
SBP-Beam(8)	4.52	12.68	2.15	9.50	4.56	19.35	1.88	9.70	5.39	62.42	4.39	56.93	10.30	140.77	38.42	374.35
SBP-Beam(9)	4.26	14.13	1.97	10.50	4.24	21.82	1.69	10.86	5.12	70.91	4.27	62.94	10.04	159.13	38.23	417.19
SBP-Beam(10)	4.08	15.51	1.96	11.68	4.14	24.21	1.58	12.05	5.02	77.84	4.13	69.64	9.95	175.27	37.73	468.53
SBP-Beam(11)	3.94	17.18	1.91	12.88	4.04	26.57	1.46	13.17	4.90	84.96	3.99	77.41	9.77	193.49	37.73	515.74
SBP-Beam(12)	3.87	18.59	1.87	14.40	3.95	28.69	1.35	14.36	4.84	92.37	3.95	84.10	9.64	212.01	37.73	562.24

Experimental Evaluation

Fig. 8.2 Gain of accuracy in each iteration of SBP-Beam(i) ($i = 1, \ldots, 12$) for each data set (each bar corresponds to one data set)



DISCUSSION REPORT

Appendix A: Experimental Evaluation of Sorted Beam Search

Selected Chapters in Image Processing SS 2016

Carmine Sansone (1529804)

April 14, 2016

Introduction

The following document is organized in three parts:

- The first part is the presentation which summarizes the main concepts that have been illustrated by the speaker.
- The second part is the question part and describes the questions that has been posed by the opponent.
- The third part is the discussion part and describes the reflections and the problematics raised during the third part of the lesson.

Presentation

The graph edit distance problem can be reformulated as an instance of a Quadratic Assignment Problem. Given two graphs, g_1 with *n* nodes and g_2 with *m* nodes, the first thing to do to reformulate the graph matching problem in a QAP is to add *m* empty nodes to the first graph and *n* empty nodes to the second graph, in this way we obtain two new graphs with the same number of nodes. We define a possible matching between the nodes of the first graph and the nodes of the second graph with a vector of *m+n* dimension having the following semantic: the number indicated in the i-th position of the array represents the number of the node in the second graph that match with the i-th node of the first graph.

If we match a node of the first graph with an empty node, that means that we are deleting that node; if we match an empty node of the first graph with a non-empty node of the second graph, that means that we are adding that node; if we match a node of the first graph with a node of the second graph, that means that we are replacing that node.

Every possible graph matching solution is represented by a different permutation of the array, there are (n+m)! permutations in total. We can also assigned a cost to every possible matching according to the following cost matrix

$$\mathbf{C} = \begin{array}{c} \\ u_1 \\ u_2 \\ \vdots \\ u_n \\ \hline \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{array} \left[\begin{array}{cccc|cccc} v_1 & v_2 & \dots & v_m & \varepsilon_1 & \varepsilon_2 & \dots & \varepsilon_n \\ \hline c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\ c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\ c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\ \hline c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\ \infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\ \infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0 \end{array}\right]$$

Let's call A the adjacency matrix of the first graph and B the adjacency matrix of the second graph:

$$\mathbf{A} = \begin{array}{c} \\ 1 \\ \vdots \\ n \\ \hline 1 \\ \vdots \\ m \end{array} \left[\begin{array}{ccc|ccc} 1 & \dots & n & 1 & \dots & m \\ \hline a_{11} & \dots & a_{1n} & \varepsilon & \dots & \varepsilon \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} & \varepsilon & \dots & \varepsilon \\ \hline \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array}\right] \qquad \mathbf{B} = \begin{array}{c} \\ 1 \\ \vdots \\ m \\ \hline 1 \\ \vdots \\ n \end{array} \left[\begin{array}{ccc|ccc} 1 & \dots & m & 1 & \dots & n \\ \hline b_{11} & \dots & b_{1m} & \varepsilon & \dots & \varepsilon \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mm} & \varepsilon & \dots & \varepsilon \\ \hline \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \varepsilon & \dots & \varepsilon & \varepsilon & \dots & \varepsilon \end{array}\right]$$

The permutation that represents the optimal solution to the graph matching problem can be find optimizing the following function:

$$(\varphi_1, \dots, \varphi_{(n+m)}) = \underset{(\varphi_1,\dots,\varphi_{(n+m)}) \in \mathcal{S}_{(n+m)}}{\arg\min} \left[\sum_{i=1}^{n+m} c_{i\varphi_i} + \sum_{i=1}^{n+m} \sum_{j=1}^{n+m} c(a_{ij} \rightarrow b_{\varphi_i \varphi_j}) \right]$$

The cost of the optimal permutation is the same value of the graph edit distance.

The quadratic assignment problem is a NP-complete problem so even though in theory it represents a correct way to find the optimal solution, in practice it can't be used to solve the graph matching problem for application of large size. For this reason we accept the fact that the optimal solution can't be always found in a linear time and reformulate the graph edit distance problem as a Linear Sum Assignment Problem. Using this type of formalism we are not sure to find the optimal solution but we can implement this method to solve real applications. To do this we exclude from the optimization the quadratic term.

We can summarize the reformulation of the graph matching problem according to the following steps:

Algorithm 2 BP-GED(g_1, g_2)

1: Build cost matrix $\mathbf{C}^* = (c^*_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}^*$
3: Complete edit path according to ψ and **return** $d_\psi(g_1, g_2)$ and/or $d'_\psi(g_1, g_2)$

Where:

- In the first step we build a cost matrix C* that is an alteration of the matrix C as such the matrix C* adds a cost depending of the local structure of the edges.
- In the second step we use the LSAP optimization to find the permutation with the least cost.
- In the third step an approximate graph edit distance is derived from the assignment of step 2.

We denote this graph edit distance approximation algorithm with BP-GED (Bipartite Graph Edit Distance).

During the class we have investigated the possibility of improving the BP-GED using a Sorted Beam Search. We have analyzed twelve different criteria (six criteria used in ascending order and six in descending order) to sort the individual node edit operations and discussed together about an experimental evaluation of these approach.

For the following section let assume that the node u_i belongs to the first graph and that the node $v_{\varphi i}$ belongs to the second graph. Let's indicate the match between the node u_i and the node $v_{\varphi i}$ as an edit operation $(u_i \rightarrow v_{\varphi i}) \in \psi$, where ψ is the set of all possible edit operations.

Each one of the following six criteria can be used two ways depending on whether the edit operations are sorted in descending or ascending order. If we use the ascending order we call the criteria "inverse".

Confident

For a given edit operation $(u_i \rightarrow v_{\varphi i}) \in \psi$ we assign to the node u_i a weight equal to the value stored in the cell (i, φ_i) of the cost matrix.

Unique

For a given edit operation $(u_i \rightarrow v_{\varphi i}) \in \psi$ we assign to the node u_i a weight equal to maximal difference between the cost of the current edit operation $(u_i \rightarrow v_{\varphi i})$, stored in the cell (i, φ_i) of the cost matrix, and the cost of a possible alternative matching node $(u_i \rightarrow v_j)$, stored in the cell (i, j) of the cost matrix.

$$\max_{\forall j=1,\ldots,m} c_{ij} - c_{i\varphi_i}$$

This value is negative if the current edit operations is suboptimal.

Divergent

The idea of this criteria is to prioritize nodes u_i that have a high divergence among all possible node edit costs. For each row of the cost matrix we compute a measurement of divergence according to the following formula:

$$\sum_{j=1}^{m-1} \sum_{k=j+1}^{m} |c_{ij} - c_{ik}|$$

Leader

This criterion weights the node u_i according to the maximum difference between the minimum cost assignment of the node u_i and the second minimum cost assignment of u_i .

According to the cost matrix, let assume that min_1 is the minimum cost assignment of node u_i and min_2 is the second minimum cost assignment of node u_i. We can define the cost of u_i according to the following formula:

$$\frac{min_2 - min_1}{min_2}$$

Interval

For every row i ∈ [1,n] of the cost matrix we define the interval δ_{ri} as the difference between the maximum and the minimum value in the row i. Then for every column j ∈ [1,m] of the cost matrix we define the interval δ_{cj} as the difference between the maximum and the minimum value in the column j.

After that we can define the mean value $\bar{\delta}_r$ among the intervals δ_{ri} with i ∈ [1,n] and the mean value $\bar{\delta}_c$ among the intervals δ_{cj} with j∈ [1,m].

The weight assigned to a given edit operation $(u_i \rightarrow v_{\varphi i}) \in \psi$ is then:

$$(u_i \rightarrow v_{\varphi i}) = \begin{cases} 1 \; if \; \delta_{ri} > \bar{\delta}_r \; and \; \delta_{c_{\varphi i}} > \bar{\delta}_c \\ 0 \; if \; \delta_{ri} < \bar{\delta}_r \; and \; \delta_{c_{\varphi i}} < \bar{\delta}_c \\ 0.5 \; otherwise \end{cases}$$

Deviation

For each row i ∈ [1,n] and each column j∈ [1,m] of the cost matrix we compute the means θ_{ri} and θ_{cj} and the deviations σ_{ri} and σ_{cj} of all entries in the corresponding row and column. Then, for each edit operation $(u_i \rightarrow v_{\varphi i}) \in \psi$ we compute its corresponding weight according to the following set of rules:

- Initially, the weight for an edit operation $(u_i \rightarrow v_{\varphi_i}) \in \psi$ is 0.
- If $c_{i\varphi_i} < \overline{\theta}_{r_i} - \overline{\sigma}_{r_i}$, we add 0.25 to the weight and compute the total number p of assignments in row i that also fulfill this condition and add $0.5/p$ to the weight.
- Repeat the previous step for column $j = \varphi_i$ using $\overline{\theta}_{c_{\varphi_i}}$ and $\overline{\sigma}_{c_{\varphi_i}}$.

To compare the different sorting criteria we have analyzed the results of an experimentation in which all the twelve sorting criteria have been evaluated on different type of data sets. Eight date sets with graphs have been used representing: molecular compounds, fingerprint images, distorted letter drawings symbols from architectural and electronic drawing.

The beam search procedure have been repeated ten times with random permutations of the node assignments in ψ. For every date set and for every sorting criteria it have been measured the mean relative overestimation Øo (in percentage) with respect to the exact graph edit distance. In the experimentation have been also measured how the different reorder strategies increase the mean run time compared to the original framework BP-GED and its extended version BP-Beam.

From the experimentation we can conclude that using sorting criteria we can dramatically reduce the overestimation of the BP-GED.

We can also observe that the Deviation-Inverse sorting criteria is in the top three on all data sets but on Alkane.

Questions

Q We have observed that if we iteratively combine the strategies with each other starting with individually best sorting criterion and then combines the best criterion with the second best criterion so on, we can reduce the overestimation of BP-GED. We have also observed that using four or more sorting criteria at once flatten out the relative gain of distance accuracy.
Is it correct to assume that for future works we should take in account just the first three iteration?

A No, we can't do generalization, we are not sure about the benefits that we can get using these sorting criteria on other date sets. We could get this conclusion just in case that the data set used in our specific application belongs to one the data set classes used in the experimentation.

Q In the data set of the LETTER they uses graphs to represent just fifty letters so they don't represent the entire alphabet and also they don't represent letters that are hard to distinguish like "O" and "0". Furthermore the maximum number of nodes for the data set of the LETTER is equal to nine.
How relevant are the results of the experimentation conducted on this data set ?

A In fact the analyses conducted on this date set is not so relevant. The size of the graph and the structure of the graph is very relevant to conclude general statement from the experimentation.
For example the MUTA data set has a relevant size but it doesn't have a relevant structure, in fact the maximal number of nodes is equal to 417 and the maximal number of edges is equal to 112; it means that the graphs are mostly disconnected so the edges don't play a major role, for this date set we can't do general statements about the edges.
Another example of graphs with specialization are the graphs that belong to the MIVIA data set created by Mario Vento. The particularity of these graphs is that they are generated randomly and we don't know how relevant could be the results of an experimentation on this date set for real applications.

Discussion

- From the results of the experimentation we can conclude for every data set which sorting criteria is the best but we don't know why. It should be interesting to know why a sorting criterium fits better for a fixed type of graph, in this way we could choose the sorting criteria to use in a specific application in according to the type of graph that we have to manage.

- In the beginning of our discussion we started with the idea to use the A* search algorithm to find the better solution to the graph matching problem. This approach has the problem that doesn't find the optimal solution in optimal time. The A* search algorithm uses a heuristic to speed up the elaboration. If cert constraints on the heuristic are satisfied we are sure to find the optimal solution also with the A* algorithm with a time that depends of how good is the heuristic. The sorting criteria approach is similar to the A* search because in both the cases we try to consider a short path among all the possible solutions but we don't have to impose any constraint.

  The question is if there exists any constraint that we can impose on the structure of the graph or on the cost function to guarantee that the optimal solution will always be found (as we do with the A* algorithm).
  - For example we know that if there is an ordering on the labels of the nodes we can solve the graph matching problem in a linear time: sort the nodes of the two graphs according to the labels and check the correspondences.
  - Another example is the following: if we impose as constraint that we have an exponential space available we can sort the nodes in a linear time and fine the correspondences in an easy way.

- If we focus on a particular family of graphs (like a tree structure) can we define if there are some benefits from the point of view of the different sorting criteria?

PRESENTATION REPORT

Improving the Distance Accuracy of Bipartite Graph Edit Distance

Selected Chapters in Image Processing SS 2016

Carmine Sansone (1529804)

April 28, 2016

Introduction

In the previous presentation it have been explained how to reformulate the graph edit distance problem as an instance of a Quadratic Assignment Problem. With this formulation of the problem the optimal solution cannot be always found because a QAP is a NP-complete problem. For this reason a new formulation of the problem has been considered. The QAP is reduced to a Linear Sum Assignment Problem (LSAP). The solution found using the LSAP formulation is not always the optimal solution to the original problem but can be computed to solve practical problems. The graph edit distance approximation algorithm is denoted as Bipartite Graph Edit Distance (BP-GED). The Figure 1 define the pseudo-code of the BP-GED algorithm. More details about this algorithm and the notation used can be found in the previous discussion report.

BP-GED(g_1, g_2)

1: Build cost matrix $\mathbf{C}^* = (c^*_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}^*$
3: Complete edit path according to ψ and **return** $d_\psi(g_1, g_2)$ and/or $d'_\psi(g_1, g_2)$

Figure 1: Pseudo-code of Bipartite Graph Edit Distance algorithm

One of the major problems of the approximation framework BP-GED is that it over- or underestimates the true edit distance quite often. The goal of this presentation is to describe some alternatives to this algorithm that can better approximate the cost of the assignment to the cost of the optimal solution. These algorithms can be classified in two types in according to the strategy used: improvements via search strategies and Improvements via integration of node centrality information.

1 Improvements via Search Strategies

From the evaluation done can be observed that the BP-GED algorithm return a sub-optimal solution to the original problem because of just few incorrect assignments present in the solution found. The idea is to modify the sub-optimal solution found with the BP-GED trying to correct these incorrect assignments. An exhaustive search cannot be done because there are *(n+m)!* different possible permutations of the solution found with the BP-GED, where *n* is the number of the nodes of the first graph and *m* is the number of the nodes of the second graph.

1.1 Iterative Search

Starting with the solution found with the BP-GED ψ, the idea is to use execute a series of iterations in which in every iteration the node operations in ψ with highest implied edge costs is prevented from the edit path solution and a new solution is computed. Using a greedy approach in every iteration a node operation is selected and excluded for all the following iterations. Iteration by iteration just the most promising node operations survive to this process and are analyzed to find a better solution.

The Figure 2 describes the pseudo-code of the iterative search algorithm.

Algorithm 3 BP-Iterative (g_1, g_2) (Meta Parameter: q)

1: Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}$
3: $d_{best} = d_{\psi}(g_1, g_2)$
4: $i = 0$
5: **while** $i < q$ **do**
6: $i++$
7: Determine node operation $(u_i \rightarrow v_{\varphi_i}) \in \psi$ with highest implied edge edit cost
8: Modify $\mathbf{C}$ by setting $c_{i\varphi_i} = \infty$ (prevent $(u_i \rightarrow v_{\varphi_i})$ for future solutions)
9: Compute optimal node assignment ψ' on modified cost matrix $\mathbf{C}$ and set $\psi = \psi'$
10: **if** $d_{\psi'}(g_1, g_2) < d_{best}$ **then**
11: $d_{best} = d_{\psi'}(g_1, g_2)$
12: **end if**
13: **end while**
14: **return** d_{best}

Figure 2: Pseudo-code of the iterative search algorithm

The first three steps are the same of the BP-GED. The line five define is the start of the iteration process. The number of the iteration is defined by q that is a input parameter of the algorithm. In every iteration, one particular cost entry $c_{i_{\phi i}}$, associate to the node operations $(u_i \rightarrow v_{\phi i}) \in \psi$ with high implied edge costs, is set to ∞ such that the corresponding node edit operation cannot occur in the next assignments.

1.2 Floating Search

In the iterative search once a modification of the form $c_{i_{\phi i}} = \infty$ has been conducted, the corresponding node operation $(u_i \rightarrow v_{\phi i})$ is lost for the remainder of the search procedure. The floating search is based on the following idea: for every iteration restore all the elements in the cost matrix that have been set to ∞ in the previous iteration to the original value if this restoration can improve the solution found. Every iteration consist in a step in forward and some steps in backward where:

- A forward step means: set the cost entry $c_{i_{\phi i}}$ associate to the node operations $(u_i \rightarrow v_{\phi i}) \in \psi$ with high implied edge costs to ∞;
- Backward steps (as long as the resulting solutions can be improved) means: resets the cost entries $c_{i_{\phi i}}$ to their original cost value;

The Figure 3 describes the pseudo-code of the floating search algorithm. This is a description of the variables used in the pseudo-code:

- i is the number of foreword steps (and it is also the number of the entry of the cost matrix set to ∞);
- j is the number of backward step for a fixed forward step (j ≤ i);
- d[0..q]: in d[i] is stored the cost of the solution found in the step i and in d[0] is stored the original distance approximation of BP-GED;
- F is a structure where the node operations that we have prevented are stored.

Algorithm 4 BP-Floating (g_1, g_2) (Meta Parameter: q)

1: Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \to v_{\varphi 1}, \ldots, u_{m+n} \to v_{\varphi m+n}\}$ on $\mathbf{C}$
3: Initialize empty array for buffering $(q+1)$ distance values $d[0 \ldots q]$
4: $d[0] = d_\psi(g_1, g_2)$
5: Initialize empty list for buffering prevented edit operations $f = \{\}$
6: $i = 1$
7: **while** $i \leq q$ **do**
8: Determine node operation $(u_k \to v_{\varphi k}) \in \psi$ with highest implied edge edit cost
9: Modify $\mathbf{C}$ by setting $c_{k\varphi k} = \infty$ (prevent $(u_k \to v_{\varphi k})$ for future solutions)
10: Compute optimal node assignment ψ' on modified cost matrix $\mathbf{C}$
11: $d[i] = d_{\psi'}(g_1, g_2)$ (buffer approximation value with i preventions)
12: $f = f \cup \{(k, \varphi k)\}$ (buffer the indices of the prevented node operation)
13: *removed = true*; *j=0*; $\xi_1 = (-), \xi_2 = (-)$
14: **while** *removed* **do**
15: *removed = false*; $d_r = \infty$; $j{+}{+}$
16: **for** all indices pairs $(k, \varphi k) \in f$ **do**
17: Modify $\mathbf{C}$ by resetting $c_{k\varphi k}$ to the original cost entry
18: Compute optimal node assignment ψ' on modified cost matrix
19: **if** $d_{\psi'}(g_1, g_2) < d_r$ **then**
20: $d_r = d_{\psi'}$
21: $\xi_1 = (k), \xi_2 = (\varphi k)$
22: **end if**
23: Modify $\mathbf{C}$ by setting $c_{k\varphi k} = \infty$
24: **end for**
25: **if** $d_r < d[i-j]$ **then**
26: $d[i-j] = d_r$
27: $f = f - \{(\xi_1, \xi_2)\}$
28: *removed = true*;
29: Modify $\mathbf{C}$ by resetting $c_{\xi_1 \xi_2}$ to the original cost entry
30: ψ = optimal node assignment on $\mathbf{C}$
31: **else**
32: $j{-}{-}$
33: **end if**
34: **end while**
35: $i = i - j$
36: **end while**
37: **return** $\min_{1,\ldots,q} d[i]$

Figure 3: Pseudo-code of the floating search algorithm

Until the line 10 the algorithm is almost the same that the iterative search. At the line 11 and 12 the data structures d and F are updated. At line 14 the backward steps begin. Inside this circle all the prevented node operations are restored to the original value one by one, and for every one of them a new solution is computed. If during this circle a better solution than the current one is found a backward step is done and the elaboration restart from the line 14. The best solution among the solution found in every iteration is returned as final result.

1.3 Genetic Search

Every possible variation of the original node assignment ψ (obtained with the BP-GED) is represented by a chromosome. Every chromosome has performance measurement called fitness. The fitness chromosome is inverse proportional to the cost of the solution represented by the chromosome so the lower $d_{\psi_k^0}$ (g1, g2) is, the better is the fitness of chromosome ψ_k^0.

An initial population of chromosomes is defined with N-1 random variations of the original assignment ψ plus the original assignment itself.

The idea is to create a population of solutions, every solution is a permutation of the solution found with the BP-GED. The created population evolves in new generations of populations. Generation by generation the exemplars of the populations should increase their fitness until the optimal solution (with the optimal fitness) is found. There is no proof on the convergence of the genetic algorithms so a limitation on the number of the generable population have to be defined. In the continue of this paragraph this process is explained in more details.

Initial population

Let P(0)= $\{\psi_1^0, \psi_2^0, \ldots, \psi_N^0\}$ be the notation used to describe the initial population also called population at the iteration zero.

Every alternative assignment ψ_k^0 with k∈[1,..,N-1] prevents one or more the edit operations $(u_i \rightarrow v_{\phi i}) \in \psi$ setting to ∞ the corresponding entries $c_{i_{\phi i}}$ of the cost matrix. The alteration of the original assignment is done by chance: every edit operation has a probability p to be prohibited. This probability is called mutation probability.

Evolution

Given a population P(t) a new population P(t + 1) of assignments is built upon a subset E ⊂ P(t), referred as parents. The parents are the f*N chromosomes with the best finest value, with f ∈ [0,1]. The parent are added to the new population P(t+1). In this way the best solution found will be not lost passing from the population P(t) to the population P(t+1). To create the remained N − |E| exemplars of the population P(t+1) the following procedure is repeated N − |E| times:

> Two assignments, ψ' and ψ'' are randomly selected from the set of parents E and combined to one assignment. Let C' and C'' be the cost matrix corresponding to the assignments ψ' and ψ''. A new cost matrix is computed in this way:
>
> $$\mathbf{C}_m = (\max\{c'_{ij}, c''_{ij}\}).$$
>
> Based on the new cost matrix C_m the assignment ψ''' is computed and added to P(t+1).

The Figure 4 shows the pseudo-code of the genetic search algorithm.

Algorithm 5 BP-GA (g_1, g_2) (Meta Parameters: N, p, δ, f, s)

Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs g_1 and g_2
Compute optimal node assignment $\psi = \{u_1 \to v_{\varphi_1}, \ldots, u_{m+n} \to v_{\varphi_{m+n}}\}$ on $\mathbf{C}$
$d_{best} = d_\psi(g_1, g_2)$
for $i = 1, \ldots, s$ **do**
  build $P(0) = \{\psi, \psi_1^{(0)}, \ldots, \psi_{N-1}^{(0)}\}$ based on ψ using mutation probability p
  $t = 0; l = 0$
  while $t - l < \delta$ **do**
    select a subset $E \subseteq P(t)$ of parents $(|E| = f \cdot N)$
    build a new population $P(t+1) = E \cup \{\psi_1^{(t+1)}, \ldots, \psi_{N-|E|}^{(t+1)}\}$ from E
    $d = \min_{i=1,\ldots,N}\{d_{\psi_i^{(t+1)}}(g_1, g_2)\}$
    $t = t + 1$
    if $d < d_{best}$ **then**
      $d_{best} = d; l = t$
    end if
  end while
end for
return d_{best}

Figure 4: Pseudo-code of the genetic search algorithm

Computational complexity problem

The evolution process continues until the best distance approximation has not been improved during the last δ iterations. It is well known that genetic algorithms are not deterministic. Therefore, one might repeat the complete search procedure *s* times from scratch and return the overall best approximation found in these *s* runs.

For every one of the s runs in average t population are generated and for every population the two main steps of the original approximation framework BP-GED are computed N times. This extended framework increases the run time by the magnitude of (s · t · N) compared to BP-GED.

1.4 Greedy Search

In the algorithms described in the previous paragraphs, alternated versions of the original assignment ψ are obtained modifying the cost matrix setting some entry to infinite and revaluating the BP-GED with the new cost matrix. With this approach the original assignment ψ is varied by means of pairwise swaps of node assignments.

The greedy search is similar to the iterative search, both of them have a greedy behavior. In contrast with BP-Iterative this search variant cannot be generalized using a floating search strategy. This is because two forward steps with pairwise swaps might not necessarily be independent from each other; for this reason backward steps cannot be done. The Figure 5 describes the pseudo-code of the greedy search algorithm.

Algorithm 6 BP-Greedy-Swap (g_1, g_2) (Meta Parameter: θ)

1: Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}$
3: $d_{best} = d_{\psi}(g_1, g_2)$
4: *swapped* = *true*
5: **while** *swapped* **do**
6: *swapped* = *false*
7: **for** $i = 1, \ldots, (m + n - 1)$ **do**
8: **for** $j = i + 1, \ldots, (m + n)$ **do**
9: $cost_{orig} = c_{i\varphi_i} + c_{j\varphi_j}$
10: $cost_{swap} = c_{i\varphi_j} + c_{j\varphi_i}$
11: **if** $|cost_{orig} - cost_{swap}| \leq \theta \cdot cost_{orig}$ **then**
12: $\psi' = \psi \setminus \{u_i \rightarrow v_{\varphi_i}, u_j \rightarrow v_{\varphi_j}\} \cup \{u_i \rightarrow v_{\varphi_j}, u_j \rightarrow v_{\varphi_i}\}$
13: Derive approximate edit distance $d_{\psi'}(g_1, g_2)$
14: **if** $d_{\psi'}(g_1, g_2) < d_{best}$ **then**
15: $d_{best} = d_{\psi'}(g_1, g_2)$
16: *best-swap* = $\{i, \varphi_j, j, \varphi_i\}$
17: *swapped* = *true*
18: **end if**
19: **end if**
20: **end for**
21: **end for**
22: **if** *swapped* **then**
23: update ψ according to *best-swap*
24: **end if**
25: **end while**
26: **return** d_{best}

Figure 5: Pseudo-code of the greedy search algorithm

Until the line 4 the instructions are almost the same as in the previous algorithms. At the lines 7 and 8 two cycles start, the first run on the index i ∈ [1,..,m+n-1] and the second one run on the index j ∈ [i,..,m+n]. The Figure 6 shows an example of elaboration.

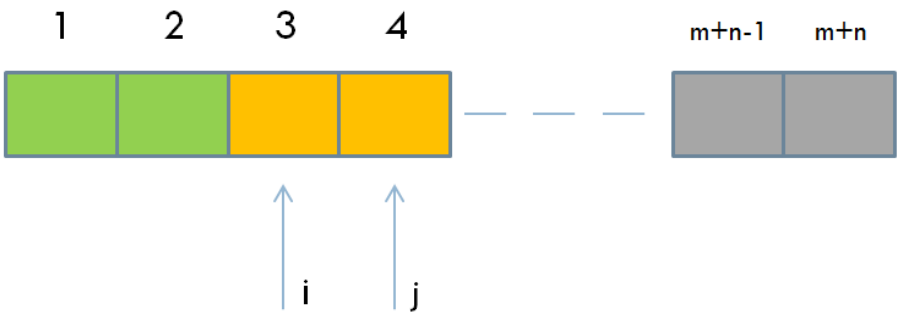


Figure 6: Example of greedy search algorithm

It can be observed that the array that represents the solution is divided in the following way: the element behind the index i (green cells) are the elements of the array that have been elaborated. The elements of the array referenced by the indexes i and j are the elements under analysis (yellow cells). The elements of the array drew in gray represent the elements not analyzed jet.

At line 10 the swap is done. At line 11 the new solution obtained with the swap is analyzed: if the cost of this new node operation is similar to the cost of the original node operation for less than a threshold means that the considerate swap is promising. In this case the cost of the complete solution with the new node operation is evaluated. If the cost of the new solution found is better than the current one, the swap is confirmed.

The threshold on the node operation cost is done using this threshold value: $\delta * c_{original}$ where $\delta \in (0,1]$ is an input of the algorithm. If δ is high more possible combinations of swaps are taken in account but the computational cost would be more high. If δ is low just the most promising swaps are considered.

1.5 Genetic Search with Swap Strategy

The genetic algorithm described in the paragraph 1.3 can be improved using a greedy approach to build the initial population P(0) and to create the population P(t+1) given the population P(t).

Initial population

Instead of create the chromosomes of the initial population alternating the node operations of the initial node assignment ψ using the mutation probability p, the alternated versions of the original node assignment ψ are created using pairwise swaps in according to the greedy search. The Figure 7 describes how the initial population is built.

Algorithm 7 Build-Initial-Pool (ψ) (Meta Parameter: $\mathbf{C} = (c_{ij})$, N, p)

Initialize $P(0) = \{\psi\}$
while $P(0) < N$ **do**
  for $i = 1, \ldots, (m + n - 1)$ **do**
    for $j = i + 1, \ldots, (m + n)$ **do**
      $cost_{orig} = c_{i\varphi_i} + c_{j\varphi_j}$
      $cost_{swap} = c_{i\varphi_j} + c_{j\varphi_i}$
      if $|cost_{orig} - cost_{swap}| \leq \theta \cdot cost_{orig}$ **then**
        Generate random number $r \in [0, 1]$
        if $r < p$ **then**
          $\psi' = \psi \setminus \{u_i \to v_{\varphi_i}, u_j \to v_{\varphi_j}\} \cup \{u_i \to v_{\varphi_j}, u_j \to v_{\varphi_i}\}$
          $P(0) = P(0) \cup \{\psi'\}$
          $i = (m + n)$; $j = (m + n + 1)$
        end if
      end if
    end for
  end for
end while
return $P(0)$

Figure 7: Creation of the initial population using the greedy search approach

Evolution

Given the population P(t), population P(t+1) is built in according to the following steps:

- All approximations from the set of parents E ⊂ P(t) are added without any modification to the next population P(t + 1)

- The remaining N − |E| chromosomes of the population P(t+1) are creating repeating the following procedure N − |E| times:

  One single assignment ψ ∈ E is randomly selected. Using ψ, the subroutine described in Figure 7 is carried with N=1. This mutated assignment is added to P(t + 1). The node assignment ψ is altered by at most one additional swap.

1.6 Beam Search

Also in this case the alternated versions of the original node assignment ψ are created doing swaps. To decide which swap has to be done to the current node assignment a tree search strategy is used.

A tree node is defined as triples (ψ, q, dψ) where: ψ is a certain node assignment, q is the depth of the tree node in the search tree, and dψ is the approximate edit distance corresponding to ψ.

The idea is to do a breadth-first search. Let open be the set of unprocessed tree nodes. The nodes in open are sorted in according two criteria: first in according to the deep and second in according to the approximate edit distance. The breadth-first search is preferred to the best-first search (in which the nodes in the set open are sorted in according to the approximate edit distance) because there is no correlation between the deep and the approximate edit distance of a node. In fact in the best-first search the nodes that have a high value of q should represent better solutions than the nodes with a low value of q.

The breadth-first search does a complete search in the solution space, for this reason a limitation on the size of the set open is imposed. In the set open are stored just the best b unprocessed tree nodes where b is an input parameter. This means that only the most promising nodes are expanded. Clearly pruning parts of the search tree it might be that the optimal solution is lost during the search process.

The Figure 8 describes the pseudo-code of the beam search algorithm.

Algorithm 8 BP-Beam (g_1, g_2) (Meta Parameter: b)

1: Build cost matrix $\mathbf{C} = (c_{ij})$ according to the input graphs g_1 and g_2
2: Compute optimal node assignment $\psi = \{u_1 \rightarrow v_{\varphi_1}, \ldots, u_{m+n} \rightarrow v_{\varphi_{m+n}}\}$ on $\mathbf{C}$
3: $d_{best} = d_\psi(g_1, g_2)$
4: Initialize $open = \{(\psi, 0, d_\psi(g_1, g_2))\}$
5: **while** *open* is not empty **do**
6: Remove first tree node in *open*: $(\psi, q, d_\psi(g_1, g_2))$
7: **for** $j = (q+1), \ldots, (m+n)$ **do**
8: $\psi' = \psi \setminus \{u_{q+1} \rightarrow v_{\varphi_{q+1}}, u_j \rightarrow v_{\varphi_j}\} \cup \{u_{q+1} \rightarrow v_{\varphi_j}, u_j \rightarrow v_{\varphi_{q+1}}\}$
9: Derive approximate edit distance $d_{\psi'}(g_1, g_2)$
10: $open = open \cup \{(\psi', q+1, d_{\psi'}(g_1, g_2))\}$
11: **if** $d_{\psi'}(g_1, g_2) < d_{best}$ **then**
12: $d_{best} = d_{\psi'}(g_1, g_2)$
13: **end if**
14: **end for**
15: **while** size of *open* $> b$ **do**
16: Remove tree node with highest approximation value d_ψ from *open*
17: **end while**
18: **end while**
19: **return** d_{best}

Figure 8: Pseudo-code of the beam search algorithm

1.7 Sorted Beam Search

The only difference to BP-Beam is that the original assignment ψ is first reordered according to a specific sorting strategy before BP-Beam is carried out using the assignment ψ´ rather than ψ.

The Figure 9 describes the pseudo-code of the sorted beam search algorithm.

Algorithm 9 SBP-Beam (g_1, g_2) (Meta Parameter: b)

1: $d_\psi(g_1, g_2) = BP - GED(g_1, g_2)$
2: $\psi' = Sort(\psi)$
3: $d_{SortedBeam}(g_1, g_2) = BP\text{-}Beam(g_1, g_2, d_\psi, \psi', b)$
4: **return** $d_{SortedBeam}(g_1, g_2)$

Figure 9: Pseudo-code of the sorted beam search algorithm

1.8 Experimental Evaluation

For the experimental evaluations, five data sets from the IAM graph database repository for graph-based pattern recognition and machine learning are used:

- AIDS and MUTA: molecular compounds
- FP: fingerprints
- LETTER: distorted letter drawings
- GREC: symbols from architectural and electronic drawings

On each data set and for each graph edit distance algorithm discussed in the previous paragraphs the following two performance indexes are computed

- The mean relative overestimation of the exact graph edit distance Ø*o*
- The mean run time Ø*t* compared to the original framework BP-GED.

The exact graph edit distance is computed with the A* algorithm. On MUTA data set the A* algorithm is inapplicable because of lack of memory so the BP-GED is used as reference of optimal. To evaluate the BP-GA just two on the five parameters are varied: population size and mutation probability.

BP-Iterative

As discussed in the previous paragraph, doing an elevate number of iteration the solution obtained with this algorithm is improved. On the AIDS data, for instance, the mean relative overestimation Øo can be reduced from 12.68% to 9.64% with just one additional iteration. This improvement is very high when the number of iteration goes from one to three but after that increasing the number of iteration just a small improvement can be observed.

BP-Floating

With algorithm BP-Floating very similar results to BP-Iterative can be observed. In fact the improvements bring from this algorithm respect the previous one are not so high. It can be observed that the run time of this algorithm is higher than the previous one. If run time is crucial, BP-Iterative is clearly preferable over BP-Floating.

BP-GA

Respect the two previous algorithms, the results obtained with the BP-GA are clearly improved. The Figure 10 shows how good are the results obtained with the BP-GA algorithm on the FP data set. On the picture (a) the results obtained with the BP-GED are plotted and on the picture (b) the results obtained with the BP-GA are plotted. Yet, this improvement is accompanied by increase the mean run time.

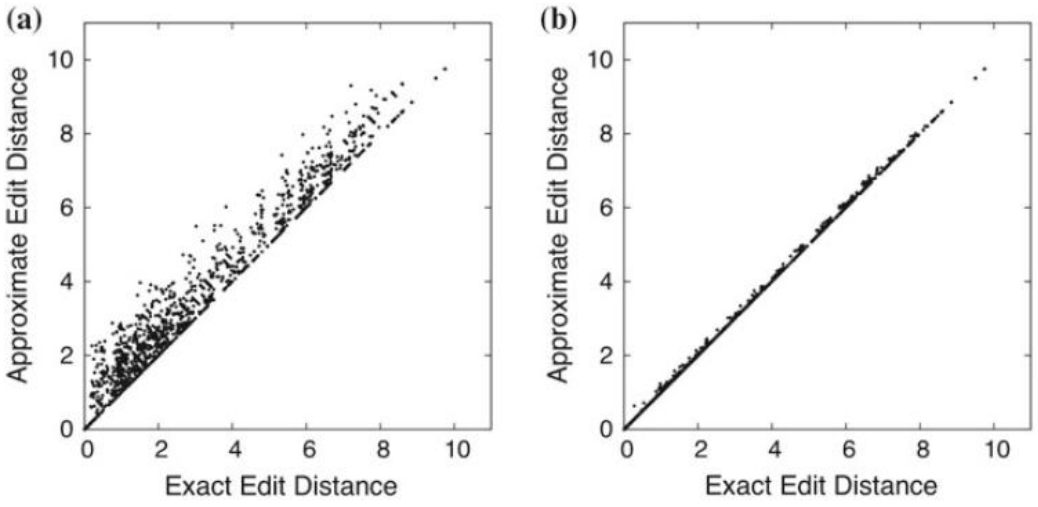


Figure 10: Comparison of the results obtained on the FP data set with BP-GED (a) and with BP-GA (b)

BP-GA-Swap

Compared to BP-GA a clear speed-up can be observed on all data sets. It can also be observed that the BP-GA-Swap does not reach the same level of distance accuracy as BP-GA.

BP-Greedy-Swap

It can be observed that using the greedy search in combination with the swap strategy clearly improve the run time and also the overestimation of the exact graph edit distance. The parameter θ has to be defined considering a trade-off between the overestimation of the exact graph edit distance and the run time.

BP-Beam

As expected the performance of the BP-Beam are strictly correlated to the input parameter b. It can be observed that doubles the b value doubles the run time but the overestimation can is reduced.

2 Improvements via Integration of Node Centrality Information

Rather than applying a post-processing search procedure to the assignment ψ the topological information of individual nodes is exploited in order to achieve a better approximation of the true edit distance.

The idea is to assign a measure of importance to nodes u ∈ V according to the topology of their surrounding nodes.

- Degree Centrality
  The degree centrality λ_i^{deg} of a node $u_i \in V$ is defined as the number of edges connected to u

- Eigenvector centrality
  The eigenvector centrality λ_i^{eig} of a node $u_i \in V$ is defined in the following way:

$$\lambda_i^{\langle eig \rangle} = \kappa_1^{-1} \sum_j a_{ij} \lambda_j$$

  a_i is an element of the adjacency matrix A of the graph and k_1 is the largest eigenvalue of A and the summary is done on all the neighboring of the node u_i. The eigenvector centrality is different from the degree centrality because in this case not all the neighboring of a node $u_i \in V$ have the same importance. The eigenvector centrality can be high because a node has many neighbors or because it has important neighbors (or both).

- Page rank centrality
  The eigenvector centrality λ_i^{pr} of a node $u_i \in V$ is defined in the following way:

$$\lambda_i^{\langle pr \rangle} = \alpha \sum_j a_{ij} \frac{\lambda_j}{\hat{k}_j} + \delta$$

$$\hat{k}_j = \max(1, k_j)$$

  $\hat{k}_j$ is a slight variant of the node's degree k_j, it is defined as showed above to avoid division for zero. Compared to eigenvector centrality, the major differences are the division by the degree and the free parameter α. The value α has been set to 0.85 as Google does but there is no theory reason about this choice.

Chosen a node centrality measurement the cost matrix is modified in according to the following formula:

$$\mathbf{C}' = (\beta \cdot c_{ij} + (1 - \beta) \cdot |\lambda_i - \lambda_j|)$$

where β ∈(0, 1) corresponds to a weighting parameter that balances the influence of the original cost c_{ij} and the centrality measurement. In the experimental evaluation presented in it turns out that using the enriched cost matrix C rather than the original matrix C, in general the distance approximation does not decrease the overall overestimation.

The idea is to build *t* altered cost matrix using *t* different topology algorithms. The minimum value of all *t* distance approximations computed on all the *t* different versions of the C matrix is returned as approximation value.

2.1 Experimental Evaluation

For experimental evaluations, three data sets from the IAM graph database repository for graph-based pattern recognition and machine learning are used:

- AIDS: molecular compounds;
- FP: fingerprints;
- GREC: symbols from architectural and electronic drawings;
-

On each data set and for each graph edit distance algorithm discussed two performance indexes have been defined:

- The mean relative overestimation of the exact graph edit distance Ø*o*;
- The mean run time Ø*t* compared to the original framework BP-GED;

The three strategies described in the previous paragraph: degree centrality, eigenvector centrality and page rank centrality have been evaluated changing the parameter β. The used values of the parameter β are: 0.1, 0.3, 0.5, 0.7 and 0.9. Also four possible combinations of the three centrality measure have been tested. It can be observed that parameter β has negligible influence on the run time behavior.

Concerning the overestimation it can be observed that on all the data sets the page rank centrality measure has got the best result. With the combination of the three centrality measures the mean relative overestimation can be further decreased. As expect, the best results are achieved when all three centrality measures are combined.

Improving the Distance Accuracy of Bipartite Graph Edit Distance: Minutes

Selected Chapters in Image Processing SS 2016

Max Langer

May 23, 2016

1 Opponent Questions

1.1 Iterative Search: For a specific class of the problem, how can i find the optimal value of the q?

The q parameter does not depend on the class of the problem. More iterations give better results. The question is how long we can wait to obtain the result.

I $q \to \infty$, is it the same as A*? This can be answered with no, because the iterative search does not use backtracking, so the tree of the optimal solution can be rejected in an earlier iterations.

q is limited on the number of operations.

1.2 Bi-directional search: Can we use this type of search for improving the distance accuracy of bipartite Graph Edit Distance?

Bi-directional search Alternate searching from the start state toward the goal and from the goal state toward the start. Stop when the frontiers intersect. Can (sometimes)lead to a solution more quickly.

The author does not use it, maybe it works well, but: You have to know two ends (know the goal) to do two half searches and have to assume the same cost for both directions.

1.3 Genetic algorithm

The genetic search is not deterministic. More chromosomes will need longer, but the chances are greater to get a better solution. Again it is a time versus quality problem.

If we have apriori information, the genetic algorithm parameters can be tuned, but in general this is not possible.

1.4 Beam Search: Can we use a different strategy to reduce the number of partial solution to be processed like Procut or Multi-Procut?

The basic idea of beam search is that only a fixed number b of (partial) solution to be processed are kept in open at any time.

An huristic has to be defined. Here we use a trick for cut away: Take the most promising partial solutions and cut away the others. Alpha beta pruning yields better solutions, but needs longer. We go to the end of the tree and back again.

2 General Discussion

2.1 If we restrict the backtracking in A* to a fixed number, we decrease the complexity.

One can use Tabu-search and skip already looked at branches for a time. In general there exist many equally good solutions and it is not worth to try all.

Datasets used to try these approaches: small graphs, graphs with few edges. Are the results the same for big connected graphs or other test data?

2.2 All algorithms are purely sequential. Because all operations are local, how important is it whether you use a specific order? Are there parallel strategies.

The main idea is to change one part and then another part.

Each modification generates new graph with cost c. Operations are independent of each other in some strategies.

With an genetic algorithm we need a parent.

Can we partition graph in set of subgraphs (and donate them to new vertices)? Is a matching of those smaller graphs, also matching of the larger graph. Can we find the optimal solution that way? Use subgraphs that belong to each other. In pyramid structure one can also preserve local properties.

A possible approach: Create pyramid, compare in higher lever and step down for solution refinement.

Learning Exact Graph Edit Distance: Minutes

Selected Chapters in Image Processing SS 2016

Daniel Pucher

June 27, 2016

The chapter 5 of KR has been presented only orally by Domenico Verlotta who was unable to provide a proper scientific summary for these proceedings. The opponent was Max Langer.

1 Opponent Questions

1.1 Can we learn an exact Graph Edit Distance? When not, can we predict a reasonable value for ϵ or control learning to achieve a certain ϵ?

With SVMs the data is clustered and close clusters make the decision hard. So maybe SVMs are not the best way to do this and other ways are better suited in some cases.

1.2 How does it come, that there are evaluation results that are worse than both bounds of BP-GED?

They don't have to stay in the bound, since the bounds are the distances from the optimal. To denote these distances as "Bounds" is a bit misleading.

1.3 Does it make sense to dismiss Node Edit Operations that are considered not to be in the exact GED and retry other configurations. Would this approach be reasonable?

To some extend yes, because if an edit operation is already bad the edit path might not be good as a whole. A problem of dismissing might occur if the system favours some edit operations over others. It could happen that edit operations that are favoured by the system lead to bad edit paths.

2 General Discussion

2.1 What happens if more than one sequence of edit operations are optimal?

An example:
Let $\Psi = \{x_1, x_2, ..., x_n\}$ be a complete (yet not necessarily optimal) edit path, where edit operations are changed to find the optimal path.

And let $\lambda_1 = \{a_1, a_2, ..., a_n\}$ and $\lambda_2 = \{b_1, b_2, ..., b_n\}$ be two optimal paths that are equivalent except for the first edit operation.

If $x_1 = b_1$ than λ_1 b_1 is classified as incorrect. To solve this problem, both optimal paths have to be taken into account.

2.2 Comments

Sequences who are equivalent to other sequences, transform one graph into another and give the same results. Assuming that the validation through the features gives the same values to all the features, only one sequence needs to be trained. For this, the edit operations would need to be validated together and not separately.

No proof for the chosen features is given and the features are based on the C matrix and the costs of the edit operations.

Speakers of Selected Chapters organized[1] since 2002:

SS 2002 AKdTI5: Walter Kropatsch: Anwendungen von Bildpyramiden

SS 2003(BV): Walter Kropatsch: BILDPYRAMIDEN + GRAPHEN

WS 2003(ME): P. Lienhardt (Poitiers): Fundamentals of Topology-based Geometric Modeling.

SS 2004(BV): Wolfgang Förstner (Bonn): Projektive Geometrie

WS 2004(ME): Walter Kropatsch: Cognitive Vision

[1] In the above SS stands for summer term and WS for winter term, BV stands for image processing and ME for pattern recognition.

SS 2005(BV): Walter Kropatsch: Repräsentationen in der Bildanalyse

WS 2005(ME): Nicu Sebe (Amsterdam): Multimedia Information Systems

WS 2006(ME): Samuel Peltier (Poitiers): Homology Groups (canceled)

SS 2007(BV): Eric Andres (Poitiers): discrete Geometry

WS 2007(ME): Walter Kropatsch: GRAPHS + Pyramids

SS 2008(BV): R. Gonzalez-Diaz (Sevilla): Extracting Topological Information of 3D Digital Images

WS 2008(ME): Kropatsch, Helena Molina (Sevilla): Pyramids + Topology

SS 2009(BV): Pedro Real Jurado (Sevilla): Computing "holes" of 3D digital objects

WS 2009(ME): Luc Brun (Caen): Partition encoding: Geometrical and topological challenges

SS 2010(BV): Walter Kropatsch: We are building a Topological Pyramid and Rocio Gonzalez-Diaz (Sevilla): (Co-)Homology Groups of 3D binary images

WS 2010(ME): Kropatsch, Vucini, Chao Chen: Pyramids + Topology

SS 2011(BV): Horst Bunke (Bern): Basic Methodology and Recent Developments in Structural Pattern Recognition

WS 2011(ME): Claudia Landi (Reggio Emilia, I): Shape-from-function methods

SS 2012(BV): Max Göbel and Walter Kropatsch: Object Detection/Recognition from 2D images

WS 2012(ME): KSFu Lecture Series: Pavlidis, Aggarwal, Huang, Kittler, Jain, Bunke

SS 2013(BV): Walter Kropatsch, GbR2013: Graph-based Representations in PR

WS 2013(ME): KSFu Lecture Series: Pavlidis, Aggarwal, Huang, Kittler, Jain, Bunke, Chellappa

SS 2014(BV): W. Kropatsch, Thomas Druml (VetMed), Wolfgang Busch (GMI): Image-based Phenotyping

WS 2014(ME): Walter Kropatsch: Selection of KSFu and BMVC Lectures

SS 2015(BV): Laszlo Nyul: Fuzzy techniques in image processing

WS 2015(ME): Walter Kropatsch, Nicole Artner, Ines Janusch, Aysylu Gabdulkhakova: Selection of PRIP research topics 2015/16

SS 2016(BV): Walter Kropatsch, (Kaspar Riesen): Graphs: Matching and Distance

References

[1] Kaspar Riesen. *Structural Pattern Recognition with Graph Edit Distance, Approximation Algorithms and Applications.* Advances in Computer Vision and Pattern Recognition. Springer International, 2015.