UNIVERSITAT ROVIRA I VIRGIL

GRAPH EDIT DISTANCE APPLIED TO DIVERSE FRAMEWORKS: LEARNING, MATCHING AND EXPLORING TECHNIQUES

María Elena Rica Alarcón

María Elena Rica Alarcón

Graph Edit Distance applied to diverse frameworks: Learning, matching and exploring techniques

DOCTORAL THESIS

UNIVERSITAT ROVIRA i VIRGILI

Tarragona, Spain

2022

Graph Edit Distance applied to diverse frameworks: Learning, matching and exploring techniques

DOCTORAL THESIS

Supervised by

Dr. Francesc Serratosa

Dra. Susana Álvarez

Departament d'Enginyeria Informàtica i Matemàtiques

UNIVERSITAT ROVIRA i VIRGILI

Tarragona, Spain

2022

UNIVERSITAT
ROVIRA i VIRGILI

Escola Tècnica Superior d'Enginyeria
Departament d'Enginyeria Informàtica i Matemàtiques
Avinguda dels Països Catalans, 26
43007 Tarragona (Spain)
Tel. +34 977 559 703 `https://deim.urv.cat/`

In Tarragona, July, 2022.

I STATE that the present study, entitled "Graph Edit Distance applied to diverse frameworks: Learning, matching and exploring techniques", presented by Mrs. María Elena Rica Alarcón for the award of the degree of Doctor, has been carried out under my supervision at the Department d'Enginyeria Informàtica i Matemàtiques of this university.

Signed by the doctoral thesis supervisors:

[illegible] [illegible]

Dr. Francesc Serratosa Dra. Susana Álvarez

Acknowledgements

I want to thank my supervisors Prof. Francesc Serratosa and Prof. Susana Alvarez for giving me the opportunity to work with them during these years. The conversations with them have been truly enriching and without them, this thesis would not have been possible.

I thank Prof. Donatello Conte and Prof. Jean-Yves Ramel for allowing me to collaborate with them at University of Tours (France) and Prof. Carlos-Francisco Moreno-García for our collaboration at Robert Gordon University (Scotland). The weekly meetings with them were very inspiring and they gave me new research perspectives and ideas. Furthermore, I would like to thank Prof. Eyad Elyan, Prof. Nirmalie Wiratunga, Prof. John McCall and their research groups at Robert Gordon University and National Subsea Center for inviting me to their weekly meetings during my mobility stay in Scotland.

I want to thank Prof. Alberto Fernández for the year working together in NanoDESK project. I want to thank the colleagues of the Graph Corner: Prof. Carme Julià Ferré, Natàlia Segura and Sarah Fadlallah for our weekly graph-meetings.

I thank Dr. Carlos García-Hernández, Dr. Mostafa Sarker, Armin Masoumian and Yarkin A. Çetin for their support.

I want to thank my friends who have been encouraging me during these years.

I want to thank my brothers José Manuel and Raúl, who have always been with me giving me their strong support.

I want to thank Dr. Antonio J. Carrasco for sharing his life with me.

We are achieving our dreams together.

Gracias a mis padres Beatriz y José Manuel por su amor incondicional.

Ellos me enseñaron que la vida es el don más valioso que tenemos y que con esfuerzo y constancia se pueden conseguir cosas maravillosas.

Es necesario sembrar la semilla adecuada y cuidarla con amor.

Así se recoge la alegría de la cosecha.

This doctoral thesis has received funding from Universitat Rovira i Virgili through grant 2018PMF-PIPF-30 and from European Union through Erasmus+ Program.

Contents

Abstract

Graphs are mathematical objects that depict the abstract representation of data when relations between elements are defined. When data is represented with graphs, nodes represent the main objects of the data and edges represent the relations between them. In this context, specific machine learning techniques need to be defined or adapted to obtain information from the data and predict characteristics or features that could be of interest for some applications. During the last 40 years, researchers have been analysing how to represent data with graphs and how to adapt machine learning methods to these structures or define new ones adapted to this framework. With this aim, the concept of *Graph Edit Distance (GED)* has been used during decades and several machine learning techniques use the *GED* as measure of dissimilarity between graphs to tackle the solution of different problems.

This thesis presents a compendium of machine learning methods focused on graph-represented data. Diverse tasks have been faced representing the data as graphs and the majority of the presented methods make use of the concept of *GED* as a tool to analyse the structure of the data. The techniques developed on this thesis demonstrate that the graph representation of data is suitable to solve different situations and can be explored for future contexts.

Nomenclature

Symbols

$\beta_{i,j}$	attribute of edge $e_{i,j}$
$\beta^t_{i,j}$	t^{th} attribute of edge $e_{i,j}$
γ_i	attribute of node v_i
γ^t_i	t^{th} attribute of node v_i
$\mathbb{R}^+_0$	$\{x \in \mathbb{R} : x \geqslant 0\}$
C^I_{Star}	Cost of insert the s
Del_v	Set of all the nodes deletions
$e'_{i,j}$	edge between i^{th} node and j^{th} node in graph G'
$e_{i,j}$	edge between i^{th} node and j^{th} node in graph G
G	Graph
G'	Graph
Ins_v	Set of all the nodes insertions
$M_{n\times m}(\mathbb{R}^+_0)$	Set of square matrix with n rows and m columns with coefficients in $\mathbb{R}^+_0$
$s.t.$	such that
$Subs_v$	Set of all the nodes substitutions

v'_i	i^{th} node in graph G'
v_i	i^{th} node in graph G
$w^e = (w^e_1, ..., w^e_M)$	vector of attributes' weights in edges
$w^v = (w^v_1, ..., w^v_N)$	vector of attributes' weights in nodes

Acronyms

GED	Graph Edit Distance
PR	Pattern Recognition
LDA	Linear Discriminant Analysis
NN	Neural Network
SVM	Support Vector Machine
P&ID	Pipping and Instrumentation Diagram

1 Introduction

Attributed graphs are commonly used as abstract representations for common structures such as documents, images or chemical compounds, among others (Sanfeliu et al. (2002)). Their applicability ranges from automatic character recognition of handwritten characters (Chaudhuri et al. (2017)) to toxicity analysis of chemical compounds (Garcia-Hernandez et al. (2019)), between others. When data is represented with graphs, nodes of graphs represent local parts of the object and edges represent the relations between these local parts.

These representations have been of crucial importance in pattern recognition throughout more than four decades, (Bunke and Allermann (1983); Sanfeliu and Fu (1983a); Sanfeliu et al. (2002)). Interesting reviews of techniques and applications are Conte et al. (2004); Vento (2015); Livi and Rizzi (2013) and Foggia et al. (2014). If elements in pattern recognition are modelled through attributed graphs, error-tolerant graph-matching algorithms (Conte et al. (2004); Vento (2013)) can be applied. The aim of these algorithms is to compute a mapping between nodes of two attributed graphs that minimises some kind of objective function (Solé-Ribalta et al. (2012); Serratosa and Cortés (2015b)) to deduce which is the best transformation between the graphs in some sense that is application dependent. A widely extended frameworks to deduce similarity between graphs is the concept of *Graph Edit Distance (GED)* (Bunke and Allermann (1983); Sanfeliu and Fu (1983b); Sanfeliu et al. (2002); Stauffer et al. (2017); Gao et al. (2010a)). The general idea of the *GED* is to find a transformation between two graphs that minimises a special cost function. This concept allows the definition of a distance between graphs that can be used to develop machine learning methods applied to graphs.

This thesis presents diverse machine learning methods applied to graph-represented data. First of all, details about *GED* and other fundamental concepts that are needed to understand the rest of the manuscript are explained in Chapter 2. In each one of the other chapters, a different method has been developed to solve a different problem related with data that have been represented with graphs. More specifically, in Chapter 3, we explain an on-line method to learn some parameters related with the *GED* that has been tested in diverse types of graph-represented data. In Chapter 4, we describe a method to classify molecules based on the *GED*. In this chapter, the molecules have been represented with graphs and the *GED* has been used as a similarity measure between the molecules.

In Chapter 5 and Chapter 6 we present two different methods related with the process of digitalisation of industrial diagrams. A data set of pipping and instrumentation diagrams of oil and gas facilities has been represented with graphs and we have faced two different tasks. First, in Chapter 5, we present an automatic method to help to engineers in the correction of the digitisation of diagrams. Secondly, in Chapter 6, we present a method to look for substructures in big diagrams using the *GED*. Chapter 5 is the only method of this thesis that does not use the *GED*, but it is an interesting method that reduces the effort of engineers using the graph representation of data.

At the end of the manuscript, Chapter 7 is devoted to general conclusions from the thesis, and the last Chapter 8 is an enumeration of the works published during the progress of this thesis.

2 Methods

2.1 Introduction

The advance of computation capacities are permitting that the graph representation of data becomes popular in the field of Pattern Recognition (PR). When PR problems are addressed with graphs, error-tolerant graph matching techniques outcome useful. The aim of these techniques is to find the best mapping between nodes and edges that minimises some kind of objective function that is adapted to the application. In this chapter we present the main concepts related with the graph matching problem and with the definition of the *Graph Edit Distance*.

2.2 Graph Matching and Graph Edit Distance

One of the most used frameworks to define the error-tolerant graph matching is through the *Graph Edit Distance (GED)* (Bunke and Allermann (1983); Sanfeliu and Fu (1983b); Sanfeliu et al. (2002); Stauffer et al. (2017); Gao et al. (2010a)). The main idea of the *GED* is to define the difference between two graphs as the amount of distortion required to transform a graph into another one. Calculate the *GED* between a pair of attributed graphs G and G' consists in finding the best sequence of edit operations that converts one graph into another with the minimum cost that is application dependent. To continue with the detailed definition of the *GED*, we introduce some concepts and notation that are required.

Given an attributed graph G, we call v_i to the i^{th} node in G and $e_{i,j}$, the edge between node v_i and node v_j in G. We define γ_i as the attribute of node v_i and γ_i^t as the t^{th} attribute of node v_i and $\beta_{i,j}^t$ is the t^{th} attribute of edge $e_{i,j}$. Given a node $v_i \in G$, each node v_j connected with v_i is called *neighbour* of v_i. We can consider the set of all the neighbours of v_i and we call it *Neighbourhood of* v_i. We call *degree* of v_i, d_i, to the number of neighbours of v_i and we define the *Star* centered on the node v_i, $Star(v_i)$ (Serratosa and Cortés (2015a)), as the local structure composed of the node itself, its neighbours, and the correspondent edges connecting them. Similarly, given another attributed graph G', we define v'_i and $e'_{i,j}$ as the i^{th} node and the edge between nodes v'_i and v'_j respectively, and all the concepts are defined in similar way than in G.

2.2.1 Definition of Graph Edit Distance

Having a pair of graphs, G and G', a node-to-node mapping $f : G \to G'$ between nodes of both graphs, is a bijective function that assigns one node of G to only one node of G'. We suppose both graphs have the same number of nodes since they have been expanded with new nodes, called $Null$. Therefore, if n and m were the initial number of nodes of G and G' respectively, the final number of nodes in both graphs is $n + m$. We use the notation $f(a) = i$ to represent the mapping from node v_a to node v'_i. Note that the mapping between edges is imposed by the mapping of

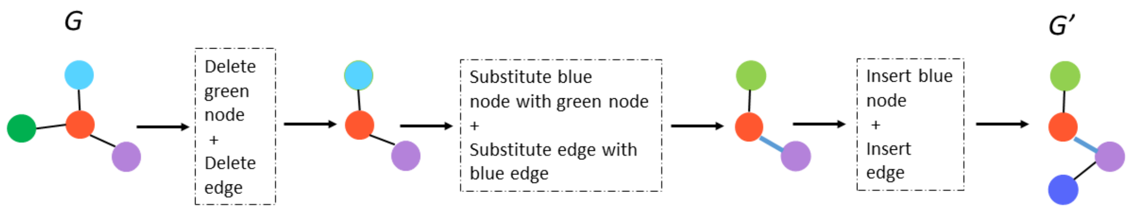

G
Delete green node + Delete edge
Substitute blue node with green node + Substitute edge with blue edge
Insert blue node + Insert edge
G'

Figure 2.1. Transformation of graph G in graph G'.

the nodes whose edges are connected. We say that $f(a) = i$ is a node substitution if both nodes are not $Null$. It is an insertion if node v_a is a $Null$ and v'_i is not a $Null$. Finally, it is a deletion if node v'_i is a $Null$ and v_a is not a $Null$. Similarly happens with the edges. Given two graphs G and G', Figure 2.1 shows an example of the edit operations to transform one into the other.

To quantify the importance of these operations transforming the graphs, a cost is assigned to each operation depending on the attributes on the involved nodes or edges. The cost of substituting a node $v_a \in G$ by a node $v'_i \in G'$ is denoted by $C_v^S(a,i)$. The cost of deleting a node v_a is $C_v^D(a)$. Finally, the cost of inserting the node v'_i is $C_v^I(i)$. Similarly happens with the costs on the edges: The cost of substituting an edge $e_{a,b}$ by an edge $e'_{i,j}$ is $C_e^S(a,i,b,j)$. The cost of deleting an edge $e_{a,b}$ is $C_e^D(a,b)$. And finally, the cost of inserting an edge $e'_{i,j}$ is $C_e^I(i,j)$. These costs are defined in equations 2.1 and 2.2:

$$C^v(i,a) = \begin{cases} C_S^v(i,a) & \text{if } v_i \neq Null \wedge v'_a \neq Null \\ C_D^v(i) & \text{if } v_i \neq Null \wedge v'_a = Null \\ C_I^v(a) & \text{if } v_i = Null \wedge v'_a \neq Null \end{cases} \tag{2.1}$$

$$C^e(i,j,a,b) = \begin{cases} C_S^e(i,j,a,b) & \text{if } e_{i,j} \neq Null \wedge e'_{a,b} \neq Null \\ C_D^e(i,j) & \text{if } e_{i,j} \neq Null \wedge e'_{a,b} = Null \\ C_I^e(a,b) & \text{if } e_{i,j} = Null \wedge e'_{a,b} \neq Null \end{cases} \tag{2.2}$$

We define the cost of transforming graph G into graph G' through the mapping f as the sum of all the edit costs with Equation 2.3:

$$Cost(G, G', f) = \sum_{\{v_i \in G\}} C^v(i, f(i)) + \sum_{\{e_{i,j} \in G\}} C^e(i, j, f(i), f(j)) \tag{2.3}$$

We observe that there is not an unique way to transform one graph into another. Thus, it makes sense trying to find the transformation or mapping between both graphs that has the lowest cost. With this aim, we define the *GED* as the minimum transformation cost, given all transformations from one graph into another. We define the *Graph Edit Distance (GED)* as:

$$GED(G, G') = \min_{\{f: G \to G'\}} \left\{ \sum_{\{v_i \in G\}} C^v(i, f(i)) + \sum_{\{e_{i,j} \in G\}} C^e(i, j, f(i), f(j)) \right\} \tag{2.4}$$

2.2.2 Computing the Graph Edit Distance

The computation of the *GED* is an optimisation problem that is NP-hard, it means that it can not be solved in polynomial time. The optimal computation of the *GED* is usually carried out by means of A* algorithm (Hart et al. (1968)). The computational cost of this method is exponential in the number of nodes of the involved graphs (Garey and Johnson (1990)). For this reason, some heuristic algorithms that deduce a sub-optimal *GED* in polynomial time have been presented (Justice and III (2006); Neuhaus et al. (2006); Riesen and Bunke (2009); Serratosa (2014b, 2015); Riesen et al. (2018)). In general, these sub-optimal algorithms optimise local instead of global criteria and a sub-optimal *GED* can be computed. One of the most extended algorithms to compute the *GED* is *Bipartite* graph matching (Riesen and Bunke (2009); Serratosa (2014a)), which is the algorithm that we have used in this thesis to compute approximations of the *GED*. This algorithm defines a cost matrix, applies a linear solver such as *Hungarian* method and deduces the correspondence f with the sub-optimal *GED*. For more details, the reader is referred to Riesen and Bunke (2009) and Serratosa (2014a).

These sub-optimal algorithms define the cost between two graphs for a specific node-to-node mapping, f, as the addition of the substitution, deletion and insertion costs of local structures of nodes. One of these local structures that can be used is the

Star centered in v_i, $Star(v_i)$, defined in the second paragraph of Subsection 2.2. We represent the sets of substitutions, deletions and insertions of nodes respectively with $Subs_v$, Del_v and Ins_v. Moreover, $C_S^{Star}(a,i)$ denotes the cost of substituting the *Star* centered at node $v_a \in G$ by the *Star* centered at node $v'_i \in G'$. $C_D^{Star}(a)$ denotes the cost of deleting the *Star* centred at v_a and $C_I^{Star}(i)$ denotes the cost of inserting the *Star* centred at v'_i. These *Star* costs depend on the costs on nodes and edges $C_S^v(a,i)$, $C_D^v(a)$, $C_I^v(i)$, $C_S^e(a,i,b,j)$, $C_D^e(a,b)$ and $C_I^e(i,j)$ (Serratosa and Cortés (2015a)).

Then, we can define a sub-optimal costs of the mapping f between G and G' as:

$$\begin{gathered} Cost(G,G',f) = \\ \sum_{\forall Subs_v} C_S^{Star}(a,i) + \sum_{\forall Del_v} C_D^{Star}(a) + \sum_{\forall Ins_v} C_I^{Star}(i) \end{gathered} \tag{2.5}$$

Consequently, a sub-optimal *GED* can be calculated finding a mapping that minimises Equation 2.5.

2.2.3 Learning the costs of the Graph Edit Distance

The penalty costs are application dependent and need to be set such that the *GED* reflects the dissimilarity between the graphs. These costs can be manually tuned or automatically computed through a learning method. In this section, we select several learning methods and we classify them into three classes, depending on the nature of the edit costs they learn. We present below a summary with the descriptions of these three classes of methods.

1) C_S^v, C_D^v , C_I^v, C_S^e, C_D^e and C_I^e as functions.

The methods in this first class define the six node and edge edit costs C_S^v, C_D^v , C_I^v, C_S^e, C_D^e and C_I^e in Equation 2.1 and Equation 2.2 as functions that depend on the attributes on the nodes and edges. These functions are learned using, for instance, a neural network or a probability density distribution.

- Neuhaus and Bunke (2005): The method feeds a self-organised map with the attributes of the nodes or the edges and at the output obtains the substitution, deletion and insertion costs on nodes and edges. The optimisation function used in the learning process is the average of eight optimisation functions: Davies–Bouldin, Dunn, C, Goodman–Krusk, Calinski–Haraba, Rand index, Jaccard, Fowlkes–Mallo.

- Neuhaus and Bunke (2007): This method is similar to the previous method. Nevertheless, it uses the Dunn index as the optimisation function. Moreover, given the attributes on the nodes or the edges, it computes the costs as the inverse of the probability set by a probability density function.

- Caetano et al. (2009): In this case, the learning set has a different structure since it is composed by pairs of attributed graphs and the node-to-node mapping between them, instead of classified attributed graphs. Thus, the node-to-node mappings in the learning set become the ground truth mappings and the optimisation function learns the edit costs such that the resulting node-to-node mappings tend to be close to the node-to-node mappings in the learning set. This optimisation function has been called the correspondence accuracy. The method learns the weights w_S^v and w_S^e of the weighted Euclidean distance that define the substitution costs on nodes and edges. Insertion and deletion of nodes and edges are not learned and assumed to be constant.

- Leordeanu et al. (2012): The method learns w_S^v and w_S^e in the same way as Caetano et al. (2009) but the optimisation function is the maximisation of the recognition ratio of the training set composed of classified graphs.

- Cortés and Serratosa (2015): The method has the same training set and optimisation function than the one described in Caetano et al. (2009). Nevertheless, the method learns the insertion and deletion costs as constants. The method assumes C_S^v and C_S^e are defined as an Euclidean distance without weights.

- Cortés and Serratosa (2016b): The method has the same optimisation function and learning set as the one in Caetano et al. (2009) but a different learning algorithm.

2) C_D^v, C_I^v, C_D^e and C_I^e as constants.

In this class, the four node and edge edit costs C_D^v, C_I^v, C_D^e and C_I^e in Equation 2.1 and Equation 2.2 are constants, what means that they do not depend on the attributes.

- Cortés et al. (2019, 2018): The method learns the substitution functions on nodes and edges through a neural network being the set of attributes on the nodes and edges its input. Insertion and deletion of nodes and edges are not learned and assumed to be constants. In this case the optimisation function is the correspondence accuracy.

- Santacruz and Serratosa (2019, 2018b): Similar to the method in Cortés et al. (2019, 2018) but the insertion and deletion costs on nodes and edges are also learned through a neural network. Thus, the method learns all six edit costs.

3) C_S^v and C_S^e defined as Equation 2.6.

Finally, in this third class, the node and edge substitution edit costs (C_S^v and C_S^e in Equation 2.1 and Equation 2.2) are defined as Equation 2.6:

$$\begin{aligned} C_S^v(i,a) &= \sum_{t=1}^{N} w_t^v \left|\gamma_i^t - \gamma_a'^t\right| \\ C_S^e(i,j,a,b) &= \sum_{t=1}^{M} w_t^e \left|\beta_{i,j}^t - \beta_{a,b}'^t\right| \end{aligned} \tag{2.6}$$

where γ_i^t is the t^{th} attribute of node v_i and $\beta_{i,j}^t$ is the t^{th} attribute of edge $e_{i,j}$. The vector $w^v = (w_1^v, ..., w_N^v)$ is the vector of nodes attributes' weights and $w^e = (w_1^e, ..., w_M^e)$ is the vector of edges attributes' weights. N and M are the dimensions of the spaces where the attributes in nodes and edges belong to.

- Algabli and Serratosa (2018a): The method learns the weights of the weighted Euclidean distance to define the substitution costs w_S^v and w_S^e, and also the deletion and insertion costs as constants on nodes and edges. The optimisation function is the correspondence accuracy.

- Martineau et al. (2018): The method learns the weights on each node or edge, instead of on the node and edge attributes. These weights depend on how important the nodes and edges are to describe the class of the graph.

3
On-line parameter learning

3.1 Introduction

The costs of the GED are application dependent and need to be set such that the GED reflects the real dissimilarity between graphs. These costs can be manually tuned or automatically computed through learning methods and we can discern between off-line and on-line learning methods.

The main characteristic of the off-line methods is that they learn in a first stage with the whole learning data set and the deduced model is used in a second phase. But in some cases, during the learning process, not all the data is available at once and there is a need to develop on-line learning methods. These on-line methods receive pieces of data and, each time a new batch is received, new parameters are learned considering the new data and also the current knowledge of the model.

In this chapter, we present an on-line method to learn the edit costs of the GED that automatically recomputes the values of the costs when new data are available. Moreover, the method allows the computation of the GED at the same time as the costs are learned. It means that we do not need to have all the data set to obtain an approximation of the GED.

This chapter is organised as follows. Firstly, we present a background in Section 3.2. Secondly, we present the proposed algorithm in Section 3.3. Thirdly, we show the experiments in Section 3.4 and, in the end, we present the conclusions of the chapter in Section 3.5.

3.2 Background

There are some scenarios where there is not the complete learning set or the learning set changes in different stages. For instance, in Cortés and Serratosa (2016a); Moreno and Serratosa (2015); Cortés and Serratosa (2015), an interactive method is defined where a human helps a fleet of robots to deduce their relative position and also to learn the computer vision parameters. Another example is the classification of job advertisements (Boselli et al. (2018)), that have to be continually actualised and classified at the same time that they appear and disappear. On-line learning is as well related with active querying approaches to decide which data are most valuable in the learning process (Zhang et al. (2016)). In these cases, on-line algorithms (Zhao and Hoi (2013)) can be used to continue learning the characteristics of the learning data set.

In the specific case of graph matching and learning the costs of the GED, many off-line algorithms have been published (see Subsection 2.2.3), but, to our knowledge, the methods published in Conte and Serratosa (2020a) and Rica et al. (2019) are the only ones that present on-line methods to learn costs and parameters related with the graph matching problem. In Conte and Serratosa (2020a), correspondences between graphs are defined in a sequential order, and the method proposes which node-to-node mapping contributes the most to the learning process. Furthermore, it gives the possibility of human interaction to correct some node-to-node mappings that could be incorrect.

The method explained in this chapter is the method presented in Rica et al. (2019). It is an on-line method that learns the costs of the GED. It is inspired in the off-line algorithm published in Algabli and Serratosa (2018a). Some processes have been incorporated to move this algorithm into the on-line paradigm and to keep the method learning with the minimum amount of data.

3.3 The proposed method

The aim of this method is to learn the costs and weights involved in the *GED* introducing new data in each step of the learning process instead of learning with the whole data set at once. This on-line learning method fits in the second and third classes of algorithms explained in Section 2.2.3 because the insertion and deletion costs are defined as constants. Therefore, the insertion and deletion costs of nodes are imposed to be equal. It means that $C_I^v(a) = C_D^v(i) = K^v$, being K^v a real number. In the same way, the insertion and deletion costs of edges are imposed to be equal, $C_I^e(i,j) = C_D^e(i,j) = K^e$, where K^e is a real numbers. We also define the substitution costs through the weighted Manhattan distance described in Equation 2.6 and we impose the next restrictions in the weights:

$$\sum_{t=1}^{N} w_t^v = 1 \qquad \sum_{t=1}^{M} w_t^e = 1 \tag{3.1}$$

Thus, the parameters to be learned with this method are K^v, K^e and $w^v = (w_1^v, ..., w_N^v)$ and $w^e = (w_1^e, ..., w_M^e)$, where w^v and w^e are the weights in Equation 2.6 and Equation 3.1. Given two graphs G and G', we want to learn these parameters such that the correspondence $g : G \to G'$ between G and G' computed by the learning algorithm in each step, becomes closer to the ground-truth correspondence $f : G \to G'$ for all pairs of graphs G and G'. This means that this method can be applied to data where each element in the data set is a triplet (G, G', f), where G and G' are graphs and f is a node to node mapping that is the ground-truth application between them, it is, the mapping between nodes that is accepted as the correct mapping between them.

Throughout the rest of this section, we explain the input, and steps of the method that returns the edit costs and weights K^v, K^e, w^v and w^e. The algorithm is composed of six main steps which are schematically shown in Figure 3.1 and summarised in Algorithm 1.

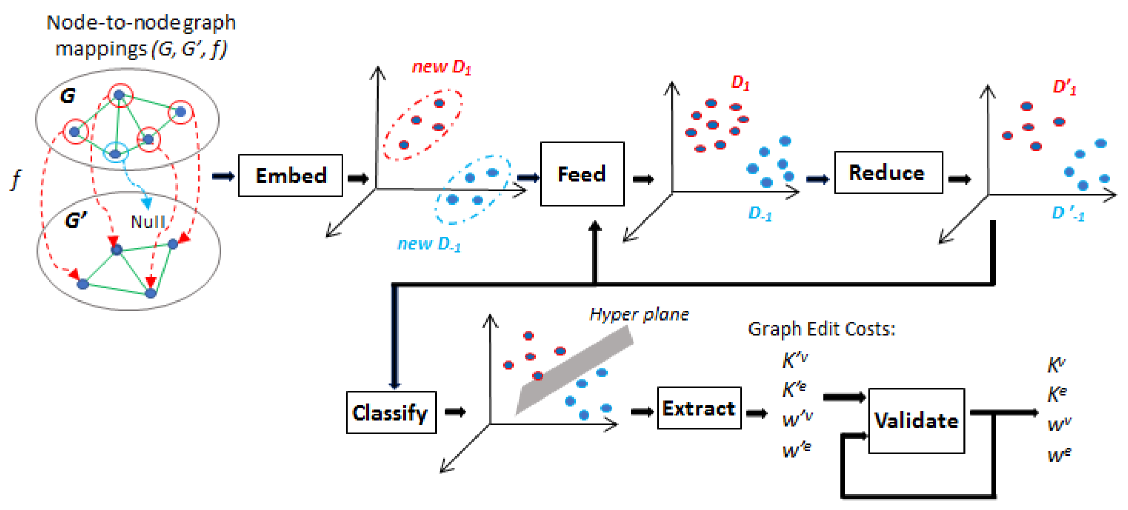


Figure 3.1. Basic scheme of the on-line learning method.

Input of the method

The input of this on-line method is composed of four main elements that are detailed below:

1) D_1 and D_{-1}. They are two data sets that contain the initial node-to-node mappings existing in the learning set. These initial data have been previously embedded with the embedding function that will be explained in the subsection 3.3 (*Embed*). These two data sets represent the current knowledge of the system and the size is not fixed.

2) (G, G', f). It is the new data introduced in the system in each step of the learning process. It is composed of a new triplet where f is the ground-truth node-to-node mapping and two graphs G and G'. Note that the ground-truth node-to-node mapping has been imposed by a specialist.

3) K. It is a natural number that controls the amount of data that the algorithm has to keep for the next iteration.

4) Val_DB. A validation data set that is used to validate the values of the learned parameters.

Below, we explain in detail each step of the method summarised in Algorithm (1):

Embed

The first step of the algorithm is to embed the available data. The new data is embedded into the euclidean space $S \subseteq \mathbb{R}^{N+M+1}$ with coordinates $(S_1^v, ..., S_N^v, S_1^e, ..., S_M^e, S_{K^e})$. This space is the same as described in Algabli and Serratosa (2018a) and the reader is referred to it for more details.

Given a new triplet (G, G', f), this step generates two sets, *new* D_1 and *new* D_{-1}, composed of elements in space S that represent the node substitutions and node deletions in (G, G', f), respectively. According to Algabli and Serratosa (2018a), each node substitution is transformed into an element in *new* D_1 but each node deletion is transformed into d elements in *new* D_{-1}, where d is the number of non-null nodes in graph G'. The coordinates of the points generated by the embedding depend on if they are substitutions or deletions:

- The substitution of v_i by v'_a imposed by $f(i) = a$, according to Algabli and Serratosa (2018a), generates the point $S^i = (S_1^v, ..., S_N^v, S_1^e, ..., S_M^e, S_{K^e})^i$ in the set *new* D_1 and the coordinates of this point are defined as follows:

$$\begin{gathered} S_t^v = \begin{cases} \frac{Z_1^v}{|n_i - m_a| - n_i - 1} & \text{if t = 1} \\ \frac{Z_t^v - Z_1^v}{|n_i - m_a| - n_i - 1} & \text{if t>1} \end{cases} \\ S_t^e = \begin{cases} \frac{Z_1^e}{|n_i - m_a| - n_i - 1} & \text{if t = 1} \\ \frac{Z_t^e - Z_1^e}{|n_i - m_a| - n_i - 1} & \text{if t>1} \end{cases} \\ S_{K^e} = \frac{|n_i - m_a| - n_i}{|n_i - m_a| - n_i - 1} \end{gathered} \tag{3.2}$$

where n_i is the number of neighbour nodes of v_i and m_a is the number of neighbour nodes of v'_a. Moreover, if $f(j) = b$, according to Algabli and Serratosa (2018a), then Z_t^v and Z_t^e are defined as follows:

$$Z_t^v = \left|\gamma_i^t - \gamma_a'^t\right| + \sum_{\forall G_{i,j} | \exists G'_{a,b}} \left|\gamma_j^t - \gamma_b'^t\right| \tag{3.3}$$

$$Z_t^e = \sum_{\forall G_{i,j} | \exists G'_{a,b}} \left| \beta_{i,j}^t - \beta'^t_{a,b} \right| \tag{3.4}$$

- The deletion of a node v_i imposed by f generates a set of points in *new* D_{-1}. The size of this set is the number of nodes in G' that are not null. Each point $s \in S$ has the coordinates $s = (s_1^v, ..., s_N^v, s_1^e, ..., s_M^e, s_{K^e})$ that have been computed according to Algabli and Serratosa (2018a) through Equation 3.2, Equation 3.3 and Equation 3.4.

Figure 3.1 shows an input triplet composed of two graphs that have five nodes, each. In the first graph, the five nodes are non-null but there is one null node in the second graph. There is also the ground-truth correspondence f composed of four node substitutions (red arrows) and one deletion (blue arrow). Then, this triplet generates four points in *new* D_1 and four points in *new* D_{-1}. Note Figure 3.1 shows the specific case of $N = M = 1$ then being S a 3-dimensional space with coordinates $S = (S_1^v, S_1^e, S_{K^e})$.

Feed and Reduce

The new sets *new* D_1 and *new* D_{-1} and the previous ones D'_1 and D'_{-1} are put together. $D_1 = D'_1 \cup new\ D_1$ and $D_{-1} = D'_{-1} \cup new\ D_{-1}$. The amount of data has to be almost constant in the on-line algorithms and cannot depend on the number of iterations of these algorithms. The aim of this step is to reduce the number of elements in sets D_1 and D_{-1} but holding two properties. The first one is keeping the general distance between elements as well as their positions. This means that we want to have less elements but maintain the same information of the sets as much as possible. The second one is keeping the same ratio of the number of elements on both sets. This is because, all the classifiers are biased by the order of the sets. Note that the generated sets D'_1 and D'_{-1} are returned by the algorithm to be fed at the next iteration.

The reduction is performed through the *Reduce Function* shown below. The input parameter K is the maximum number of elements per set at the end of each iteration.

From Line 1 to Line 8, the function deduces the number of elements that the updated sets D'_1 and D'_{-1} will have.

In Lines 9 and 10, the reduction is done in each set. Several strategies can be used to make this reduction in function *Data-Reduction*. We have tested two of them. In the first one, elements obtained by *Data-reduction* function are the centroids computed by the clustering algorithm *K-means*. In the second one, we use *K-Nearest Neighbours* algorithm, that is called throughout the thesis *Nearest*, which keeps the closest elements to the hyper-plane deduced in the previous iterations.

Reduce Function

Input(D_1,D_{-1},K)

Output(D'_1,D'_{-1})

```
1.   If $|D_1| > K \vee |D_{-1}| > K$
2.     If $|D_1| \geq |D_{-1}|$
3.       $K_1 = K$
4.       $K_{-1} = K * (|D_{-1}| / |D_1|)$
5.     Else
6.       $K_1 = K * (|D_1| / |D_{-1}|)$
7.       $K_{-1} = K$
8.     End if
9.     $D'_1 = \textit{Data-reduction}(D_1, K_1)$
10.    $D'_{-1} = \textit{Data-reduction}(D_{-1}, K_{-1})$
11.  End if
     End Function
```

Classify

This step computes the hyper-plane, shown in Equation 3.5 that best splits sets D'_1 and D'_{-1}. Note that the embedded space S was specifically defined such that the Equation 3.5 was the linear border between both sets (see Algabli and Serratosa (2018a) for demonstration). Constants in this linear equation are the substitution weights $w^v_2, ..., w^v_N$ and $w^e_2, ..., w^e_M$ and also the insertion and deletion costs on nodes and edges K^v and K^e.

$$\begin{aligned} S^v_1 + w^v_2 \cdot S^v_2 + ... + w^v_N \cdot S^v_N + \\ S^e_1 + w^e_2 \cdot S^e_2 + ... + w^e_M \cdot S^e_M + \\ K^e \cdot S_{K^e} + K^v = 0 \end{aligned} \tag{3.5}$$

We have explored two different strategies to deduce a hyper-plane given these two sets, Support Vector Machine (Cortes and Vapnik (1995)) and Linear Discriminant Analysis (Xanthopoulos et al. (2013)).

As an example, Figure 3.2 shows the points generated by substitutions (red crosses) and deletions (green dots) and, also, the plane that best splits both sets (Equation 3.5) computed using *Linear Discriminant Analysis (LDA)*. We chose the representation in *Letter Low* database (explained in the Experiments 3.4) because nodes in this database have only two attributes (the position in the image (x,y)) and edges do not have attributes. Consequently, the embedded space S has dimension three ($S \subseteq \mathbb{R}^3$) and we represent its coordinates as (S^v_1, S^v_2, S_{K^e}). Then, the expression of Equation 3.5 in this case is

$S^v_1 + w^v_2 \cdot S^v_2 + S_{K^e} + K^v = 0.$

Extract

The edit costs constants, $w^v_1, ..., w^v_N$, $w^e_1, ..., w^e_M$, K^v and K^e are extracted from the hyper-plane (Equation 3.5). Moreover, w^v_1 and w^e_1 are obtained through Equation 3.1.

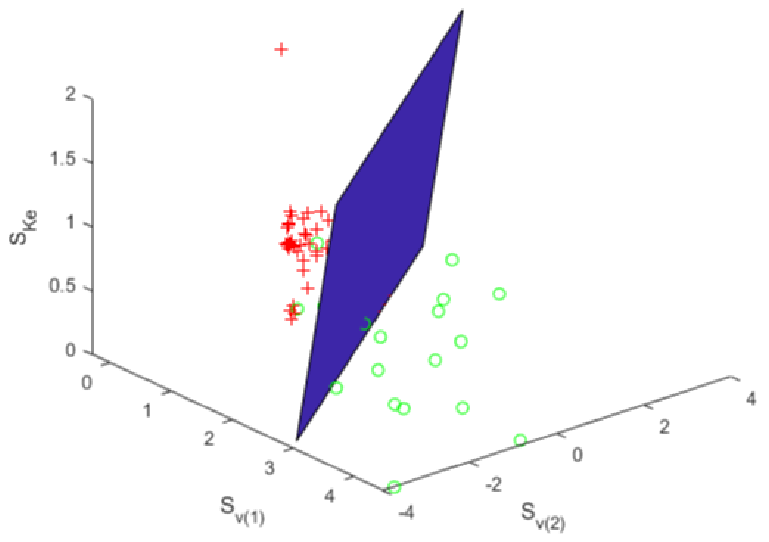


Figure 3.2. Embedded domain S with the embedded points and the splitting plane. LDA has been used to find the plane

Validate

This step validates $w_1^v, ..., w_N^v$, $w_1^e, ..., w_M^e$, K^v and K^e computed in **Extract** step using a validation database. For each element (G, G', f) of this validation database, we calculate the best correspondence obtained with the weights and costs $w_1^v, ..., w_N^v$, $w_1^e, ..., w_M^e$, K^v and K^e deduced by the method. Then, we calculate the number of node mappings that are different between the obtained correspondence and the ground truth correspondence. This measure is called the *hamming distance*. The values of $w_1^v, ..., w_N^v$, $w_1^e, ..., w_M^e$, K^v and K^e are only updated whether the average hamming distance between the ground truth correspondences in Validation database and the correspondences returned in the current iteration is lower than the average hamming distance obtained in previous iterations.

This strategy tries not to unlearn and it is called *pocket algorithm* (Stephen (1990)).

Algorithm 1 On-line costs learning.

```
Input($K$,$D_1$,$D_{-1}$,$(G, G', f)$,$Val\_DB$)
Output($K^v$,$K^e$,$w^v$, $w^e$)

    1. $(new\ D_1,\ new\ D_{-1}) = \mathbf{Embed}(G, G', f)$
    2. $(D_1,\ D_{-1}) = \mathbf{Feed}(D'_1, D'_{-1},\ new\ D_1,\ new\ D_{-1})$
    3. $(D'_1,\ D'_{-1}) = \mathbf{Reduce}(D_1, D_{-1}, K)$
    4. $Hyper\text{-}plane = \mathbf{Classify}\left(D'_1, D'_{-1}\right)$
    5. $[K'^v, K'^e, w'^v, w'^e] = \mathbf{Extract}(Hyper\text{-}plane)$
    6. $[K^v, K^e, w^v, w^e] = \mathbf{Validate}(K'^v, K'^e, w'^v, w'^e, Val\_DB)$

End Algorithm
```

3.4 Experiments

We validate the method using eight databases, which were used to test other learning methods (Moreno-García et al. (2016), Cortés and Serratosa (2015); Cortés and Serratosa (2015)). The data sets *Letter Low*, *Letter Med* and *Letter High* represent artificially distorted letters of the Latin alphabet with an increasing level of distortion. *Boat*, *East Park*, *East South* and *Resid* represent images and *Fingerprint* represents human fingerprints. The code and databases used to validate our method are publically available at [1]. The main characteristic of these databases is that their registers are not only composed of a graph and its class, but they are composed of a pair of graphs and a ground-truth node-to-node mapping between them, as well as their class. This register structure is useful to analyse and develop graph matching algorithms and to learn their parameters in a broad manner. Table 3.1 summarises the main features of these databases. With the aim of using the minimum amount of information in the learning algorithm, only 5% and 20% of the validation set have been used for the first three databases and the fifth last databases, respectively.

This section has been split in two sub-sections. The aim of the first one is to show the behaviour of our method considering different values of parameter K in the *Reduce* step, which controls the amount of data kept in the model. Also we compare different strategies, on the one hand, *K-means* or *Nearest* in the *Reduce* step, and,

[1]https://deim.urv.cat/~francesc.serratosa/

		Letter High	Letter Med	Letter Low	Boat	East Park	East South	Resid	Finger-print
	Learn	750	750	750	5	5	5	5	10
Graphs	**Validation**	750	750	750	5	5	5	5	10
	Test	750	750	750	5	5	5	5	10
	Learn	37500	37500	37500	25	25	25	25	20
Corresp.	**Validation**	37500	37500	37500	25	25	25	25	20
	Test	37500	37500	37500	25	25	25	25	20
N° classes		15	15	15	1	1	1	1	5
N° attributes		2	2	2	64	64	64	64	4
Description			(x,y)				*SURF*		(x,y,θ,T/P)
Avg. N°nodes		4.6	4.6	4.6	50	50	50	50	37.5
Avg. N°edges		6.2	6.4	9	278.4	276	278.8	276.4	199.2
Max. N° nodes		8	9	9	50	50	50	50	71
Max. N° edges		12	14	18	282	280	282	278	384
Avg. N° subst.		4.2	4.2	4.2	18	16	13	18	4
Avg. Dels./Ins.		0.4	0.4	0.4	32	34	37	32	33.5

Table 3.1. Main features of the eight databases.

on the other hand, *Support Vector Machine* (*SVM*) or *Linear Discriminant Analysis* (*LDA*) in the *Classify* step. The goodness of the method is analysed through the *Area Under the Curve* (*AUC*) and the *Accuracy*. The aim of the second sub-section is to compare our on-line method to several off-line methods. The goodness of the compared methods is evaluated through the accuracy. The accuracy is defined as the number of node mappings equal to the node ground truth mappings divided by the total number of node mappings.

3.4.1 Analysis of the on-line method

Tables 3.2, 3.3 and 3.4 show the *AUC* returned by the on-line method given the combination of strategies *K-means* or *Nearest* and algorithms *SVM* or *LDA*. Moreover, we show the *AUC* given $K = 125$, $K = 250$, $K = 500$ in the first seven databases and $K = 25$ in the eighth.

		Letter High	Letter Med	Letter Low	Boat	East Park	East South	Resid	Finger-print
k-means	**SVM**	85	82	94	87	88	95	86	79
	LDA	85	82	94	74	75	82	77	63
Nearest	**SVM**	86	86	94	88	91	97	92	73
	LDA	86	76	95	76	78	83	78	72

Table 3.2. Area under the curve (AUC) obtained with 100% of the learning set with K=125 in the first seven databases (columns), and K=10 in the last one.

		Letter High	**Letter Med**	**Letter Low**	**Boat**	**East Park**	**East South**	**Resid**	**Finger-print**
k-means	**SVM**	84	81	94	87	89	95	87	68
	LDA	86	78	94	81	75	90	76	80
Nearest	**SVM**	85	78	94	88	91	97	93	75
	LDA	86	78	95	86	84	95	80	71

Table 3.3. Area under the curve (AUC) obtained with 100% of the learning set with K=250 in the first seven databases (columns), and K=25 in the last one.

		Letter High	**Letter Med**	**Letter Low**	**Boat**	**East Park**	**East South**	**Resid**	**Finger-print**
k-means	**SVM**	85	81	94	87	89	95	87	72
	LDA	86	78	94	82	85	93	84	65
Nearest	**SVM**	85	79	93	90	91	96	92	50
	LDA	86	68	95	81	84	95	85	72

Table 3.4. Area under the curve (AUC) obtained with 100% of the learning set with K=500 in the first seven databases (columns), and K=50 in the last one.

In most cases, it seems as K almost does not influence the AUC when SVM is used. Contrarily, there is some influence on the AUC when LDA is selected. This influence is more appreciated in the last five databases. In general, the combination that returns the highest AUC is $Nearest$ and SVM. Besides, it is also independent of parameter K. Figure 3.3 shows an example of the *Accuracy* returned by the test set while the on-line algorithm keeps incorporating new data from the learning set. We have selected three different values of K and the combination K-*means* and SVM. We realise that the accuracy increases with different slopes at the beginning of the training process. Higher the K is, slower the training process becomes. Nevertheless, the accuracy stabilises for each value of K when an enough percentage of data has been introduced. This behaviour is similar to the other databases and methods. To illustrate the behaviour of our method, Figure 3.4, Figure 3.5 and Figure 3.6 also show the *Accuracy* for all the on-line methods with the same value of K in databases $Letter\ Med$, $East\ South$ and $Fingerprint$. In the three examples, plots are different but they stabilise as all databases do.

Tables 3.5 and 3.6 show respectively the values of parameters K^v and K^e obtained with 20% and 100% of databases with different values of K. We realise that different algorithms return different values, but in some cases the are similar values.

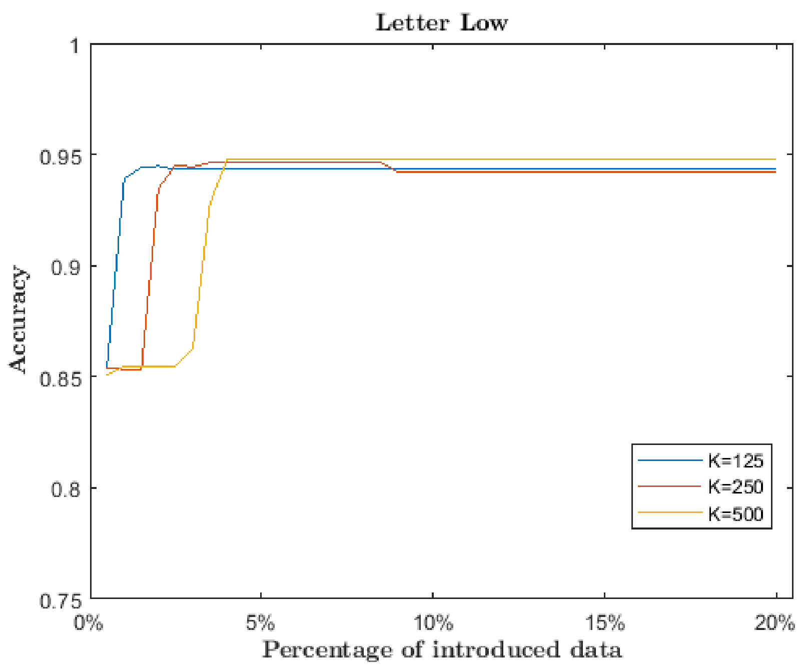


Figure 3.3. Letter Low accuracy using *K-means* and *SVM*.

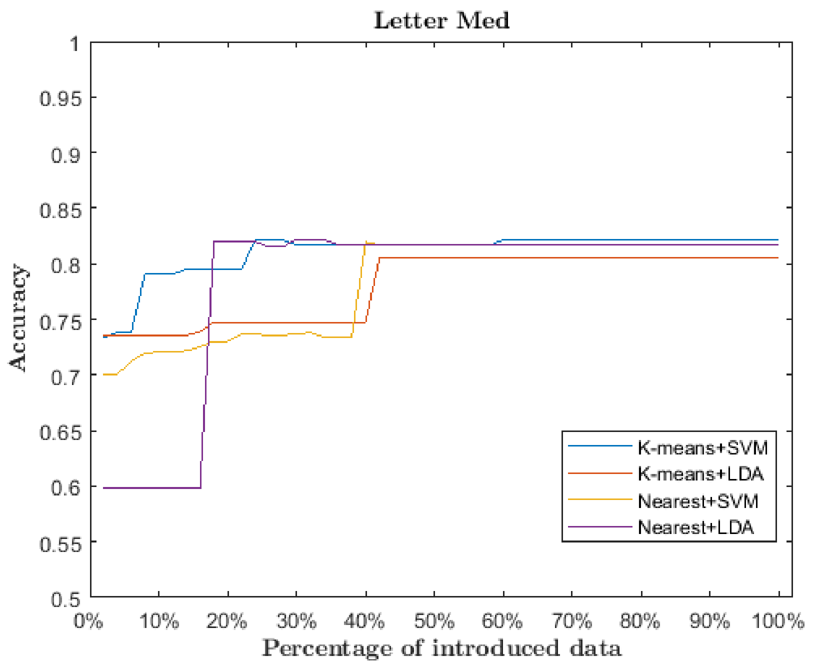


Figure 3.4. Accuracy in *Letter Med* with K=250.

Also, in Table 3.8 we show run times of the method (Matlab 2020a and Intel Core i7).

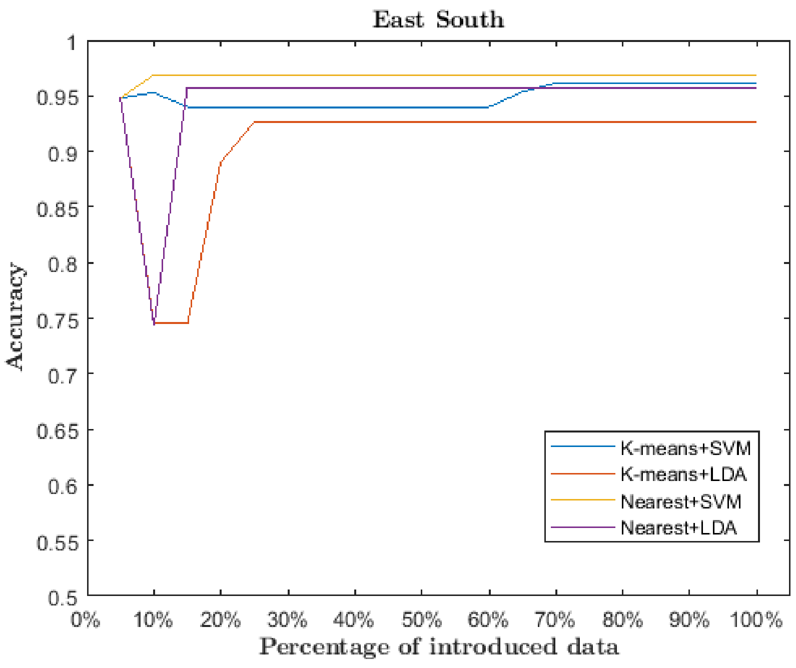


Figure 3.5. Accuracy in *East South* with K=250.

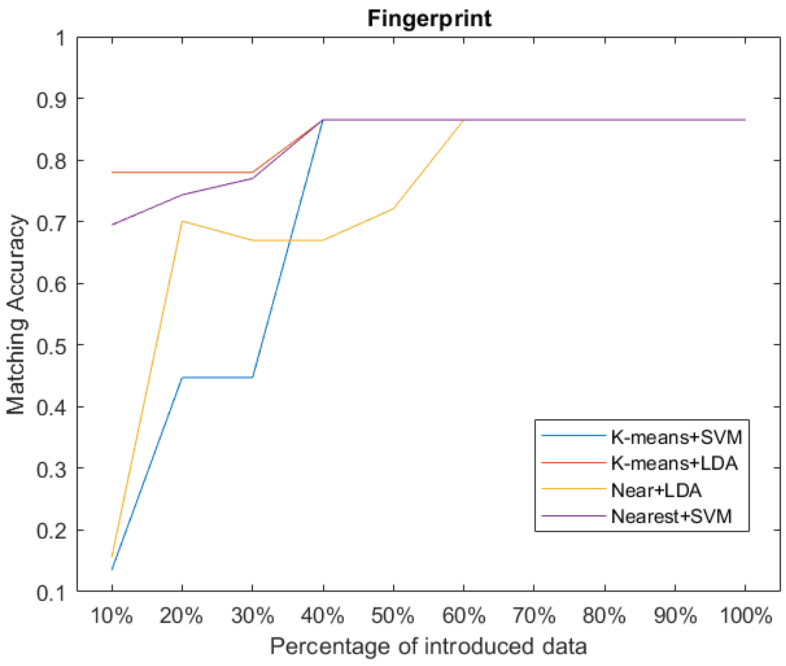


Figure 3.6. Accuracy in *Fingerprint* with K=25.

3.4.2 Comparing to the off-line methods

In the first five rows in Table 3.7, it is shown the accuracy of the off-line methods presented in Neuhaus and Bunke (2005, 2007); Cortés and Serratosa (2015); Algabli and Serratosa (2018a); Santacruz and Serratosa (2018b) given the test set. These

			Letter High	Letter Med	Letter Low	Boat	East Park	East South	Resid	Finger-print
100% data	**k-means**	**SVM**	-23.449	-52.372	-0'53	96	125'3	115'6	232'1	-10'9
		LDA	-0'89	-0'9	-0'81	-0'02	-0'02	-0'02	-0'01	-8'3
	Nearest	**SVM**	-80.339	20.071	-0'33	195'3	-6'1	13'3	-51	-2'8
		LDA	0'58	-2'27	0'22	-0'02	-0'02	-0'02	-0'02	-0'5
20% data	**k-means**	**SVM**	22.190	-90.168	-0'53	127	125'3	1440'3	93'1	1'4
		LDA	-0'89	-2'77	-0'81	-0'02	-0'02	-0'02	-0'01	-9'5
	Nearest	**SVM**	-80.339	-28.854	-0'4	161'6	-6'1	13'3	-19'2	-0'4
		LDA	-0'14	-0'98	-0'26	-0'01	-0'02	-0'02	-0'01	2'1

Table 3.5. K^v parameter obtained with 100% and 20% of learning set and k=250 in the first seven databases, and k=25 in the last one.

			Letter High	Letter Med	Letter Low	Boat	East Park	East South	Resid	Finger-print
100% data	**k-means**	**SVM**	-11'24	-0'87	-0'51	0'9	3'8	2'5	1	-12'5
		LDA	-0'22	-1'07	-0'25	0'02	0'02	0'02	0'01	8'4
	Nearest	**SVM**	-0'8	-80.291	-0'15	1'7	16	-17'2	-44'5	-2'4
		LDA	-3'4	5'28	-0'28	0'02	0'02	0'01	0'02	-0'1
20% data	**k-means**	**SVM**	-0'34	-0'41	-0'51	0'7	3'8	4'8	4'5	-5'4
		LDA	-0'22	-0'26	-0'25	0'02	0'02	0'01	0'02	10'9
	Nearest	**SVM**	-0'8	1'68	-0'15	16'2	16	-17'2	42'1	-1'5
		LDA	-1'55	0'27	-0'1	0'01	0'03	0'01	0'02	-2'8

Table 3.6. K^e parameter obtained with 100% and 20% of learning set and k=250 in the first seven databases, and k=25 in the last one.

values have been extracted from the experimental sections of those papers. In the rest of the rows in Table 3.7, it is shown the accuracy of our on-line method when the model has been trained using 100% and 20% of the learning set. We realise that, on the one hand, our method returns very competitive results in respect to the off-line methods. In the *Letter* databases, the accuracy turns out to be almost equal and in the rest of the databases, our method returns higher values. On the other hand, the accuracy returned by our method trained only with 20% of the learning database is almost similar to the accuracy returned by our method but trained with the whole learning database. The combinations of both facts makes the presented method to be really interesting since it has the three following properties:

1) The training stage can be alternated to the matching stage.

2) The accuracy tends to be the highest one.

3) Only a portion of the learning data is enough to properly train the model.

			Letter High	**Letter Med**	**Letter Low**	**Boat**	**East Park**	**East South**	**Resid**	**Finger-print**
		[A]	83	76	93	i.c.	i.c.	i.c.	i.c.	-
		[B]	89	87	**97**	42	54	40	54	-
		[C]	-	-	71	22	21	20	20	-
		[D]	82	70	85	68	68	74	78	-
		[E]	**89**	**87**	**97**	44	56	40	55	-
100% data	**k-means**	**SVM**	85	82	94	87	89	96	87	87
		LDA	86	81	94	83	70	93	77	87
	Nearest	**SVM**	85	82	95	**88**	**91**	**97**	**94**	87
		LDA	86	82	**97**	86	84	96	84	87
20% data	**k-means**	**SVM**	84	80	94	86	89	94	87	45
		LDA	86	75	94	75	68	75	76	78
	Nearest	**SVM**	85	74	95	**88**	**91**	**97**	92	74
		LDA	86	82	95	80	87	96	72	70

Table 3.7. Accuracies (in percentage) deduced by methods referenced in the first column given the eight databases. The on-line results are obtained with 20% of the learning set and $K = 250$ in the first seven databases and $K = 25$ in the last one. The blank cells are values not given in the original papers. "i.c" means "ill conditioned" (the learning method is not able to generate the Gaussian function). [A]:Neuhaus and Bunke (2005). [B]:Neuhaus and Bunke (2007). [C]:Cortés and Serratosa (2015). [D]:Algabli and Serratosa (2018a). [E]:Santacruz and Serratosa (2019).

		Letter High	**Letter Med**	**Letter Low**	**Boat**	**East Park**	**East South**	**Resid**	**Finger-print**
K-means	**SVM**	314'9	305'8	250'2	56'7	64'4	59'1	53'8	11'5
	LDA	1.249'9	1.112'2	1.080'2	72'5	77'4	80'6	76'3	15'7
Nearest	**SVM**	1.183'1	987'2	1.010'6	52'2	70'3	61'0	55'3	6'1
	LDA	164'2	215'5	214'6	43'4	55'8	51'0	46'1	18'2

Table 3.8. Run time in seconds of learning process introducing 20% of learning data set in each database. It has been set $K = 250$ in the first seven databases and $K = 25$ in the last one.

3.5 Conclusions

The on-line method presented in this chapter learns the edit costs of the *GED* embedding node-to-node mappings between graphs into an euclidean space previously defined in an off-line method.

This on-line method has the particularity that the learning method is limited to the applications where substitution costs are represented as weighted Manhattan distances and insertion and deletion costs are constants.

The method needs a parameter, K, that controls the amount of data to be kept during the learning process and a small validation data set to validate the values obtained in some steps of the learning process.

The experimental validation shows that the accuracy obtained with the learned parameters is similar to or higher than the obtained with the off-line methods, but the needed amount of data is considerably lower.

From a practical point of view, this method has the main advantage that the learned costs can be used each time the learning process is executed because the training stage is alternated to the matching stage.

4

Discrete parameter learning applied to Virtual Screening

4.1 Introduction

The high increase in chemical compounds data has created the need to develop computational tools to reduce the drug synthesis and drug test cycle runtimes. When activity data are analysed, these tools are required to generate new models for virtual screening techniques (Kubinyi et al. (2008); Bajorath (2001); Schneider et al. (2000)). In the drug discovery process, virtual screening is a common step in which computational techniques are used to search and filter chemical compounds in databases. Basically, there are two main types of methods in the virtual screening: ligand-based virtual screening (LBVS) (Cereto-Massagué et al. (2015)) and structure-based virtual screening (SBVS) (Heikamp and Bajorath (2013)). In this work, we focus only in LBVS applications. The idea of the LBVS method is to predict the unknown activity of new molecules (Sun (2008)) using the information about the known activity of some molecules. Specifically, their behaviour as ligands that bind to a receptor.

Some LBVS approaches are shape-based similarity (Kirchmair et al. (2009)), pharmacophore mapping (Sun (2008)), fingerprint similarity and machine learning methods (Melville et al. (2009)). According to Johnson and Maggiora (1990), structurally similar molecules are presumed to have similar activity properties. Then, in the context of LBVS methods, the chosen molecular similarity metric is important because it can determine the success of a virtual screening method to discover proper drug candidates. Various similarity methods are used in several applications (Bender and Glen (2004); Nikolova and Jaworska (2003); Willett (2004); Lajiness (1990); Willett (1987)).

The outline of this chapter is as follows. In Section 4.2, we present a background on representation and classification of molecules. In Section 4.3 we explain in detail the method based on the graph representation of molecules and the *GED*. In Section 4.4, we present the data sets and we discuss the results of the experiments. Finally, we present the general conclusions about the method in Section 4.5.

4.2 Background

To compute molecular similarity, it is possible to define a distance and define a descriptor representing the molecule. Hundreds of molecular descriptors have been reported in the literature (Xue and Bajorath (2000)). For instance, one-dimensional descriptors include general molecular properties, such as size, molecular weight, logP or dipole moment, or BCUT parameters (Menard et al. (1998); Pearlman and Smith (1999); Schnur (1999); Livingstone (2000)). Two-dimensional descriptors generate an array of representations of the molecules by simplifying the atomic information within them, such as 2D fingerprints (Barnard (1993); James and Weininger (1995); McGregor and Pallai (1997)). Finally, three-dimensional descriptors use 3D information, such as molecular volume (Güner (2000); Beno and Mason (2001)). Other existing methods, instead of representing molecules by an N-dimensional vector, use relational structures, such as trees (Rarey and Dixon (1998)) or graphs (Barker et al. (2006); Takahashi et al. (1992)). Regarding the molecule representation by graphs, some methods represent compounds using reduced graphs (Stiefl et al. (2006); Gillet et al. (2003, 1991); Fisanick et al. (1994)) and other ones, such as extended reduced graphs (ErGs) (Stiefl et al. (2006)). Reduced graphs group atomic sub-structures that have related features, e.g., pharmacophoric features, ring systems, hydrogen-bonding or others. Moreover, ErGs are an extension of reduced graphs that introduce some changes to better represent shape, size and pharmacophoric properties of the molecules. The method presented in Stiefl et al. (2006) has demonstrated its use as a powerful tool for virtual screening.

To perform reduced graph comparisons, three different similarity measures have been used: In Stiefl et al. (2006); Gillet et al. (2003); Barker et al. (2003), they map the reduced graphs into a 2D fingerprint. In Harper et al. (2004), they map reduced graphs into sets of shortest paths. Finally, in Garcia-Hernandez et al. (2019, 2020), they perform the comparison on the graphs using the Graph Edit Distance (GED). GED considers the distance between two graphs as the minimum cost of modifications required to transform one graph into another. Each modification can be one of the following six operations: insertion, deletion and substitution of both nodes and edges in the graph (Munkres (1957); Sanfeliu and Fu (1983b); Gao et al. (2010b)). The main goal of this chapter is to present an algorithm that learns the edit costs in the GED to improve the classification ratio returned by the system when the Harper costs were used.

In an initial paper, Garcia-Hernandez et al. (2019), the edit costs were imposed and extracted from Harper et al. (2004), given the chemical expertise of the authors and considering the different node and edge types. Later, in Garcia-Hernandez et al. (2020), authors presented an algorithm for optimising those edit costs based on minimising the distance between correctly classified molecules and maximising the distance between incorrectly classified molecules. That work was inspired in a similar one carried out by Birchall et al. (2006), in which the authors optimise the transformation costs of a String Edit Distance method to compare molecules using reduced graphs.

The main problem of the algorithm in Garcia-Hernandez et al. (2020) was the huge computational cost, which depends on the number of edit costs to be optimised. Thus, for practical reasons, in the experimental section in Garcia-Hernandez et al. (2020), they presented four experiments, in which only one edit cost was optimised in each experiment. They imposed the other costs (126 in total) to be the ones defined in Harper et al. (2004). In contrast, starting from the costs defined by Harper et al. (2004), the method presented in this chapter learns all the edit costs of the GED to compare molecules with a lower computational cost obtaining higher classification ratios in the ligand-based screening application, as shown in the experimental section (4.4).

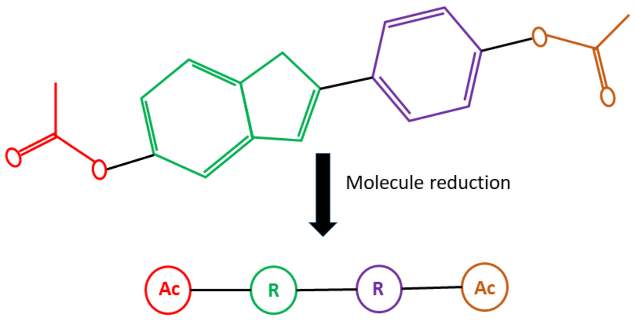


Figure 4.1. Example of molecule reduction using ErG. The original molecule is on the top and its ErG representation is below. Elements of the same colour on the top are reduced to nodes on the ErG. R: Ring system, Ac: Acyclic components.

4.2.1 Molecular representation

Reduced graphs are compact representations of chemical compounds, in which the main information is condensed in feature nodes to give abstractions of the chemical structures. Different versions of reduced graphs have been presented (Barker et al. (2003); Gillet et al. (1991); Harper et al. (2004); Barker et al. (2006); Stiefl et al. (2006)) and they depend on the features that they summarise or the use that is given to them. In the virtual screening context, the structures are reduced to track down features or sub-structures that have the potential to interact with a specific receptor and, at the same time, try to keep the topology and spatial distribution of those features. Figure 4.1 presents an example of molecule reduction.

4.2.2 Classification of molecules

Once the molecules are represented as ErGs, we can compare them by means of the *Graph Edit Distance (GED)* (Solé et al. (2012); Serratosa (2021)). The *GED* (see Section 2.2) is defined as the minimum cost of transformations required to convert one graph into the other. Thus, in our application, it is the cost to transform an ErG into the other one. To classify a molecule, we apply the Nearest Neighbour (NN) strategy that consists of calculating the *GED* between this molecule and the other ones, of which class we know, and predicting its class (active or inactive) to be the class of the nearest molecule. In the case the molecule is equidistant from more than one classified molecule, the method arbitrarily selects one of the closest

Node attributes

Attribute	Description
[0]	hydrogen-bond donor
[1]	hydrogen-bond acceptor
[2]	positive charge
[3]	negative charge
[4]	hydrophobic group
[5]	aromatic ring system
[6]	carbon link node
[7]	non-carbon link node
[0, 1]	hydrogen-bond donor + hydrogen-bond acceptor
[0, 2]	hydrogen-bond donor + positive charge
[0, 3]	hydrogen-bond donor + negative charge
[1, 2]	hydrogen-bond acceptor + positive charge
[1, 3]	hydrogen-bond acceptor + negative charge
[2, 3]	positive charge + negative charge
[0, 1, 2]	hydrogen-bond donor + hydrogen-bond acceptor + positive charge

Edge attributes

Attribute	Description
-	single bond
$=$	double bond
$\equiv$	triple bond

Table 4.1. Node and edge attributes description in an ErG.

molecules.

Edit costs have been introduced to quantitatively evaluate each edit operation. The aim of the edit costs is to designate a coherent penalty to the transformation in proportion to the extent to which it modifies the transformation sequence. For instance, when ErGs are compared, it makes sense that the cost of substituting a ”hydrogen-bond donor” feature with a joint ”hydrogen-bond donor-acceptor” feature be less heavily penalized than the cost of substituting a ”hydrogen-bond donor” feature with an ”aromatic ring” system. Similarly, inserting a single bond should have a lower penalization cost than inserting a double bond, and so on. In a previous work (Garcia-Hernandez et al. (2019)), the edit costs proposed by Harper et al. (2004) were used. The node and edge descriptions are shown in Table 4.1, and the specific costs proposed by Harper et al. (2004) are exposed in Tables 4.2 and 4.3.

The final edit cost for a given transformation sequence is obtained by adding up all of the individual edition costs. Figure 4.2 shows a schematic example of a transformation of a molecule G_1 into another one, G_2. As we can see, the executed operations in this transformation are: a deletion of node type [1], a deletion of a simple edge, an insertion of node type [5], an insertion of a simple edge a substitution of node type [7] by node of type [2], and a substitution of a simple edge with a double edge. If we sum the values of Harper costs associated with these operations in Tables 4.2 and 4.3, we obtain that the cost of this transformation equals: $2 + 0 + 2 + 0 + 3 + 3 = 10$.

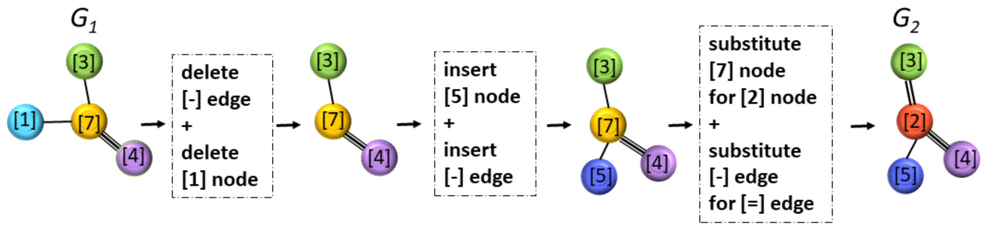


Figure 4.2. Transformation sequence from graph G_1 to graph G_2.

Since several transformation sequences can be applied to transform a graph into another one, the *GED* resulting for any pair of graphs is defined as the minimum cost under all those possible transformation sequences. Usually, the final distance is normalized according to the number of nodes in both graphs being compared. This is performed in order to make the measure independent of the size of the graphs.

More formally, we define the *GED* as follows,

$$GED(G_a, G_b, C_1, \ldots, C_n) = \min_{\{N_i : i=1,\ldots,n\}} \frac{C_1 N_1 + \ldots + C_n N_n}{L} \tag{4.1}$$

where C_t is the imposed cost of the t^{th} edit operation on nodes and edges, and N_t is the number of times this edit operation has been applied. Moreover, the combination of N_1, $N_2, \ldots$ is restricted to be one that transforms G_a into G_b. Finally, L is the sum of the number of nodes of both graphs, and n is the number of different edit operations on nodes and edges.

Several GED computational methods have been proposed during the last three decades, they can be classified into two groups: those returning the exact value of the GED in the exponential computational cost with respect to the number of nodes Blumenthal and Gamper (2018), and those returning an approximation of the GED in the polynomial cost Serratosa (2014a); Santacruz and Serratosa (2018a); Serratosa (2014b, 2015). These two groups of GED computational methods have been widely studied Conte et al. (2004); Vento (2015). In our experiments, we used the fast bipartite graph matching method Serratosa (2014a) (polynomial computational cost), although our learning method is independent of the matching algorithm.

Substitution costs for nodes

	[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[0, 1]**	**[0, 2]**	**[0, 3]**	**[1, 2]**	**[1, 3]**	**[2, 3]**	**[0, 1, 2]**
[0]	0	2	2	2	2	2	2	3	1	1	1	2	2	2	1
[1]	2	0	2	2	2	2	2	3	1	2	2	1	1	2	1
[2]	2	2	0	2	2	2	2	3	2	1	2	1	2	1	1
[3]	2	2	2	0	2	2	2	3	2	2	1	2	1	1	2
[4]	2	2	2	2	0	2	2	3	2	2	2	2	2	2	2
[5]	2	2	2	2	2	0	2	3	2	2	2	2	2	2	2
[6]	2	2	2	2	2	2	0	3	2	2	2	2	2	2	2
[7]	3	3	3	3	3	3	3	0	3	3	3	3	3	3	3
[0, 1]	1	1	2	2	2	2	2	3	0	2	2	2	2	2	2
[0, 2]	1	2	1	2	2	2	2	3	2	0	2	2	2	2	2
[0, 3]	1	2	2	1	2	2	2	3	2	2	0	2	2	2	2
[1, 2]	2	1	1	2	2	2	2	3	2	2	2	0	2	2	2
[1, 3]	2	1	2	1	2	2	2	3	2	2	2	2	0	2	2
[2, 3]	2	2	1	1	2	2	2	3	2	2	2	2	2	0	2
[0, 1, 2]	1	1	1	2	2	2	2	3	2	2	2	2	2	2	0

Insertion/Deletion costs for nodes

	[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[0, 1]**	**[0, 2]**	**[0, 3]**	**[1, 2]**	**[1, 3]**	**[2, 3]**	**[0, 1, 2]**
insert	2	2	2	2	2	2	1	1	2	2	2	2	2	2	2
delete	2	2	2	2	2	2	1	1	2	2	2	2	2	2	2

Table 4.2. Substitution, insertion and deletion costs for nodes proposed by Harper et al. (2004).

Substitution costs for edges

	-	**=**	**≡**
-	0	3	3
=	3	0	3
≡	3	3	0

Insertion/Deletion costs for edges

	-	**=**	**≡**
insert	0	1	1
delete	0	1	1

Table 4.3. Substitution, insertion and deletion costs for edges proposed by Harper et al. (2004).

Initially, the edit costs were manually set in a trial and error process considering the application at hand Harper et al. (2004); Garcia-Hernandez et al. (2019). (As previously commented, Tables 4.2 and 4.3 show their edit cost proposal.) Nevertheless, there has been a tendency to automatically learn these costs since it has been seen that a proper tuning of them is crucial to achieve good classification ratios in virtual screening Garcia-Hernandez et al. (2020) and other applications Rica et al.

(2021); Conte and Serratosa (2020b); Santacruz and Serratosa (2020); Algabli and Serratosa (2018b); Cortés and Serratosa (2016). In Garcia-Hernandez et al. (2020), authors presented a learning algorithm that is forced to learn only one edit cost at once due to runtime restrictions. Thus, they perform four different experiments on the same data as Garcia-Hernandez et al. (2019) in which they use all the costs of Garcia-Hernandez et al. (2019) except the one that is learned. These experiments are:

C1: Learning the deletion/insertion cost of the carbon link ([6]).

C2: Learning the cost of substituting a carbon link node ([6]) with an aromatic ring system ([5]).

C3: Learning the insertion/deletion cost of the bond edge ([-]).

C4: Learning the substitution cost between a single bond edge ([-]) and a double bond edge ([=]).

Table 4.4 shows their learnt costs.

	Type of cost	**CAPST**	**DUD-E**	**GLL&GDD**	**MUV**	**NRLiSt_BDB**	**ULS-UDS**
C1	Ins/Del [6]	0.000002	0.005	0.014	0.490	0.012	0.115
C2	Subs [5] by [6]	0.013	0.145	0.333	0.867	0.104	0.500
C3	Ins/Del [-]	0.004	0.001	0.003	0.327	0.003	0.011
C4	Subs [-] by [=]	0.017	0.186	0.206	1.005	0.024	0.607

Table 4.4. Costs obtained in Garcia-Hernandez et al. (2020).
Each row corresponds to one of their experiments C1-C4.

The next section presents our method, which has the advantage of learning the whole set of edit costs at once.

4.3 The proposed method

We explain the proposed method in the next three subsections. The first one explains the classification of compounds based on structural information; in the second one, we explain the learning algorithm; and in the third one, we detail the code of the algorithm.

4.3.1 The learning method

We present an iterative algorithm, in which, in each iteration, the NN strategy is applied and the initial edit costs are modified such that one molecule that has been incorrectly classified becomes correctly classified. Modifying the edit costs could cause other incorrectly classified molecules to also be properly classified, but, unfortunately, some other ones that were properly classified become incorrectly classified. This is the reason why we want to generate the minimum modification on the edit costs. To do so, the selected molecule is the one that it is easier to move from the incorrectly classified ones to the correctly classified ones. In the next paragraphs, our learning algorithm is explained in detail and it is summarised in Subsection 4.3.2.

Let G_j be a molecule in the learning set that has been incorrectly classified using the NN strategy and the current costs $C_1, \dots, C_n$. We define D_j as the minimal GED between G_j and all the molecules but restricted to be the ones that have a different class:

$$D_j = \min_q GED(G_j, G_q, C_1, \dots, C_n) \text{ , where class}(G_q) \neq \text{class}(G_j). \tag{4.2}$$

Moreover, we define D'_j as the minimal GED between G_j and all the molecules of the learning set but restricted to be the ones that have the same class:

$$D'_j = \min_p GED(G_j, G_p, C_1, \dots, C_n), \text{ where class}(G_p) = \text{class}(G_j) \tag{4.3}$$

Since G_j is incorrectly classified, we can confirm that $D'_j > D_j$. Figure 4.3 schematically shows this situation. It turns out that G_j and G_q belong to different classes even though the distance between them is smaller than the distance between G_j and its closest molecule that has the same class, G_p.

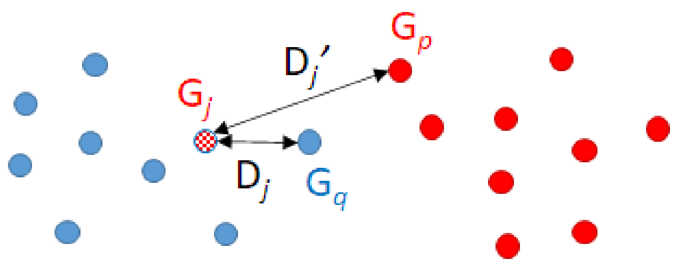


Figure 4.3. Classification of molecule G_j. The true classes are in solid colours. G_j is classified in the wrong class (blue), but the correct class is the red one. The distance between G_j and G_q is lower than the distance between G_j and G_p.

The main idea of our method is to permute D_j' and D_j, modifying the edit costs. With this exchange, we achieve a lower distance between G_j and the molecule of its same class (G_p) than the distance between G_j and the molecule with different classes (G_q). Thus, G_j will be correctly classified. However, considering that adapting these distances affects all the molecules' classifications, we select a molecule G_i among the incorrectly-classified ones, $\{G_j \ s.t. \ D_j' > D_j\}$, which satisfies that the difference of the distances $D_j' - D_j$ is the minimum one, as shown in Equation (4.4). Note that in Equation (4.4), all the values of $D_j' - D_j$ are always positive because $D_j' > D_j$ by definition of G_j.

$$G_i = arg \min_{\{G_j \ s.t. \ D_j' > D_j\}} (D_j' - D_j) \tag{4.4}$$

Figure 4.4 shows this idea. However, what is crucial to understand is that this modification is performed in the distances since the molecule representations are not modified. Furthermore, this is carried out by modifying the edit costs. Thus, the strategy is to define the new edit costs such that D_i' becomes D_i and vice versa.

The rest of this section is devoted to explaining how to modify the edit costs.

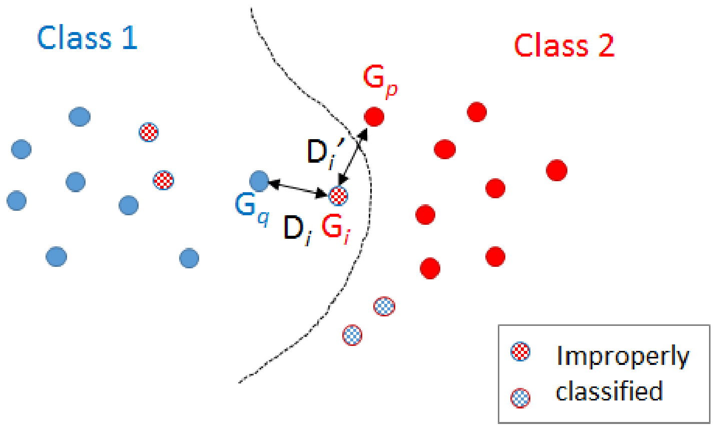


Figure 4.4. Stripped molecules have been improperly classified using NN strategy. G_i is the one that minimises $D_j' - D_j$ being $D_j' > D_j$.

Considering Equation (4.1), the distance is composed of edit costs $C_1, \ldots, C_n$ and the number of times the specific edit operations have been taken $N_1, \ldots, N_n$. Our method modifies the edit costs without altering the number of operations $N_1, \ldots, N_n$.

Thus, we define D_i and D_i' as follows:

$$\begin{aligned} D_i &= \frac{C_1 N_1 + \ldots + C_n N_n}{L} \\ D_i' &= \frac{C_1 N_1' + \ldots + C_n N_n'}{L'} \end{aligned} \tag{4.5}$$

Then, we exchange the distances D_i and D_i' and modify the edits costs by adding new terms:

$$\begin{aligned} D_i &= \frac{(C_1 + \alpha_1')N_1' + \ldots + (C_n + \alpha_n')N_n'}{L'} \\ D_i' &= \frac{(C_1 + \alpha_1)N_1 + \ldots + (C_n + \alpha_n)N_n}{L} \end{aligned} \tag{4.6}$$

Note that these new terms $\alpha_1, \ldots, \alpha_n$ and also $\alpha_1', \ldots, \alpha_n'$ are defined such that the new value of D_i is D_i' instead of D_i and vice versa. Moreover, the edit costs $C_1, \ldots, C_n$ are the same in both expressions. We proceed to explain below how to deduce the terms $\alpha_1, \ldots, \alpha_n$ and also $\alpha_1', \ldots, \alpha_n'$.

From Equation (4.6), we obtain:

$$\begin{aligned} D_i &= \frac{C_1N'_1 + \ldots + C_nN'_n}{L'} + \frac{\alpha'_1N'_1 + \ldots + \alpha'_nN'_n}{L'} \\ D'_i &= \frac{C_1N_1 + \ldots + C_nN_n}{L} + \frac{\alpha_1N_1 + \ldots + \alpha_nN_n}{L} \end{aligned} \tag{4.7}$$

We observe that the first terms in both expressions are D'_j and D_j, respectively:

$$\begin{aligned} D_i &= D'_i + \frac{\alpha'_1N'_1 + \ldots + \alpha'_{n'}N'_n}{L'} \\ D'_i &= D_i + \frac{\alpha_1N_1 + \ldots + \alpha_nN_n}{L} \end{aligned} \tag{4.8}$$

By regrouping the terms again, we have:

$$\begin{aligned} D_i - D'_i &= \frac{\alpha'_1N'_1 + \ldots + \alpha'_{n'}N'_n}{L'} \\ D'_i - D_i &= \frac{\alpha_1N_1 + \ldots + \alpha_nN_n}{L} \end{aligned} \tag{4.9}$$

Furthermore, finally, we divide by $D_i - D'_i$ and $D'_i - D_i$ in each expression to arrive at the following normalised expressions:

$$\begin{aligned} 1 &= \frac{\alpha'_1N'_1}{(D_i - D'_i)L'} + \ldots + \frac{\alpha'_nN'_n}{(D_i - D'_i)L'} \\ 1 &= \frac{\alpha_1N_1}{(D'_i - D_i)L} + \ldots + \frac{\alpha_nN_n}{(D'_i - D_i)L} \end{aligned} \tag{4.10}$$

Note that, as commented in the definition of the GED, not all of the edit operations are used to transform a molecule into another. These edit operations are the ones that $N_t = 0$ or $N'_t = 0$. Because of this, in Equation (4.10), there are some addends that are null. We use m and m' to denote the number of edit operations that have been used, that is, the ones that $N_t \neq 0$ or $N'_t \neq 0$, respectively.

We want to deduce $\alpha_1, \alpha_2, \ldots$ and also $\alpha'_1, \alpha'_2, \ldots$ such that Equation (4.10) is fulfilled. The easiest way is to impose that each non-null term in these expressions equal $1/m'$ or $1/m$, respectively. Then, we achieve the following expressions,

$$\begin{aligned} 1/m' &= \frac{\alpha'_t N'_t}{(D_i - D'_i)L'} \text{ being } N'_t > 0 \\ 1/m &= \frac{\alpha_t N_t}{(D'_i - D_i)L} \text{ being } N_t > 0 \end{aligned} \tag{4.11}$$

From the previous expressions, we arrive at the definitions of α_t that allow the modification from D_i to D'_i. Moreover, we also arrive at the definitions of $\alpha'_{t'}$ that allow the modification from D'_i to D_i.

$$\begin{aligned} \alpha'_t &= \frac{(D_i - D'_i)L'}{m' N'_{t'}}, N'_t > 0 \\ \alpha_t &= \frac{(D'_i - D_i)L}{m N_t}, N_t > 0 \end{aligned} \tag{4.12}$$

Note that considering Equations (4.5), (4.6) and (4.12), we have, on one hand, that the new costs $\overline{C}_t = C_t + \alpha_t$ and, on the other hand, that $\overline{C}_t = C_t + \alpha'_t$. Since it may happen that $\alpha_t \neq \alpha'_t$, we assume the average option is the best choice when both weights are computed,

$$\overline{C}_t = \begin{cases} C_t + \frac{\alpha_t + \alpha'_t}{2}, \text{ if } N_t > 0 \text{ and } N'_t > 0 \\ C_t + \alpha_t, \text{ if } N_t > 0 \text{ and } N'_t = 0 \\ C_t + \alpha'_t, \text{ if } N_t = 0 \text{ and } N'_t > 0 \\ C_t, \text{ if } N_t = 0 \text{ and } N'_t = 0 \end{cases} \tag{4.13}$$

In the next subsection, we present our algorithm.

4.3.2 Algorithm

Algorithm 2 is an iterative process that updates the edit costs in each iteration to correct the classification of one selected molecule. The updated costs are used in the next iteration to classify all the molecules again. Then, it selects another molecule incorrectly classified and modifies the costs again to correct it.

Algorithm 2 Discrete costs learning.

```
Input ( Learning Set, Initial edit costs, $Max\_Iter$ )

Output ( Learnt edit costs )

1. Initialise:
   $iter = 1$.
   $C_1,\ldots, C_n$ = Initial edit costs.
   While $iter \leq Max\_Iter$:
2. Classify all molecules with nearest neighbour and $GED$
   (Equation (4.1) using $C_1, \ldots, C_n$).
3. Compute $D_j$ and $D'_j$:
   (Equations (4.2) and (4.3)) for all $G_j$ incorrectly classified.
4. Deduce $G_i$ (Equation (4.4)).
5. Compute $\alpha_t$, $t = 1, \ldots, m$ and $\alpha'_t$, $t = 1, \ldots, m'$: (Equation 4.12).
6. Compute $\overline{C}_1,\ldots, \overline{C}_n$ (Eq. 4.13).
7. Update costs: $C_t = \overline{C}_t, t = 1, \ldots, n$.
8. $iter = iter + 1$.
   End While

End Algorithm
```

4.4 Experiments

4.4.1 Datasets

To validate this method, we have used six available public datasets: ULS-UDS (Xia et al. (2015)), GLL&GDD (Gatica and Cavasotto (2011)), CAPST (Sanders et al. (2012)), DUD-E (Mysinger et al. (2012)), NRLiSt-BDB (Lagarde et al. (2014)) and MUV (Rohrer and Baumann (2009)). All these datasets had been formatted and standardized by the LBVS benchmarking platform developed by Skoda and Hoksza (Skoda and Hoksza (2017)). The datasets are composed of various groups of active and inactive molecules arranged according to the purpose of a target. Each group is split in two halves, the test and train sets, which are required when using machine learning methods. The train set is used to optimize the transformation costs, and the test set is used to evaluate the classification ratio. The targets of the data sets are shown in Table 4.5. In our experimentation, we have taken a subset of the first 100 active molecules and 100 of the first inactive molecules per target. Some datasets have less than 100 active molecules; in this case, all active molecules are taken and also the same number of inactivemolecules.

4.4.2 Results

Table 4.6 shows the classification ratios obtained in each dataset using different edit cost configurations, algorithms and initialisations. The first row corresponds to the accuracies obtained with the costs proposed by Harper et al. (2004), the second row corresponds to the accuracies deduced by setting all the costs to 1 (no learning algorithm). The next four rows correspond to the accuracies obtained using the costs deduced in Garcia-Hernandez et al. (2020) in their four experiments (C1, C2, C3 and C4). Finally, the last two rows present the accuracies obtained by our method: the first row by initialising the algorithm by the Harper costs and the second one by initialising all the costs to 1. We note the used costs are the mean of the learned costs in all the databases, and our algorithm performed 50 iterations.

We realise that in all the datasets, except for MUV and ULS-UDS, our costs with Harper initialisation obtained the highest classification ratios. In these two datasets, the best accuracy is obtained by Harper costs. Note that our method initialised by all-ones costs returns lower accuracies than our method initialised by Harper costs, except for the ULS-UDS dataset. This behaviour makes us think that the initialisation point is very important in this type of algorithm. Another highlight is that we have achieved better accuracies than the four experiments presented by Garcia-Hernandez et al. (2020) in all the tests. In the ULS-UDS dataset, our method returns close accuracy to the Harper costs. Nevertheless, that is not the case for MUV dataset. To deeply analyse this behaviour, we have computed the accuracy using the costs learned by only the MUV targets. In this case, the accuracy is 64.9%, which is significantly lower than using mean costs. This is not the normal behaviour in learning algorithms since while conducting specific learning, the classification ratio tends to increase. We think there are other reasons for this abnormal behaviour: one could be the small size of this dataset and the other the separability between ligands and decoys in MUV is low, which makes our algorithm not to converge to a proper solution.

In Figure 4.5, we present the classification ratio obtained in the 127 targets in the six datasets. At a glance, we realise that our method achieves most of the highest accuracies in all the targets in CAPST, DUD-E, GLL&GDD and NRLiSt-BDB databases. Specifically, we point out targets from 19 to 31 in the GLL&GDD dataset where the other cost combinations have very low accuracies while our method achieves much higher results. We observe that targets in the datasets MUV and ULS-UDS, in which our method does not return the highest accuracies, have a high variability because the same costs produce very different results.

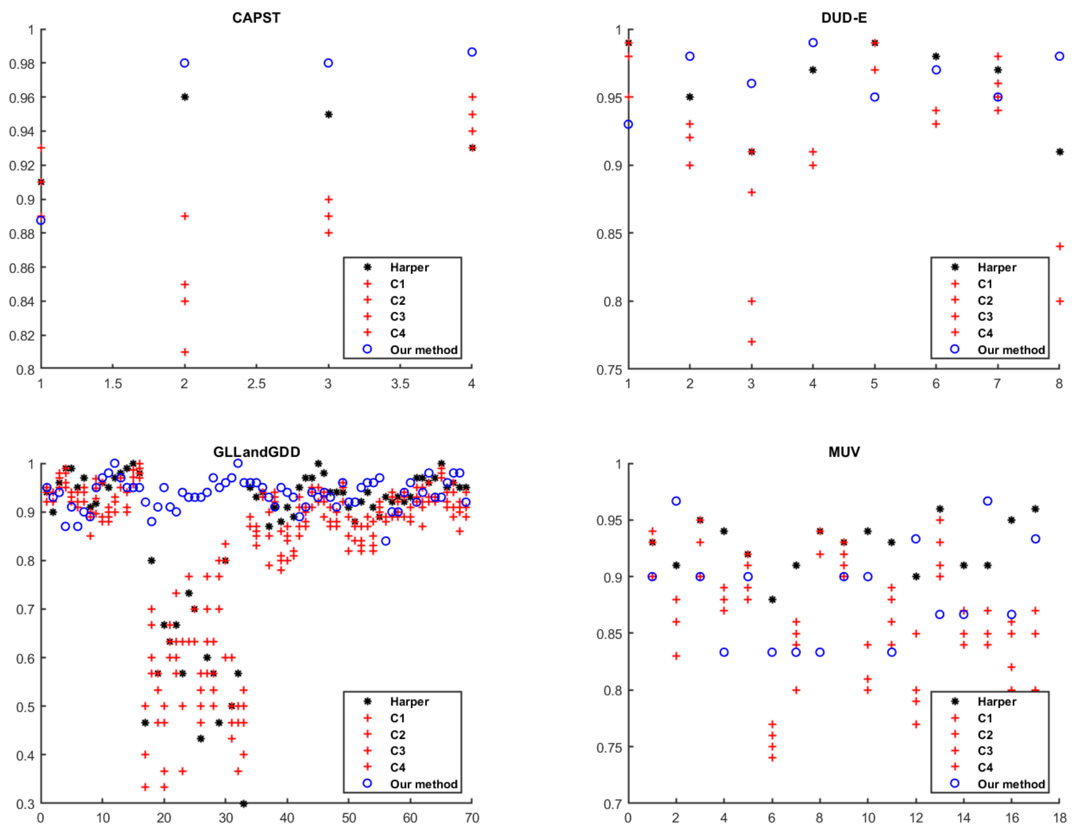


Figure 4.5. *Cont.*

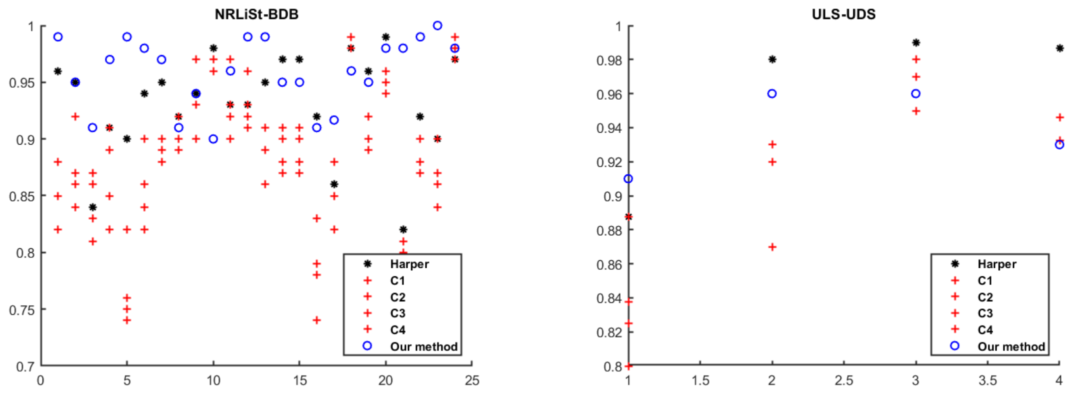


Figure 4.5. Classification ratio in the test set over the 127 targets available in the six datasets. The horizontal axis represents the index of the targets presented in Table 4.5.

Note that in Garcia-Hernandez et al. (2020), authors computed a cost per each of the six datasets and each target. Conversely, we learn the edit costs given the six datasets at once. In general, using several datasets at once makes the learnt parameters less specific for the application at hand, and thus, the classification ratios tend to decrease. In spite of this possible disadvantage, our method returns better classification ratios than the one in Garcia-Hernandez et al. (2020) in all the datasets. Figure 4.6 shows the percentage of times that each cost configuration obtains the highest classification ratio taking into account all the 127 targets, given the four configurations proposed in Garcia-Hernandez et al. (2020), one configuration proposed in Harper et al. (2004) and our deduced configuration. The presented method obtains the best classification ratio the highest number oftimes.

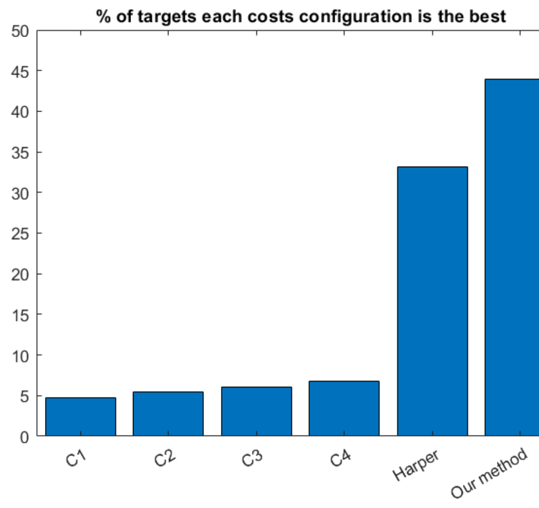


Figure 4.6. Percentage of times that each set of costs returns the best classification ratio.

Tables 4.7 and 4.8 show the learned edit costs for nodes and edges, respectively. In bold are the ones that are different to the ones proposed by Harper et al. (2004). As we can see, the results are very similar to Harper costs because the method introduces a very small modification in each step. In addition, there are many costs that have not been modified. This is because these costs were not involved in the modifications of molecules that are improperly classified, minimising $D_i' - D_i$.

Data set	**Used targets**
CAPST	CDK2, CHK1, PTP1B, UROKINASE
DUD-E	COX2, DHFR, EGFR, FGFR1, FXA, P38, PDGFRB, SRC, AA2AR
GLL&GDD	5HT1A_Agonist, 5HT1A_Antagonist, 5HT1D_Agonist, 5HT1D_Antagonist, 5HT1F_Agonist, 5HT2A_Antagonist, 5HT2B_Antagonist, 5HT2C_Agonist, 5HT2C_Antagonist, 5HT4R_Agonist, 5HT4R_Antagonist, AA1R_Agonist, AA1R_Antagonist, AA2AR_Antagonist, AA2BR_Antagonist, ACM1_Agonist, ACM2_Antagonist, ACM3_Antagonist, ADA1A_Antagonist, ADA1B_Antagonist, ADA1D_Antagonist, ADA2A_Agonist, ADA2A_Antagonist, ADA2B_Agonist, ADA2B_Antagonist, ADA2C_Agonist, ADA2C_Antagonist, ADRB1_Agonist, ADRB1_Antagonist, ADRB2_Agonist, ADRB2_Antagonist, ADRB3_Agonist, ADRB3_Antagonist, AG2R_Antagonist, BKRB1_Antagonist, BKRB2_Antagonist, CCKAR_Antagonist, CLTR1_Antagonist, DRD1_Antagonist, DRD2_Agonist, DRD2_Antagonist, DRD3_Antagonist, DRD4_Antagonist, EDNRA_Antagonist, EDNRB_Antagonist, GASR_Antagonist, HRH2_Antagonist, HRH3_Antagonist, LSHR_Antagonist, LT4R1_Antagonist, LT4R2_Antagonist, MTR1A_Agonist, MTR1B_Agonist, MTR1L_Agonist, NK1R_Antagonist, NK2R_Antagonist, NK3R_Antagonist, OPRD_Agonist, OPRK_Agonist, OPRM_Agonist, OXYR_Antagonist, PE2R1_Antagonist, PE2R2_Antagonist, PE2R3_Antagonist, PE2R4_Antagonist, TA2R_Antagonist, V1AR_Antagonist, V1BR_Antagonist, V2R_Antagonist
MUV	466, 548, 600, 644, 652, 689, 692, 712, 713, 733, 737, 810, 832, 846, 852, 858, 859
NRLiSt_BDB	AR_Agonist, AR_Antagonist, ER_Alpha_Agonist, ER_Alpha_Antagonist, ER_Beta_Agonist, FXR_Alpha_Agonist, GR_Agonist, GR_Antagonist, LXR_Alpha_Agonist, LXR_Beta_Agonist, MR_Antagonist, PPAR_Alpha_Agonist, PPAR_Beta_Agonist, PPAR_Gamma_Agonist, PR_Agonist, PR_Antagonist, PXR_Agonist, RAR_Alpha_Agonist, RAR_Beta_Agonist, RAR_Gamma_Agonist, RXR_Alpha_Agonist, RXR_Alpha_Antagonist, RXR_Gamma_Agonist, VDR_Agonist
ULS-UDS	5HT1F_Agonist, MTR1B_Agonist, OPRM_Agonist, PE2R3_Antagonist

Table 4.5. Data sets used for the experiments. Each data set on the left contains the targets on the right.

	CAPST	**DUD-E**	**GLL&GDD**	**MUV**	**NRLiSt_BDB**	**ULS-UDS**	**Mean**
Harper	93,75	95,88	85,68	**92,76**	93,17	**96,10**	92,89
1s	92,93	91,25	93,03	56,01	94,75	92,94	86,82
C1	89,25	92,63	82,47	86,06	88,58	89,65	88,11
C2	89,75	91,13	82,51	87,35	88,21	91,69	88,44
C3	91,25	91,25	83,25	86,65	87,75	92,34	88,75
C4	89,50	90,88	82,43	86,00	89,92	92,59	88,55
Our method (Harper init.)	**95,85**	**96,38**	**93,67**	88,63	**95,90**	94,00	**94,07**
Our method (1s init.)	88,15	93,50	93,30	61,76	94,98	95,25	87,82

Table 4.6. Accuracy (%) obtained in each data set. In bold, the highest ones. The last column shows the mean accuracy.

Substitution costs for nodes

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	**[0, 1]**	**[0, 2]**	**[0, 3]**	**[1, 2]**	**[1, 3]**	**[2, 3]**	**[0, 1, 2]**
[0]	0	**1,99**	**2,02**	2,00	**1,99**	**2,04**	**2,05**	3,00	**1,06**	**0,99**	1,00	2,00	2,00	2,00	**0,97**
[1]	**1,99**	0	2,00	2,00	**1,98**	**1.99**	**1,96**	3,00	**1,02**	**1,99**	2,00	**1,02**	1,00	2,00	**1,04**
[2]	**2,02**	2,00	0	2,00	2,00	2,00	**1,99**	3,00	2,00	**0,99**	2,00	1,00	2,00	1,00	**0,98**
[3]	2,00	2,00	2,00	0	2,00	2,00	**2,05**	3,00	**1,99**	2,00	1,00	2,00	1,00	1,00	2,00
[4]	**1,99**	**1,98**	2,00	2,00	0	**1,99**	**2,01**	3,00	**2,01**	**2,01**	2,00	2,00	2,00	2,00	2,00
[5]	**2,04**	**1,99**	2,00	2,00	**1,99**	0	**1,99**	3,00	**1,96**	**1,96**	2,00	2,00	2,00	2,00	**2,02**
[6]	**2,05**	**1,96**	**1,99**	**2,05**	**2,01**	**1,99**	0	3,00	2,00	**2,01**	2,00	2,00	2,00	2,00	**1,98**
[7]	3,00	3,00	3,00	3,00	3,00	3,00	3,00	0	3,00	3,00	3,00	3,00	3,00	3,00	3,00
[0, 1]	**1,06**	**1,02**	2,00	**1,99**	**2,01**	**1,96**	2,00	3,00	0	**2,02**	2,00	2,00	2,00	2,00	**2,01**
[0, 2]	**0,99**	**1,99**	**0,99**	2,00	**2,01**	**1.96**	**2.01**	3,00	**2.02**	0	2,00	2,00	2,00	2,00	2,00
[0, 3]	1,00	2,00	2,00	1,00	2,00	2,00	2,00	3,00	2,00	2,00	0	2,00	2,00	2,00	2,00
[1, 2]	2,00	**1,02**	1,00	2,00	2,00	2,00	2,00	3,00	2,00	2,00	2,00	0	2,00	2,00	2,00
[1, 3]	2,00	1,00	2,00	1,00	2,00	2,00	2,00	3,00	2,00	2,00	2,00	2,00	0	2,00	2,00
[2, 3]	2,00	2,00	1,00	1,00	2,00	2,00	2,00	3,00	2,00	2,00	2,00	2,00	2,00	0	2,00
[0, 1, 2]	**0,97**	**1,04**	**0,98**	2,00	2,00	**2,02**	**1,98**	3,00	**2,01**	2,00	2,00	2,00	2,00	2,00	0

Insertion/Deletion costs for nodes

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	**[0, 1]**	**[0, 2]**	**[0, 3]**	**[1, 2]**	**[1, 3]**	**[2, 3]**	**[0, 1, 2]**
insert	**1,95**	**1,98**	2,00	2,00	**1,99**	**1,89**	**0,97**	1,00	**2,03**	**2,02**	2,00	**1,99**	2,00	2,00	**1,96**
delete	**1,95**	**1,98**	2,00	2,00	**1,99**	**1,89**	**0,97**	1,00	**2,03**	**2,02**	2,00	**1,99**	2,00	2,00	**1,96**

Table 4.7. Substitution, insertion and deletion costs of nodes obtained with our method. In bold, the ones that are different from Table 4.2.

Substitution costs for edges

	-	$=$	$\equiv$
-	0	3,00	3,00
$=$	3,00	0	3,00
$\equiv$	3,00	3,00	0

Insertion/Deletion costs for edges

	-	$=$	$\equiv$
insert	0	**1,02**	1,00
delete	0	**1,02**	1,00

Table 4.8. Substitution, insertion and deletion costs of edges obtained with our method. In bold, the ones that are different from Table 4.3.

4.5 Conclusions

In some ligand-based virtual screening (LBVS) methods, molecules are represented by extended reduced graphs. In this case, the Graph Edit Distance (GED) can be used to compute the dissimilarity between molecules.

The method presented in this chapter is an iterative method that, in each iteration, modifies the costs of the GED to properly classify only one molecule that had been incorrectly classified in the previous step. While updating the costs, other improperly classified molecules can be properly classified, but other correctly classified could become incorrectly classified. This is the reason why the convergence of the method is not guaranteed. To reduce this effect, the algorithm selects a molecule that requires the minimum modification of the costs with the aim of slightly moving to the best solution.

The method requires the initialization of the edit costs. In the experimental section they have been tuned by aleatory costs and by the costs proposed by Harper. In all the tests, the highest accuracies appear while initialising the costs by the Harper proposal, what reveals that the classification accuracy is highly dependent on the edit costs because a slight modification of the costs can make the classification to be different.

5

P&ID: Automatic validation based on Graphs

5.1 Introduction

Piping and Instrumentation Diagrams (P&IDs) are commonly used for representing the structure and functionality of Oil & Gas facilities such as oil rigs and plants. These facilities are huge and composed of thousands of electric, electronic or mechanical components that are connected by a vast network of pipelines. In the past, P&IDs were manually drawn on paper or by means of tools which are incompatible with modern software. Consequently, printed handbooks composed of thousands of pages were required to depict them. Currently, these complex engineering drawings are generated by means of computer-aided design (CAD) tools.

In the last years, there has been a tendency to migrate printed P&IDs towards a digital environment. Different techniques have been developed and the final result has always to be validated by an expert to guarantee that there are no errors in the final document. This chapter explains a method to reduce the effort of engineers to validate the automatically generated CAD models keeping the zero-error aim. The main idea is that the engineer does not need to check the whole generated diagram, but only some highlighted components with chance of having been incorrectly identified.

The chapter is structured as follows. Section 5.2 presents the current state of the art methodology used to transform P&IDs into digital assets. Section 5.3 examines the necessary concepts and presents the methodology aimed at aiding on the human validation of the digitised asset. Section 5.4 presents the experiments carried out to validate the proposed model. Finally, Section 5.5 is reserved for conclusions.

5.2 Background

Recently, there has been an increased effort to join forces between the research community and industrial partners [1] to collaborate in the digitalization of printed engineering drawings. [2] This has been tested in the past by numerous authors for different types of printed assets, such as mechanical drawings (Vaxiviere and Tombre (1992)), electrical drawings (Yu et al. (1997)), telephone manholes (Arias et al. (1995)), sensor-equipment diagrams (Moreno-García (2018)), P&IDs (Howie et al. (1998)), (Tan et al. (2016)), (Moreno-García et al. (2017)), (Rahul et al. (2019)), (Rantala et al. (2019)), (Kang et al. (2019)), even on maps (Cao and Tan (2002)) and musical scores (Calvo-Zaragoza et al. (2018)). The process of migration paper P&IDs to CAD models is extremely complex due to the quality of the scanned papers, and the amount and variability of the involved components. Consequently, the possibility of symbol miss-identification during the digitisation or the incorrect association of some properties to certain components becomes high (Arroyo et al. (2016), Elyan et al. (2018)). Thus, it is expected that this process is not perfect and therefore, most systems enable human interaction to validate the symbol identification, connection and property association. Therefore, the final *Automatic CAD* document is always verified by an engineer due to the need of being a zero-error process.

5.2.1 Generating a CAD model given a P&ID

The previously explained migration of paper P&IDs implies two main steps: digitisation and contextualisation.

Digitisation

In this first step, the main shapes of the P&ID are detected and localised. Basically, there are three types of figures in these drawings: symbols representing equipment and instrumentation, lines providing the connectivity between equipment and text

[1] `https://cfmgcomputing.blogspot.com/2018/06/rgu-and-dnv-gl-join-forces-to-create.html`

[2] `https://cfmgcomputing.blogspot.com/2018/06/rgu-and-dnv-gl-join-forces-to-create.html`

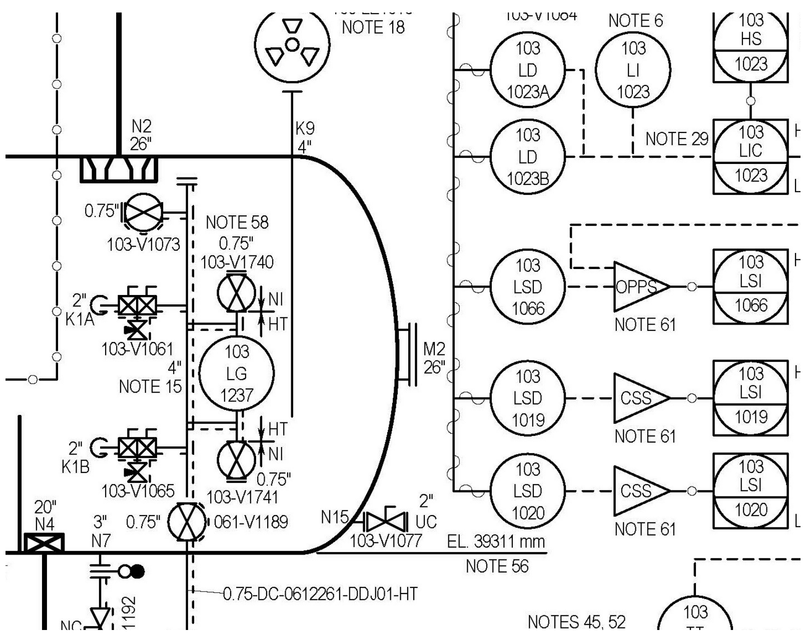


Figure 5.1. An example of a piece of a P&ID.

describing the characteristics of equipment and connectors. Figure 5.1 shows a piece of a P&ID that contains these elements.

As mentioned in Section 5.1, numerous systems have been proposed in the literature for the digitisation of P&IDs. Most of these systems rely in heuristic-based computer vision methods to recognise the shapes (Howie et al. (1995), Tan et al. (2016), Arroyo et al. (2016), Moreno-García et al. (2017), Kang et al. (2019)), although most recent literature has preferred the use of deep learning based technologies to localise and extract the figures from the engineering drawing (Gellaboina and Venkoparao (2009), Rahul et al. (2019)). Regardless of the approach, the outcome of a digitisation system is usually a list of components along with some characteristics (such as tag id, thickness or location within the drawing or the facility) and a list of pipelines (i.e. connectors between components). Notice that in the case of P&IDs, it may be the case that pipelines also have some associated properties, such as material, thickness, composition, etc. These properties are usually indicated by text which lies adjacent to the line or that is connected to the pipeline through a lead line.

Although most of the times digitisation is done as a standalone process where shapes are recognised either by heuristic or deep learning based methodologies, some previous work has been benefited from understanding *a priori* the standards used to produce these drawings to obtain all shapes with improved accuracy. For instance, Moreno-García et al. (2017) presented a comparative study of text/graphics separation algorithms (Tombre et al. (2002)) applied on P&IDs, where parametrical symbols (i.e. the ones described with circles and lines) were found first by means of state-of-the-art methods such as Hough circles and transform (Ballard (1981)). Based on the previous knowledge, the text within the shapes is located and characterised easily. Thus, some text characters can be detected along with their properties (i.e. width, height, pixel density, stroke, amongst others). This allows an easier detection of the remaining text throughout the drawing.

Once the P&ID has been digitised, a *netlist* which contains the position of each element within the drawing is produced. Nonetheless, a *netlist* is not directly usable for an expert, as it only contains the positions of the found shapes, and it does not reflect the topology of the schema. Some online tools have been presented to do quick visualisations of the extracted *netlists*, such as *NetVis* or *Netlist2CAD*[3], however none of these tools offer a true connectivity comprehension of the extracted information. Still, these come handy for a quick human inspection of the extracted data and to manipulate/correct some shapes.

Contextualization

In this second step, the shapes in the netlist are contextualised. Contextualisation means making sense out of the digitised data. Thus, the CAD model topology describes the connectivity and the relation between the components. The output can be a standardised file (Howie et al. (1995), Arroyo et al. (2016)) or a graph (Rantala et al. (2019)). These files are naturally imported by CAD applications. For a more in-depth analysis of the challenges of digitising and contextualising these drawings, the reader is referred to the reviews presented in Moreno-García et al. (2019), Moreno-García and Elyan (2019).

Regarding actual contextualisation, the use of graph-based or tree-based structures to represent and store P&IDs has become widespread in recent years (Howie et al. (1998); Rahul et al. (2019); Rantala et al. (2019); Kang et al. (2019)), however no attempts have been done at analysing the topology of the diagram or exploiting the

[3]`http://cfmgcomputing.blogspot.com/p/software-demos.html`

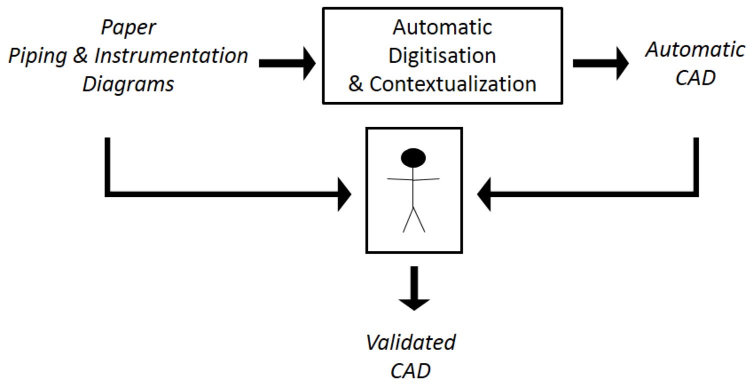


Figure 5.2. The process of deducing the CAD document, in which a human is involved to validate the data.

topological information contained on the key drawings (i.e. the sheet containing the legend of each symbol and the most common configurations, an example can be found in[4]) to improve or automatically correct the digitisation of P&IDs.

The methods presented until now perform the digitisation and contextualisation and the final result is always validated by a human expert. Figure 5.2 shows a general flow diagram of this classical approach. Thus, the topology of the P&ID is deduced from the digitisation but the identity of the components is never reviewed given the deduced topology of the P&ID. The final result obtained after the Automatic Digitisation and Contextualization will be called *Automatic CAD* during all this chapter. The method that will be explained in the next section reviews the identity of the components in the *Automatic CAD* by comparing the deduced topology to the most probable one. Then, it presents to the human the components that could have been incorrectly identified.

5.3 The proposed method

The method is represented in Figure 5.3. The difference between this model and classical models (Figure 5.2) is the incorporation of the Automatic Validation method that is marked in red in the figure. The aim of this method is to deduce the iden-

[4]https://www.edrawsoft.com/pid-legend.php

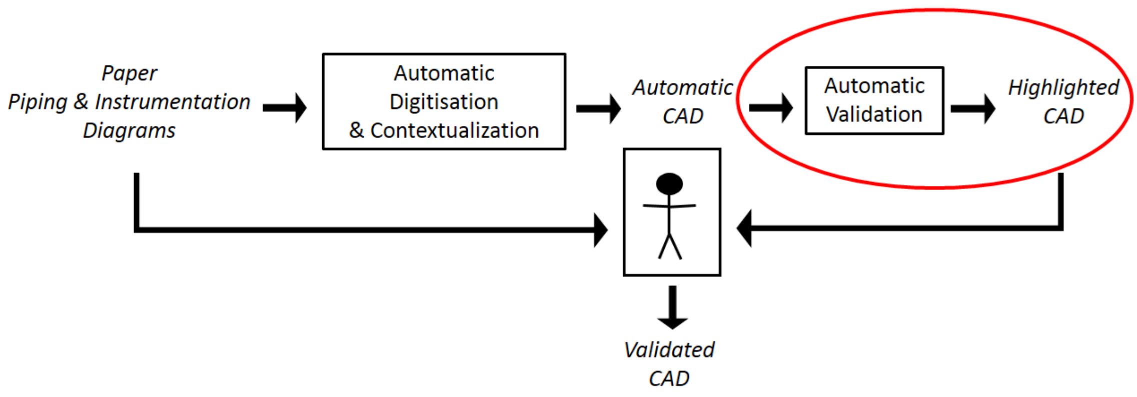


Figure 5.3. Our model for automatic detection of possible incorrectly identified components and final human validation.

tity of the components in the *Automatic CAD* and highlight the ones that must be reviewed by an human expert. In this way, the number of components that need to be reviewed is reduced.

Our *Automatic Validation* method is composed of two main modules that are represented in Figure 5.4 and detailed in the next subsections:

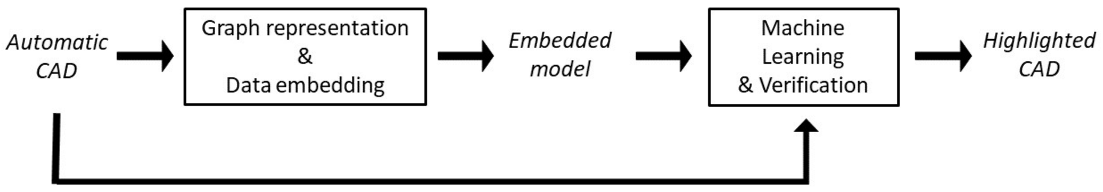


Figure 5.4. Automatic Validation module incorporated in Figure 5.3.

5.3.1 Graph representation and data embedding module

The aim of this first module is to represent a P&ID as an attributed graph (see Section 2.1) and embed it in a euclidean space.

Graph representation

To do the graph representation of the P&IDs, nodes represent components and edges represent pipelines that connect these components. Moreover, nodes have only one attribute, which is the component identity (for instance, valve or compressor) and edges are unattributed and undirected. A sample of graph representation of a P&ID is presented in the left part of Figure 5.5.

Data embedding

Graphs have some limitations when they are applied to machine learning, due to their intrinsic relational representation and because some trivial mathematical operations used in traditional numeric machine learning have not an equivalence in the graph domain. Given an arbitrary set of graphs, a possible way to address this problem is to define an embedding function from the graph domain to a vector space (Gibert et al. (2012)). Broadly speaking, an embedding function converts an attributed graph into a vector. However, defining such embedding function is extremely challenging, when the constraints on time efficiency and preserving the underlying structural information is concerned.

Explicit graph embedding is based on defining a function that, given a graph, generates points in an euclidean space. These embedding functions can be divided into four classes:

- Graph probing (Luqman et al. (2013)) that measures the frequency of specific substructures.
- Spectral graph theory (Caelli and Kosinov (2004)), which analyses the structural properties of graphs in terms of eigenvectors and eigenvalues.
- Dissimilarity measurements (Duin and Pekalska (2011)), in which the function depends on its distance to a selected set of graphs.
- Geometric deep learning (Defferrard et al. (2016)), in which the embedding uses deep neural networks.

The embedding used in this work is a probing embedding that, given a node, has a computational cost linear in respect to the number of outgoing edges of the node. It considers a node and the set of nodes and edges connected with it. It is defined as follows:

$$\Phi : G \rightarrow R^{n+2}$$

$$v_i \mapsto \Phi(v_i) = (c_i, d_i, f_i^1, ..., f_i^n)$$

where c_i is the identity of node v_i, d_i is the degree of v_i, f_i^p is the number of neighbours of v_i with identity p, with $p = 1, ..., n$, and n is the number of different identities. A sample of a node embedding is presented in the right part of Figure 5.5.

After representing the P&ID as a graph and embed all the nodes, the output of this module is the set of the embeddings of all the nodes representing the components of the P&ID.

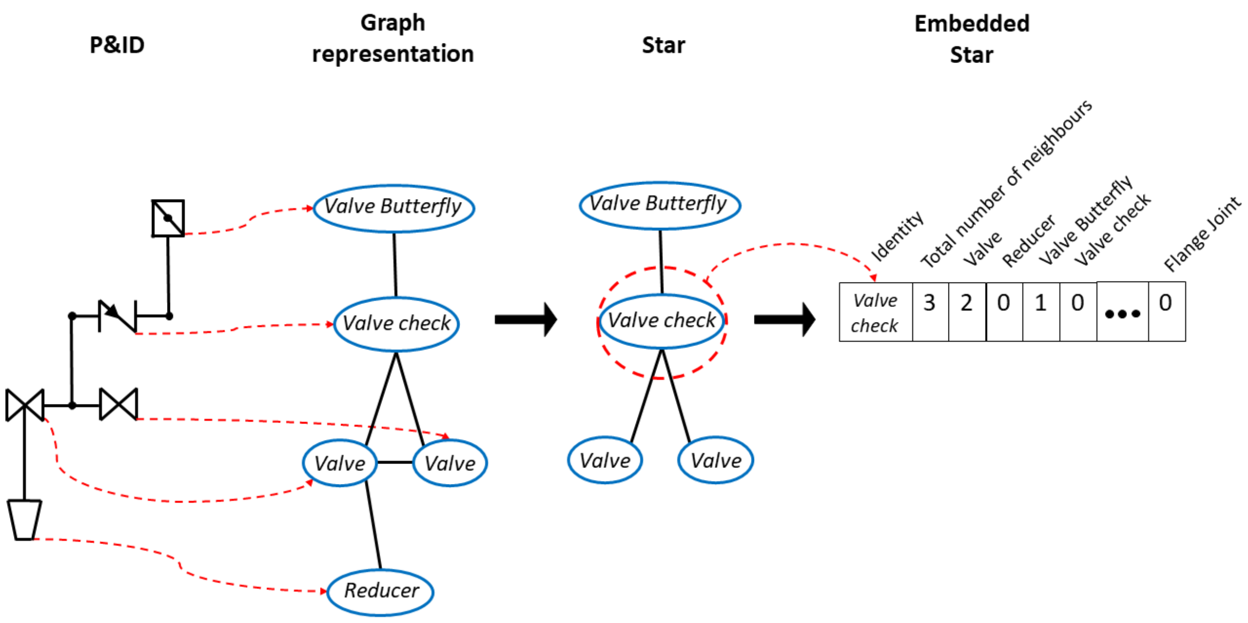


Figure 5.5. An example of embedding a *Valve check* star into a vector.

5.3.2 Machine learning and verification

The *Machine Machine Learning and Verification* module performs the following tasks:

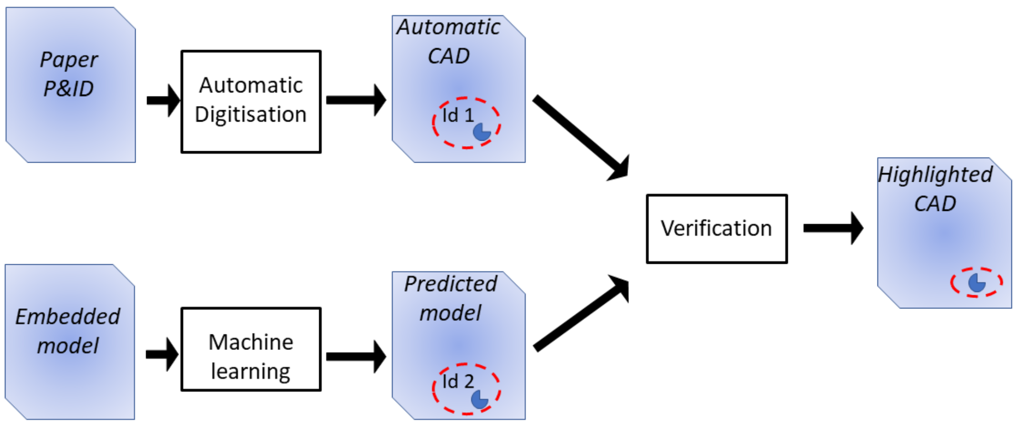


Figure 5.6. Creation of the Highlighted CAD.

Machine learning

The set of vectors representing the embedded components in the P&ID are introduced into a feed-forward neural network (NN) that returns the predicted identity of each component. This NN is composed of three layers (input, hidden and output layer). It has a hidden sigmoid neuron and softmax output neuron. The training of the NN is done with scaled conjugate gradient back propagation algorithm.

Verification

The identities returned by the NN are contrasted with the identities obtained from the digitised and contextualized *Automatic CAD* that have not been verified by the engineer. Thirdly, the method detects the components of the *Automatic CAD* whose identities are different to the identities obtained by the neural network. Then, they are highlighted in a *Highlighted CAD* that has to be validated by the human expert. Figure 5.6 represents this step.

5.3.3 Theoretical analysis of the validation method

The simplest metric to evaluate the quality of an identification method is by the percentage of correctly identified elements (true predictions) and the percentage of incorrectly identified elements (false predictions). Nevertheless, the aim of this method is not to evaluate the quality of the predictions given by the neural network, but to detect the errors in the *Automatic CAD* deduced by the Digitisation and Contextualisation process. In this method there are not only the ground-truth identities and the predicted identities, but also the identities in the *Automatic CAD*. This means that this method deals with three different sets of data.

In this context, the identities predicted by the NN are split in two main scenarios presented in Table 5.1. Per each component, it is considered whether the identity predicted by the NN (called ”Prediction” in Table 5.1) is the same as or different from the identity obtained by the *Automatic CAD* (columns in Table 5.1). This is because, if they are the same, the human is not asked to validate the identity of the component but, if they are not the same, the human is asked to validate it. In this second case, the component is highlighted in the *Highlighted CAD* (Figure 5.6).

To find the errors that could appear, we compare the predictions of the NN with the ground truth set, that is, the set of components that have been previously checked by the engineer. We call this set the *Validated CAD*. We split the previously explained categories (columns in Table 5.1) again depending on whether the prediction given by the NN is equal to the ground truth (True Prediction) or it is different from the ground truth (False Prediction) (rows in Table 5.1). With this division, the next four situations are possible:

- Case A: Both identities are the same and they are correct. The human is not asked to validate the component and the identity in the *Automatic CAD* is correct. Therefore, there is no error in this situation and the component does not need to be corrected.
- Case B: Both identities are the same and they are incorrect. Although both identities are incorrect, the human is not asked to validate the component because both are the same. In this case, we face a not detected error.
- Case C: Both identities are different. The predicted by the NN is correct but the identity in the *Automatic CAD* is not. The human is asked to validate the identity of the component, there is need of validation. We cannot influence on the number of elements in C since it depends on the quality of the *Automatic CAD*.

- Case *D*: Both identities are different. The prediction of the NN is incorrect and the identity in the *Automatic CAD* is correct. The human is asked to validate it because both are different. Validation is required to check the correct identity in the *Automatic CAD*.

	Prediction=Automatic CAD	**Prediction≠Automatic CAD**
True Prediction	A	C
False Prediction	B	D
	↓	↓
	Human does not validate	Human does validate

Table 5.1. Possible cases of the Automatic Validation method.

After considering the previous cases, we propose two metrics to validate the Automatic Validation method instead of the classical *True Prediction* or *False Prediction*. These are:

- **Human effort**: The percentage of components that the human has to validate since the identities in the *Automatic CAD* and the predicted by the NN. It is given by the sum of cells *C* and *D* in Table 5.1. Clearly, we wish this metric to be the lowest as possible to reduce the economical and temporal impact of checking the incorrectly identified components in the *Automatic CAD*, although we cannot influence on *C* as commented above.

- **Validation error**: The percentage of components that the human does not validate and would have to be validated. This value is shown in cell *B*. If we want the method to be error-free, this value has to be strictly zero.

5.4 Experiments

In this section the dataset, experimental setup, results and discussions are presented.

5.4.1 Dataset

The experiments have been carried out with industrial data. A batch of P&IDs was obtained from an existing collaboration with an industrial partner by means of a project funded by Scottish Innovation Centres. Nevertheless, all data have been anonymised to fulfil the company requirements. The experimental validation has been carried out as follows. First, four P&ID sheets from the same standard have been used. From each P&ID, the *Validated CAD* sheet has been used for training, and the *Automatic CAD* sheet for test.

Thus, the learning set is composed of four *Validated CAD* sheets (ground truth set) and the test set is composed of the four *Automatic CAD* sheets. Table 5.2 shows the number of components and different identities in each P&ID sheet in the learning set.

	Sheet 1	**Sheet 2**	**Sheet 3**	**Sheet 4**	**Total**
Number of components	125	107	177	115	524
Number of identities	22	10	24	18	38

Table 5.2. Number of components and identities in the four sheets that compose the learning set.

As we can see, in the learning set there are 524 components, which have 38 different identities. Note that the total number of identities is not the sum of the number of identities in all the sheets because some identities appear in more than one sheet. Table 5.3 shows the most common identities with their total frequencies in the learning set. We observe that there are 108 *Valve Ball* components, which is one fifth of the learning set. In Table 5.3 there are only 7 classes presented because the other ones have less than 19 appearances and we did not present them. In total, there are 38 different identities, it means that there are other 31 identities very poorly represented in the learning and test set. This means that we have a highly unbalanced learning set that hinders the learning process.

Identity	**Frequency**
Valve Ball	108
Reducer	77
Continuity Label	68
Flange Joint	41
Arrowhead	35
DB&BPV	23
DB&BBV	19

Table 5.3. Frequencies of the most common identities in the learning set. Junctions are not considered as nodes.

5.4.2 Experimental setup

On the previously described sets, the "Graph Representation & Data Embedding" module (Figure 5.5 and Section 5.3.1) returns the embedding of each component. As the total number of identities is 38 (see Table 5.2), the vectors resulting from the embedding are of dimension 40. The first position in each vector is the identity of the component and the second position is the number of pipelines connected to the component. Each one of the other 38 positions, i, is the number of neighbours with identity i.

The neural network defined in the "Machine Learning and verification" module (see Figure 5.4 and Section 5.3.2) has 40, 39 and 38 neurons in the input, hidden and output layers respectively. The neural network architecture and training process has been carried out through the *"Neural Network Matlab tool"*.

5.4.3 Experimental results

Table 5.4 shows the true and false predictions per each sheet in the test set. A detailed theoretical explanation of the results was presented in Section 5.3.3.

		Pred.=Automatic CAD	**Pred.≠Automatic CAD**
Sheet 1	True Pred.	42	2
	False Pred.	0	81
Sheet 2	True Pred.	66	1
	False Pred.	0	40
Sheet 3	True Pred.	56	3
	False Pred.	0	118
Sheet 4	True Pred.	35	0
	False Pred.	0	80
Total	True Pred.	199	6
	False Pred.	0	319

Table 5.4. Evaluation of the proposed method. Pred. represents the identity predicted by the neural network.

Considering the total values of the four sheets (two last rows in Table 5.4, we observe that there are 199 components that do not need to be validated by the human expert because both predictions are equal and both are correct. There are six cases where both predictions do not match, our method is correct but the *Automatic CAD* does not. On the other hand, there are 319 cases in which the two predictions are different and the NN predicts an incorrect identity. In these cases, some of them have been correctly predicted by the *Automatic CAD*, but others do not. The human expert has to check the identity of $6 + 319 = 325$ components instead of the total 524. It means that the expert just has to check 62.02% of the components. We observe that there are not cases in which both methods deduced the same incorrect identity and, consequently, all errors are detected and we achieve the aim of zero-error process.

Table 5.5 shows the two metrics presented in Section 5.3: *Human effort* and *Validation error*. It turns out that, thanks to our method, in the worst of the cases (Sheet 4), the human expert needs to check 69.57% of the components and in the best case (Sheet 2), just 38.32%. The average human effort is 60.66% of the effort without applying our method, this means that there is a human effort reduction of 39.34%, which is the main aim of our method. Also, it is important to note that in all the sheets, the zero-error process is achieved because all the incorrectly identified components have been detected (Validation error = 0).

Sheet	**Human effort**	**Validation error**
1	66.40 %	0 %
2	38.32 %	0 %
3	68.36 %	0 %
4	69.57 %	0 %
Average	60.66 %	0 %

Table 5.5. Human effort and Validation error of the four sheets and their average.

We realise that there is a human effort reduction keeping an error-free process and therefore, our aim has been achieved. Nevertheless, there is a gap to improve these results. We believe a higher reduction of the human effort could be achieved by increasing the size of the learning set.

5.5 Conclusions

The automatic migration of paper engineering drawings into digital environment is a tendency and the development of new techniques to validate the digitisation process is required.

The method explained in this chapter is designed to reduce the human effort while validating the digitisation of pipping and instrumentation diagrams.

The method is based on the graph representation and embedding of the diagrams. The diagrams have been represented with attributed graphs and their components have been embedded into vectors that reflect the connections between the components. For each vector, a neural network has been used to predict the identity of the component represented by this vector. Only the components where the predicted identity is different from the identity that had been previously automatically deduced, are presented to the human expert to be validated.

The experimental validation shows an average reduction of approximately the 40% of the human effort, while keeping an error-free process.

6
P&IDs: Subgraph querying

6.1 Introduction

As commented in Chapter 5, there is a tendency to migrate printed P&IDs towards a digital environment. In this context, new tools that facilitate the use of digital engineering drawings are needed. Some have been presented to visualise the digital information of the drawings, but the development of new techniques is required to extract more information from the sheets of components.

In this chapter we continue with this topic adding the new functionality of finding in a P&ID a group of components that has a specific structure or similar to it. For instance, to detect the appearances of structures that include a *valve check* connected to two *general valves* and a *butterfly valve.* In the field of graphs, this problem is related with Top-K-sub-graph search, that consists on finding the K appearances of a query graph into a commonly larger graph. Algorithms solving this problem have been widely used in some applications such as internet searches and social data analysis.

This chapter is organised as follows: In Section 6.2, some algorithms related with this topic are exposed. In the Section 6.3, the algorithm is detailed. In Section 6.4, different experiments analyse some characteristics and the goodness of the method and Section 6.5 is dedicated to conclusions.

6.2 Background

As commented in Chapter 5, if we represent P&IDs with graphs, nodes represent the components of the diagram and the attributes in nodes are one value representing the identity of the component (Rantala et al. (2019); Rica et al. (2020)). In this situation, the problem of searching a component in a P&ID is linear respect to the number of components in the P&ID. Nonetheless, moving from detecting the appearance of one component to K similar groups of connected components to a query one, makes the problem to be much more complex. In this case, the equivalent in the field of graphs is to detect the appearance of K similar sub-graphs in a graph, called *Top-k sub-graph search* problem. In this case, we are facing a NP-problem (it is not solvable in polynomial computational cost in respect to the number of nodes).

In this context, this problem has been addressed in two main different directions. The aim of the first direction is to find K exact appearances of the query graph seeking K exact sub-graph isomorphisms (Yang et al. (2016); Zou et al. (2007)). The aim of the second one is to find K sub-graphs in the larger graph that are similar in some sense to the query graph. This similarity is commonly measured through the definition of a distance that is adapted to each application. In this case, the algorithms return what are usually called non-exact sub-graph isomorphisms (Habi et al. (2019); Fan et al. (2013); Khan et al. (2011); Gou and Chirkova (2008)).

The problem of searching groups of connected components similar to a given one in a P&ID, fits with the second family of methods because we do not need only the exact substructure. But methods presented in Habi et al. (2019); Fan et al. (2013); Khan et al. (2011) or Gou and Chirkova (2008) are based on the assumption that connected nodes tend to be similar. If we think about the graph representation of a P&ID, the attribute in each node is just the numerical representation of the identity of the component without any real relation between the attribute and the component (for instance, "valve" is represented with a "1", and "reducer" is represented with a "2"). Indeed, we can not guarantee that connected components are similar and, thus, we can not apply these methods to the problem of searching similar substructures in P&IDs. Although we can not guarantee that connected components are similar, we can define some type of measure of similarity between components in a P&ID

independently on if they are connected or not. To do it, we suggest to make use of the concept of Graph Edit Distance (GED). Using the GED, the expert engineer is allowed to set the approximated cost of changing a component for another one in a P&ID. For instance, the engineer can set the cost of changing one *valve* for a *reducer* equal to some value depending on the application. If this change represent a big modification in the diagram, the cost should be tuned as high, whereas if the change does not affect a lot to the diagram, the cost should be set as low.

6.3 The proposed method

The presented method is called *Top-K-GED* sub-graph search algorithm and it returns K locations in the P&ID where similar sets of components appear. The value of K is previously set by the user and it defines the number of substructures that the engineer needs to find. The main idea of the method is to compute the edit cost of transforming the queried substructure into other ones and to give as output the K substructures that have obtained lower cost. Moreover, the method returns them ordered by the similarity between the queried substructure and the detected ones.

This section has three subsections. In Subsection 6.3.1, general considerations about the algorithm are presented. In Subsection 6.3.2, the input and output parameters of the algorithm are defined. And finally, in Subsection 6.3.3, the algorithm is detailed.

6.3.1 General considerations of the method

Before explaining the method, it is necessary to set the following premises that will condition the performance of the algorithm:

- Only engineers know how similar two components in the P&ID are. This knowledge is introduced into the system through the cost of substituting components imposed by the engineers before doing the query.
- In a similar way, only engineers know how important is a component or a

pipeline in the P&ID. This knowledge is considered in the cost of deleting and inserting a component or a pipeline.

As commented before, the algorithm defines P&IDs as attributed graphs. In these graphs, nodes represent components and edges represent pipelines that connect these components. Moreover, nodes have only one discrete attribute, which is the identity of the component (valve, compressor,...), and edges are unattributed and undirected since we do not have information about the type of pipelines or their features. Moreover, the method defines the local substructure to be queried as other smaller graph.

The costs that the engineer sets for substitution, deletion and insertion of components and pipelines (nodes and edges), correspond to the costs presented in Equation 2.1 and Equation 2.2 respectively and the presented method is based on minimising the suboptimal value of Equation 2.5. To minimise this equation, it is needed to use some optimisation algorithm. In this work, we have applied *Belief* graph matching algorithm (Santacruz and Serratosa (2018a)). This algorithm has been chosen because it has two main properties not found in other algorithms:

- The computational cost of *Belief* algorithm is linear in respect to the number of nodes.
- The substituted nodes and edges in the resulting query, form a compact graph, that is one of the main objectives of the method.

6.3.2 Input and output parameters of the algorithm

The **input** of the method is composed of:

- Q: A small graph that represents the queried substructure. Q has n nodes (components).
- G: A graph larger than the queried graph Q representing the big P&ID. G has m nodes ($m > n$).

- K: The number of compact substructures that the method has to return.
- $S_v \in M_{N\times N}(\mathbb{R}_0^+)$: A matrix of nodes substitution costs (N is the total number of different existing attributes (identities) in nodes in G and Q). Each cell of the matrix S_v is a non-negative real number that represents the cost of substituting a component with identity i by a component with identity j. All the elements in the diagonal are zeros.
- $D_v \in (\mathbb{R}_0^+)^n$: A vector of node deletion costs. Each cell $D_v(i)$ is a non-negative real number that represents the cost of deleting a component with identity i.
- $D_e \in \mathbb{R}_0^+$: Edge deletion cost. Since edges do not have attributes, the cost of deleting an edge is the same for all edges. It is a constant, previously set.

Note that the edge substitution cost is not an input parameter since edges (pipelines) do not have attributes. Furthermore, edge and node insertion costs are not input parameters either since they equal zero to assure that the number of components in the big graph does not influence on the final result.

In the application of the method on the P&IDs or in any other applications, the values of the introduced costs S_v, D_v, D_e have to be imposed by an expert engineer in the field where we want to apply the method. As it will be explained in Subsection 6.4.3, the introduced values affect directly on the obtained results.

The **output** of the method is composed of:

- $\{f_1, ..., f_K,\ s.t.\ f_i : Q \to G, i = 1, ..., K\}$. A list of K node-to-node mappings between the query graph Q and the big graph G.
- $\{Cost(Q, G, f_1)$,..., $Cost(Q, G, f_K)\}$. A list of K edit costs (Equation 2.5), given the query graph Q, the big graph G and the above mappings f_p, $1 \leq p \leq K$. If we define G_p as the nodes in G reached by substitutions in f_p, then $Cost(Q, G, f_p)$ is the distance between Q and G_p, $Cost(Q, G, f_p) = GED(Q, G_p)$ where GED is defined in Equation 2.4.

The mappings f_1,...,f_K returned by the algorithm hold the following four conditions:

- For each $1 \leq p \leq K$, the set of nodes $\{f_p(v)|v \in Q\}$ and the corresponding edges, define a compact sub-graph.
- *If* $f_p(v) = f_q(v), \forall v \in Q \Rightarrow p = q$. This means that two mappings cannot be identical.
- The mappings f_1,...,f_K are listed in ascending order on their edit costs: $Cost(Q, G, f_1) \leq Cost(Q, G, f_2) \leq ,..., \leq Cost(Q, G, f_K)$.
- *If* $f \notin \{f_1, ..., f_K\} \Rightarrow Cost(Q, G, f) \geqslant Cost(Q, G, f_p)$, $\forall p = 1, ..., K$. This means that the mappings f_1,...,f_K might have to be the ones that return the minimum costs. Note that this imposition cannot always be guaranteed due to our algorithm does not always return the optimal solution.

6.3.3 *Top-K-GED* sub-graph search algorithm

In this section, *Top-K-GED* sub-graph search algorithm is explained and summarized in Algorithm 3. It has three main steps:

In the first one, a cost matrix C is computed with dimensions $m \times n$, where m and n are, respectively, the number of nodes of the query graph Q and the big graph G with $m \leq n$. Each cell in C, $C(a, i)$, $1 \leq a \leq m$, $1 \leq i \leq n$, represents the cost of substituting the star S_a in Q by the star S_i in G as follows (see Chapter 2, Section 2.2):

$$C(a, i) = C_S^{Star}(a, i) \quad 1 \leq a \leq m \quad and \quad 1 \leq i \leq n \tag{6.1}$$

where $C_S^{Star}(a, i)$ denotes the cost of substituting the $Star$ centred at node v_a by the $Star$ centred at node v_i' as presented in Equation 2.5.

In the second step, the K cells in the cost matrix that have the minimum value are selected. These substitution costs are used to set the K different *Seeds* that *Belief* algorithm is going to use in the K times it is run in the next step of our algorithm.

If we define C_p as the p lowest value in C, then:

$$Seed_p = (a, i), \quad being \quad C_p = C(a, i) \tag{6.2}$$

Note that several cells can be selected from the same column or from the same row. Thus, it is accepted to have $Seed_p = (a, i)$ and $Seed_q = (b, j)$, $p \neq q$, being $a = b$ or $i = j$ but not both.

Finally, in the third step, a modified version of *Belief* algorithm is executed K times. Each time, a different seed is used: $Seed_1$, ..., $Seed_K$. Using a different seed makes the algorithm to return a different mapping between the query Q and the big graph G. This property could be considered a drawback in other methods but it becomes a must in this case. Matlab implementation in [1].

Algorithm 3 *Top-K-GED*

```
Input: $Q$, $G$, $S_v$, $D_v$, $D_e$, $K$
Output: $f_1$, ... , $f_K$, $Cost_1$, ... , $Cost_K$, being $Cost_p = Cost(Q, G, f_p)$

1.      $C = ComputeCostMatrix(Q, G, S_v, D_v, D_e)$
2.      $(Seed_1, ..., Seed_K) = SelectLowerCostCells(C, K)$
        For $p = 1..K$
3.          $(f_p, Cost_p) = Belief(C, Seed_p)$
        End For
    End Algorithm
```

6.4 Experiments

The experimental validation is divided in five subsections. In the first one (6.4.1), the metric to evaluate the method is defined. In the second one (6.4.2) the data set is introduced. In the third (6.4.3) and fourth (6.4.4) ones, the analysis of the

[1]http://deim.urv.cat/~francesc.serratosa/SW/

influence of the graph matching algorithm and the edit costs on the query results are presented. In the fifth one (6.4.5), two examples of query results are shown and finally, in the last one (6.4.6), the evaluation of the method is presented.

6.4.1 Metric to evaluate the method

We use *Recall* metric to analyse our method, as usual in retrieval applications. In our case, it is:

$$Recall = \frac{\textit{Number of relevant retrieved sub-graphs}}{\textit{Number of relevant sub-graphs in the P\&ID}} \tag{6.3}$$

The relevant retrieved sub-graphs (the numerator in Equation 6.3) are the ones found by the method that have an edit cost lower than a threshold imposed by the user, $Cost_{max}$. This threshold is a parameter independent of the method and it is defined to decide which returned sub-graphs are considered as acceptable or not. The value of this parameter determines how similar the retrieved structures are to the original query. The lower the value of $Cost_{max}$ is, the more similar the found substructures are to the queried graph. More specifically,

$$Recall(Q, G, K, Cost_{max}) = \frac{|\{f_p : Q \to G, p = 1...K\; s.t.\, Cost(Q, G, f_p) \leqslant Cost_{max}\}|}{|\{f : Q \to G\; s.t.\, Cost(Q, G, f) \leqslant Cost_{max}\}|} \tag{6.4}$$

Note that in our case, the value of *Recall* depends on K and on $Cost_{max}$. More specifically, the numerator depends on K and $Cost_{max}$, but the denominator only depends on $Cost_{max}$. This is because the method returns the number of substructures given by the value of the parameter K (number of retrieved sub-structures). Observing the expression, *Recall* is a non-decreasing function with respect K given a specific $Cost_{max}$. If K increases, the number of relevant retrieved mappings should increase. As a consequence, for each query Q and a fixed $Cost_{max}$, when K increases, *Recall* should increase up to its maximum value, 1.

6.4.2 Dataset

All the experiments have been carried out using the ground-truth data used in Chapter 5, available in [2]. In these experiments junctions have been considered as normal nodes and the most common types of components have been summarised in Table 6.1 (note that in Chapter 5, junctions were not considered as nodes).

Label	Junction	Valve Ball	Reducer	Continuity Label	Flange Joint	Arrowhead
Number of appearances	262	108	77	68	41	35

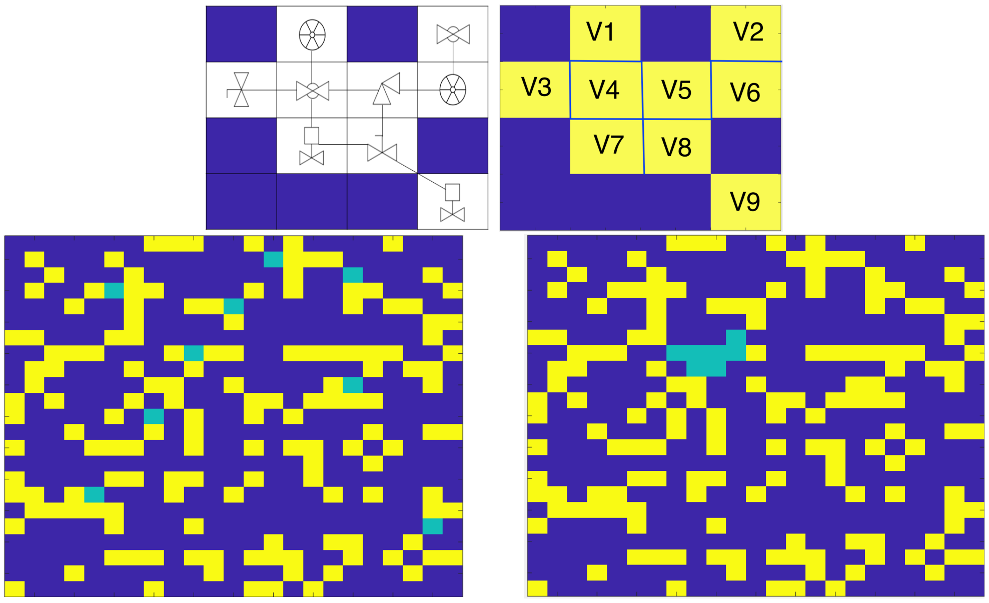
Table 6.1. Most common types of components in the four P&IDs (or different labels in the graphs).

6.4.3 Influence of the graph matching algorithm

The aim of this section is to heuristically analyse how the graph matching algorithm affects the results when searching sub-graphs. We compare *Fast Bipartite* algorithm (Serratosa (2014a)) and *Belief* algorithm (Santacruz and Serratosa (2018a)). Other algorithms that returns an optimal distance such as (Ferrer et al. (2015)) could not be executed due to time restrictions. To do this analysis, we have mapped a graph with both algorithms to see the difference between the results. Figure 6.1 shows a query graph with nine nodes and eight edges and, in the right, its representation based on spatial positions. Yellow cells represent positions where there is one component. In the lower row, we show the query results over the same P&ID obtained by the two algorithms.

First of all, we realise that the mapped sub-graph in the P&ID returned by *Belief* algorithm is a compact sub-graph, which is an important imposition of this method for the application of sub-graphs in P&IDs. This is not the case of the sub-graph returned by the *Fast Bipartite* algorithm.

[2]http://deim.urv.cat/~francesc.serratosa/databases/



(a) *Fast Bipartite* algorithm. **(b)** *Belief* algorithm.

Figure 6.1. Upper images: In the left, a query graph with nine components and in the right its representation based on its spacial positions. The yellow squares are components and the dark blue ones are positions without components. Lower row images: In cyan the query results over the same P&ID. In the left, *Fast Bipartite* algorithm (Serratosa (2014a)) has been used and in the right, *Belief* algorithm (Santacruz and Serratosa (2018a)). Pipelines are not shown.

6.4.4 Influence of the edit costs

In this section, we study how different combinations of the edit costs affect the returned sub-graphs. To do so, we have run *Top-K-Belief* algorithm three times setting different values of the edit costs given the same query in a synthetically generated P&ID. The query and the P&ID are composed of 7 and 259 components respectively. Edges are only defined between nearby nodes and there are 20 different types of components equally distributed among the nodes. Thus, the node substitution costs form a matrix $S_v \in M_{20\times 20}(\mathbb{R}_0^+)$ and the node deletion costs form a vector $D_v \in (\mathbb{R}_0^+)^{20}$.

In the three runs of the algorithm, $K = 3$ has been set and S_v has been defined as a matrix of all ones except for the diagonal cells that equal zero. This means that we consider the cost of substituting an identity by another one is exactly the same for all the components. Moreover, the 20 cells of D_v and D_e have been set as follows:

1) In the first run: $D_v(a) = D_e = 0.1$, $1 \leq a \leq 20$.

2) In the second run: $D_v(a) = D_e = 1$, $1 \leq a \leq 20$.

3) In the third run: $D_v(a) = D_e = 10$, $1 \leq a \leq 20$.

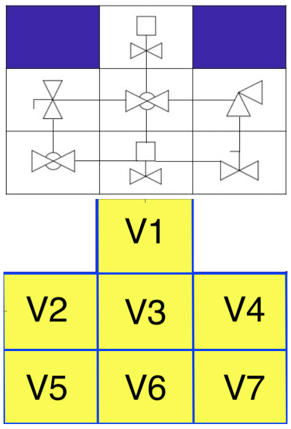
We set the cost of deleting a component to be the same than the cost of deleting a connection, $D_e = D_v(a)$, because we observe that the identities are usually connected in the same way and deleting a node or edge might have the same impact in the substructure.

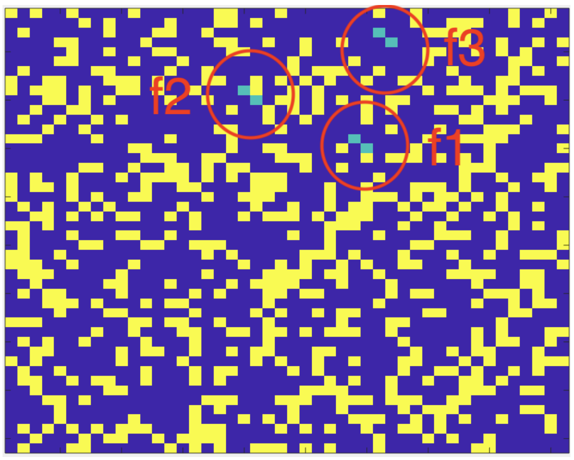
In general, the value of D_e, $D_v(a)$ and $S_v(a)$ are not important per se, but the proportions between them. Note that if all of these costs are multiplied by a real number, the GED in Equation (2.4) will be multiplied by this number because GED is linear respect to the costs. Thus, the final optimal node-to-node correspondence will be the same. Nevertheless, the increase or decrease of some of these costs will make the optimal correspondence to be different and therefore, the resulting GED and optimal node-to-node correspondence will be different.

Figure 6.2 shows in the upper-left image a query composed of seven nodes. The three next images show the results obtained in the three runs of the algorithm. Since we have set $K = 3$, we obtain three sub-graphs, f_1, f_2 and f_3, in each run. The last row in Figure 6.2 shows the detail of the nodes of the query graph that have been substituted in the P&ID.

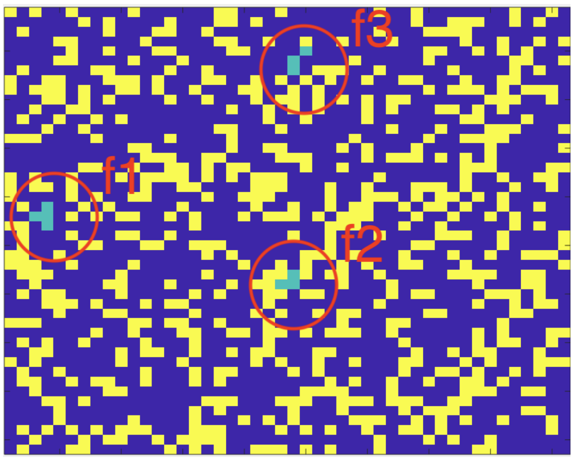
In the first run, the cost of deleting a node and an edge is very low compared to the substitution cost. For this reason, the three returned sub-graphs are composed of only two nodes since seven nodes of the query have been deleted. In the other two runs, the costs of deleting nodes and edges increase and, consequently, the number of substituted nodes increases to decrease the final cost.

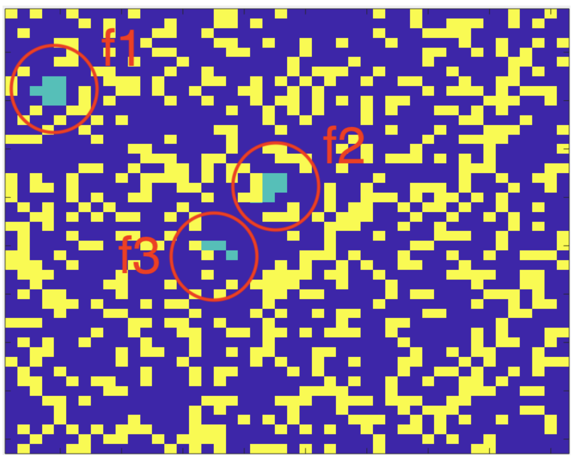
We realise that in the last run, when $D_e = D_v(a) = 10$, we obtain an exact appearance of the query graph. In this case, the cost of deleting nodes and edges is 10 times higher than the substitution cost. It means that we prioritise substitutions instead of deletion of connections and components. If the method finds exactly the same queried substructure is because all the nodes and edges have been substituted and any node is deleted. This sub-graph does not emerge in the other previous runs





(a) First run.





(b) Second run.

(c) Third run.

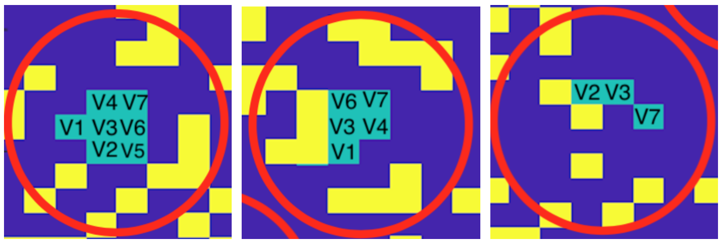


Figure 6.2. Upper left: The query graph and its representation based on its spacial positions. Upper right and second row: The P&ID components represented by yellow cells and the three returned mappings in each of the three runs of our algorithm are highlighted in cyan. Bottom row: A detail of the three mappings deduced in the last run. From left to right: f_1, f_2 and f_3.

because the cost of substituting the nodes is larger than deleting some of them. In this sense, if the user wants to obtain exactly the same substructure, the costs of substitution must be much higher than deletion costs. In the presented case, other experiments with substitution costs higher than 10, made the method to return only isomorphic graphs.

As a final conclusion, we realise that the configuration of the costs affect the final result of the algorithm. In the proposed example, the combination of values: $D_e = 10$, $D_v(a) = 10$ and $S_v(a) = 1$, was the most exact, but other proportional configurations would obtain the same results.

6.4.5 Two examples of sub-structures detection

To analyse the behaviour of the method, we show two examples of sub-graph detection to see how the method works. Figure 6.3 shows an example of two queried substructures. The substructure in the left has been queried in Sheet 2 and the results of the algorithm are shown in Figure 6.4. The substructure in the right of Figure 6.3 has been queried in Sheet 3 and the results are shown in Figure 6.5. In both examples, K has been set to 10.

All edit costs in this section and in Section 6.4.6 have been set to one. That is, for the deletion costs, $D_e = D_v(a) = 1$, $1 \leq a \leq 39$ (there are 39 different components, as shown in Table 5.2). And, for the substitution costs, $S_v(a, b) = 1$ if $a \neq b$ and $S_v(a, a) = 0$, $1 \leq a, b \leq 39$.

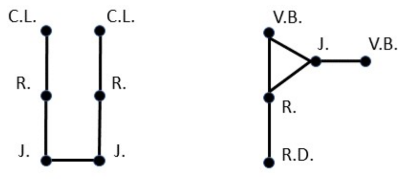


Figure 6.3. Queries searched in Sheets 2 and Sheet 3 of our database, respectively. Notation: C.L.: Continuity label, R.: Reducer, R.D.: Rupture Disc, V.B.: Valve ball, J.: Junction.

Table 6.2 shows, in each row, the returned node-to-node mappings of the first example. The first two columns are the node-to-node mapping and the cost of these

	$\boldsymbol{Cost_i}$ (**Eq. 2.5**)	$\boldsymbol{v_1 = C.L.}$	$\boldsymbol{v_2 = R.}$	$\boldsymbol{v_3 = J.}$	$\boldsymbol{v_4 = J.}$	$\boldsymbol{v_5 = R.}$	$\boldsymbol{v_6 = C.L.}$
$\boldsymbol{f_1}$	0	$v'_2 = C.L.$	$v'_{40} = R.$	$v'_{151} = J.$	$v'_{150} = J.$	$v'_{41} = R.$	$v'_3 = C.L.$
$\boldsymbol{f_2}$	0	$v'_3 = C.L.$	$v'_{41} = R.$	$v'_{150} = J.$	$v'_{149} = J.$	$v'_{42} = R.$	$v'_4 = C.L.$
$\boldsymbol{f_3}$	0	$v'_4 = C.L.$	$v'_{42} = R.$	$v'_{149} = J.$	$v'_{148} = J.$	$v'_{43} = R.$	$v'_5 = C.L.$
$\boldsymbol{f_4}$	0	$v'_5 = C.L.$	$v'_{43} = R.$	$v'_{148} = J.$	$v'_{147} = J.$	$v'_{44} = R.$	$v'_6 = C.L.$
$\boldsymbol{f_5}$	0	$v'_8 = C.L.$	$v'_{45} = R.$	$v'_{145} = J.$	$v'_{144} = J.$	$v'_{46} = R.$	$v'_9 = C.L.$
$\boldsymbol{f_6}$	0	$v'_9 = C.L.$	$v'_{46} = R.$	$v'_{144} = J.$	$v'_{143} = J.$	$v'_{47} = R.$	$v'_{10} = C.L.$
$\boldsymbol{f_7}$	3	$v'_6 = C.L.$	$v'_{44} = R.$	$v'_{147} = J.$	$v'_{146} = J.$	$v'_{98} = R. + F.$	$v'_7 = C.L.$
$\boldsymbol{f_8}$	4.8	$v'_{10} = C.L.$	$v'_{47} = R.$	$v'_{143} = J.$	Del	Del	Del
$\boldsymbol{f_9}$	4.8	$v'_{12} = C.L.$	$v'_{49} = R.$	$v'_{138} = J.$	Del	Del	Del
$\boldsymbol{f_{10}}$	5	$v'_{11} = C.L.$	$v'_{54} = R.$	$v'_{137} = J.$	$v'_{111} = J.$	$v'_{100} = F.J.$	$v'_{107} = A.B.$

Table 6.2. Returned mappings by *Top-K-GED* in Sheet 2 with K=10. Notation: A.B.: Area Break, C.L.: Continuity label, F.J.: Flange Joint, R.: Reducer, R.+F.: Reducer+Flange Joint, R.D.: Rupture Disc, V.B.: Valve ball, J.: Junction.

mappings, respectively. For the rest of the table, if the cell in i^{th} row and j^{th} column takes the value "$v'_n = M$" then it holds that: 1) $f_i(v_j) = v'_n$: the node v_j of the query Q has been substituted by the node v'_n of G. 2) $v'_n = M$: the identity of the image node v'_n is M. Contrarily, if the cell in i^{th} row and j^{th} column takes the value $f_i(v_j) = Del$, it means that the node v_j of the query Q is deleted in the node-to-node mapping f_i.

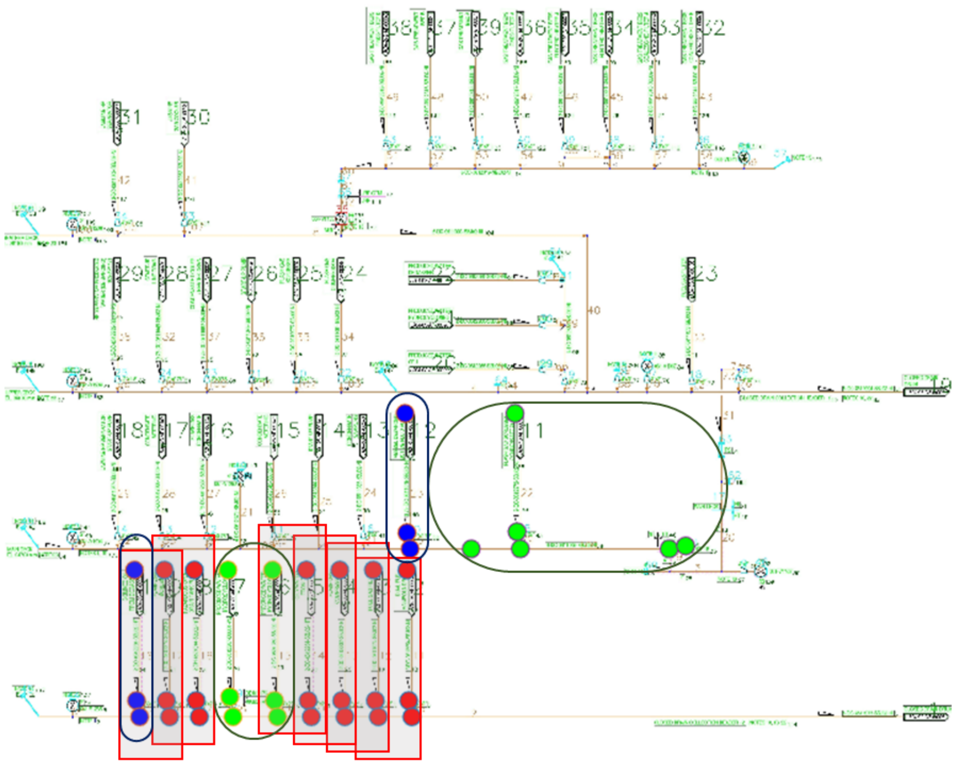


Figure 6.4. A screen shot of the first example.

The first six node-to-node mappings produce a zero cost, $Cost_i = 0$, $i = 1,..,6$. Deletion operations are set to have a non-zero cost and substitution operations are set to have a non-zero costs when the identities are different. This configuration forces the identity of each query node in Q to be the same as the identity of its image node in G and also the pipelines to be located in the same positions. This is seen in the table since the identity in each column are exactly the same. The rest of mappings produce a non-zero cost, $Cost_i > 0$, $i = 7,..,10$. Two of them are achieved by deleting three nodes in the query, f_8 and f_9, and the other ones by mapping all the nodes in the query, f_7 and f_{10}. Note the substitution $f_7(v_5) = v'_{98}$ produces a non-zero cost since $v_5 = R$ and $v'_{98} = R + F$. Similarly happens with $f_{10}(v_5) = v'_{100}$ and $f_{10}(v_6) = v'_{107}$. Finally, we realise that different mappings have query components mapped to the same component in the P&ID. For instance, $f_6(v_6) = f_8(v_1) = v'_{10}$ or $f_1(v_4) = f_2(v_3) = v'_{150}$.

Figure 6.4 shows a screen shot of the first example. The six first mappings that returned a null cost are highlighted by red points and rectangles. Some rectangles are overlapped, this is because, as we have just commented, different mappings have query components mapped to the same component in the P&ID. The four last mappings that returned a positive cost are highlighted by blue and green points and ellipses. The mappings that have deletions (f_8 and f_9) have blue points and the mappings without deletions (f_7 and f_{10}) have green points. There are some overlapping parts between ellipses and rectangles.

Figure 6.5 shows a screen shot of the second example. In this case, we do not show the specific mappings, as it is done in Table 6.2 but we realise two mappings returned cost zero and the other eight ones returned a larger cost. Six of them do not have node deletion operations and three of them have node deletion operations. As in the previous case, we observe that there are nodes that are involved in several mappings.

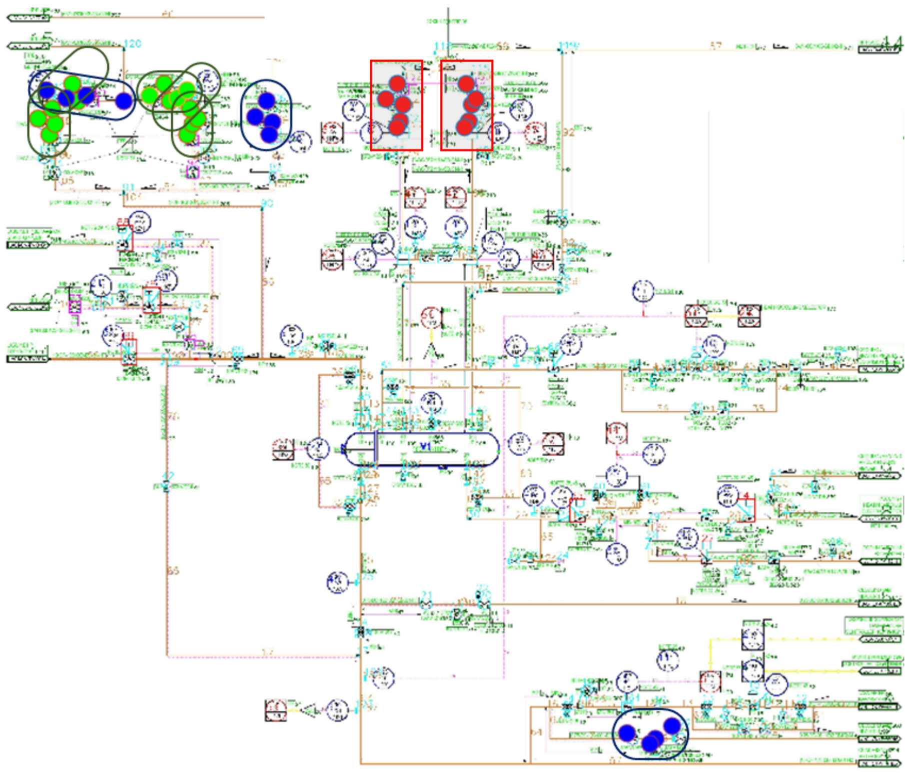


Figure 6.5. A screen shot of the second example.

6.4.6 Evaluation of the method

To evaluate the method, we have queried ten small substructures in the four sheets. Figure 6.6 shows the substructures queried in the P&IDs. We have chosen these ones to guarantee that they appear in the P&IDs.

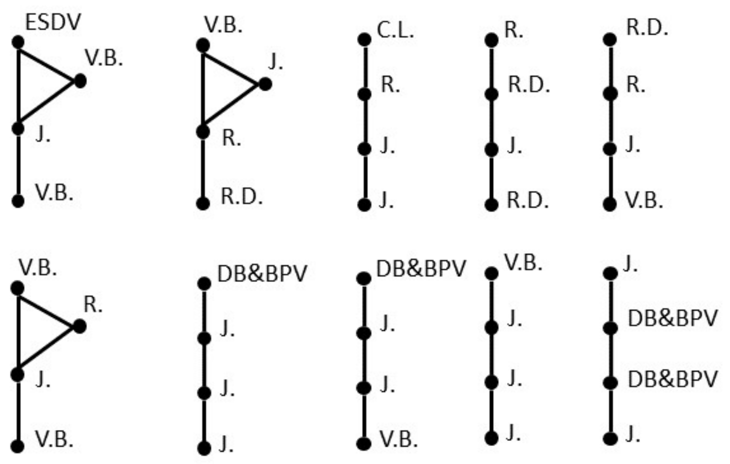


Figure 6.6. Ten queried substructures in the P&IDs. ESDV = ESDV Valve Ball Piston Operated, V.B. = Valve Ball, J = Junction, R = Reducer, R.D. = Rupture Disc, C.L. = Continuity Label, DB&BPV.

As commented in Subsection 6.4.1, the metric used to evaluate this method is the *Recall*. Figure 6.7 shows the average *Recall* values obtained in the four sheets of the database given the 10 different query sub-graphs and three different thresholds, $Cost_{max}$. These thresholds represent the ....

Given that all edit costs equal one, when $Cost_{max} = 1$ only one deletion operation (on a node or on an edge) or one node substitution with different identities is allowed. In the cases where $Cost_{max} = 2$ or $Cost_{max} = 3$ further combinations of edit costs are allowed that have a cost lower or equal to $Cost_{max}$.

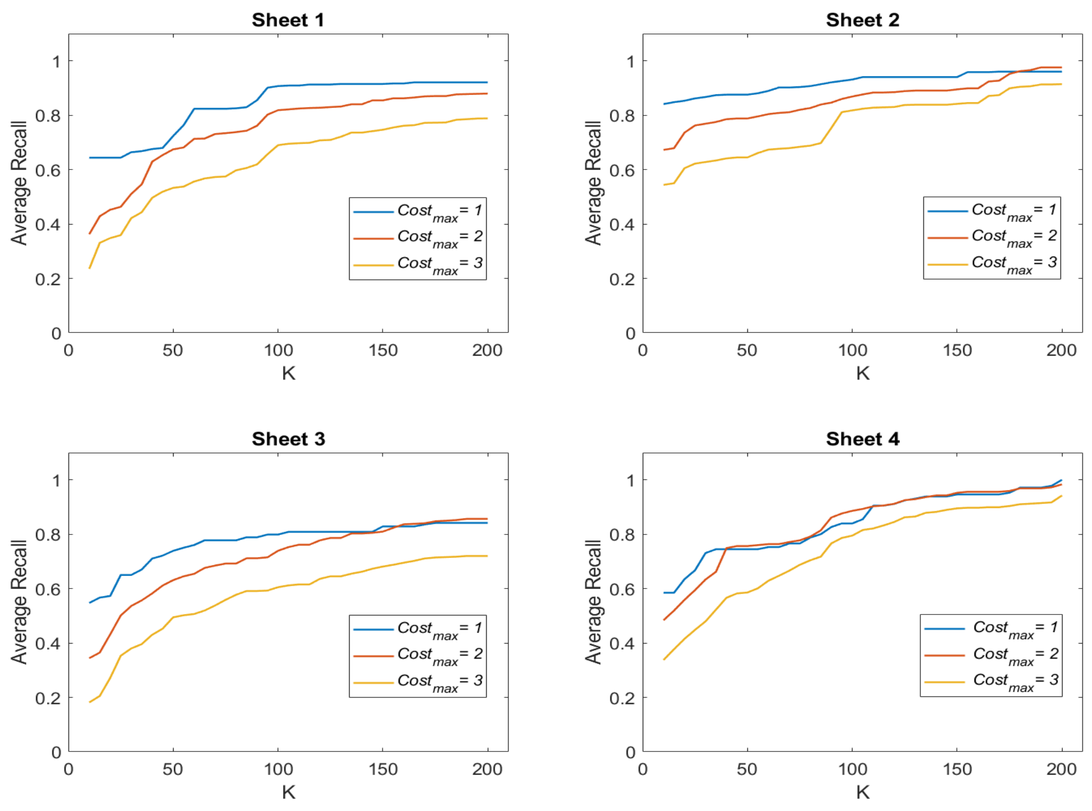


Figure 6.7. Average Recall returned by our method given 10 queries in the four sheets.

If we fix parameter K, the value of the numerator and the denominator only depends on parameter $Cost_{max}$. Moreover, both numerator and denominator are non-decreasing functions with respect $Cost_{max}$. From the experiment shown in Figure 6.7 we realise the *Recall* tends to be higher when $Cost_{max}$ is higher, given a specific K. Nevertheless, it is not always true, as we can see in the higher values of Sheet 2, Sheet 3 and Sheet 4.

Finally, note that the average *Recalls* returned in the four plots in Figure 6.7 do not achieve 1. This is because K has not been set larger enough. We do not have used values larger than 200 since it is not usual in this type of applications, where the engineer has to look up a whole P&ID.

In Figure 6.8, we show average runtimes in seconds of the algorithm (Matlab 2020a and Intel Core i7). We realise that the runtime is approximately linear in respect to K although the increment is not very significative.

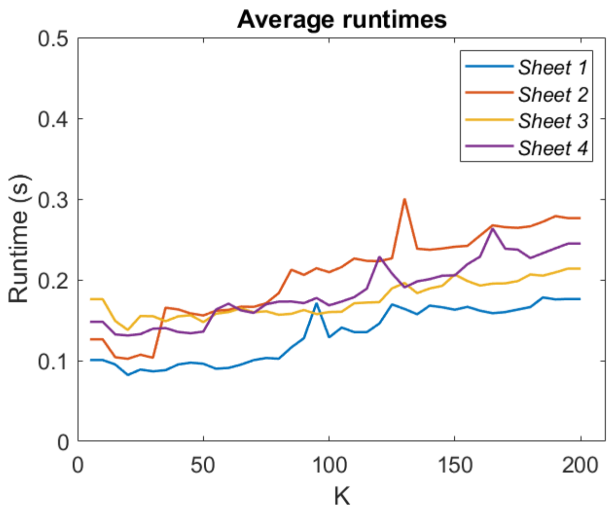


Figure 6.8. Average runtimes of our method.

6.5 Conclusions

Engineers that maintain gas and oil factories use CAD applications to inspect piping and instrumentation diagrams. These applications allow them to search for specific components but not to search for groups of connected components that are similar to some specific structure.

The method presented in this chapter looks for connected substructures similar to a queried one and it is based on solving the problem of inexact sub-graph search. Since graph matching is an NP-hard problem, heuristic algorithms have been used to achieve an acceptable runtime at the cost of not obtaining the optimal results. The method has linear computational cost respect to the number of nodes and it achieves good results in a data set of industrial diagrams.

The method is not compared to other ones because the analysed sub-graph matching algorithms return non-connected sub-graphs and they can not be applied to this problem because the principal objective of this method is to find connected sub-structures. On the other hand, the analysed Top-K sub-graph searching algorithms cannot be used since they are based on the fact that close nodes tend to have similar properties and this assumption is not fulfilled in these diagrams.

This method is a novel functionality that could be adapted and added to other CAD tools.

7 General Conclusions

The general conclusions of this thesis are as follows:

The graph representation of data allows researchers to understand the structure of data and create specific machine learning algorithms that take into account the relations between the elements in data that are represented through nodes and edges.

The Graph Edit Distance can be used as measure of similarity between graphs to solve the task of classification of nodes and graphs, and to search elements in data.

The combination of the Graph Edit Distance with vector embeddings, Support Vector Machine and Linear Discriminant Analysis achieves good results in the task of graph matching applied to diverse types of data sets.

It is possible to define on-line methods to learn the costs of the Graph Edit Distance performing a considerable reduction of the amount of data needed in the learning process.

The combination of the Graph Edit Distance with the K-Nearest Neighbours strategy achieves as well good results in the task of classification of molecules.

The graph representation of pipping and instrumentation diagrams and the application of neural networks facilitates the task of supervision in the process of digitisation of these type of industrial diagrams. The human effort can be decreased maintaining a zero error process.

The application of the Graph Edit Distance in the field of pipping and instrumentation diagrams allows to find small groups of connected substructures similar to a queried one in big diagrams with low computational cost.

As future prospects and perspectives, other concepts such as the Quadratic Assignment Problem or Graph Convolutional Neural Networks could be explored to solve tasks similar to the presented in this thesis.

8 List of publications and conferences

- Reducing human effort in engineering drawing validation.

  Elena Rica, C.F. Moreno-García, S. Álvarez, F. Serratosa.

  Computers in Industry, 117, 103198. 2020.

  `https://doi.org/10.1016/j.compind.2020.103198`

- On-line learning the graph edit distance costs.

  Elena Rica, S. Álvarez, F. Serratosa.

  Pattern Recognition Letters 146, 55-62. 2021.

  `https://doi.org/10.1016/j.patrec.2021.02.019`

- Group of components detection in engineering drawings based on graph matching.

  Elena Rica, S. Álvarez, F. Serratosa.

  Engineering Applications of Artificial Intelligence, 104, 104404. 2021.

  `https://doi.org/10.1016/j.engappai.2021.104404`

- Ligand-Based Virtual Screening based on the Graph Edit Distance.
  Elena Rica, S. Álvarez, F. Serratosa.
  International Journal of Molecular Sciences, 22 (23), 12751. 2021.
  `https://doi.org/10.3390/ijms222312751`

- Tarragona Graph Database for machine learning based on graphs.
  Elena Rica, S. Álvarez, F. Serratosa.
  Conference paper in Proceedings *S+SSPR* 2022.

- Zero-Error digitisation and contextualisation of piping and instrumentation diagrams using node classification and sub-graph search.
  Elena Rica, C.F. Moreno-García, S. Álvarez, F. Serratosa.
  Conference paper in Proceedings *S+SSPR* 2022.

- Learning distances between graph nodes and edges.
  Elena Rica, S. Álvarez, F. Serratosa.
  Conference paper in Proceedings *S+SSPR* 2022.

- On-line learning the edit costs based on an embedded model.
  Elena Rica, S. Álvarez, F. Serratosa.
  GbRPR 2019: 121-130.
  `https://doi.org/10.1007/978-3-030-20081-7_12`

- Learning the Graph Edit Costs: What do we Want to optimise?
  Elena Rica, S. Álvarez, F. Serratosa.
  GbRPR 2019: 25-34.
  `https://link.springer.com/chapter/10.1007/978-3-030-20081-7_3`

- A Human assisted model for engineering drawing validation.
  Elena Rica.
  6th URV Doctoral Workshop in Computer Science and Mathematics, 41. 2020.
  `http://digital.publicacionsurv.cat/index.php/purv/catalog/download/430/448/1011-2?inline=1#page=47`

- Clinical outcomes and safety of passive leg raising in out-of-hospital cardiac arrest: a randomized controlled trial.

Youcef Azeli, ..., **Elena Rica**, S. Álvarez, ..., M. F. Jiménez-Herrera.
Critical Care 25 (1), 1-10. 2021.
`https://ccforum.biomedcentral.com/articles/10.1186/s13054-021-03593-7`

- Effectiveness and safety of passive leg raise during cardiopulmonary resuscitation in out-of-hospital cardiac arrest.
  Y Azeli, ..., **Elena Rica**, S. Álvarez, ..., M. F. Jiménez-Herrera.
  Resuscitation 155, S38, 2020.
  `https://doi.org/10.1016/j.resuscitation.2020.08.107`

Bibliography

Algabli, S. and Serratosa, F. (2018a). Embedding the node-to-node mappings to learn the graph edit distance parameters. *Pattern Recognition Letters*, 112:353–360.

Algabli, S. and Serratosa, F. (2018b). Embedding the node-to-node mappings to learn the graph edit distance parameters. *Pattern Recognit. Lett.*, 112:353–360.

Arias, J. F., Lai, C. P., Chandran, S., Kasturi, R., and Chhabra, A. K. (1995). Interpretation of Telephone System Manhole Drawings. *Pattern Recognition Letters*, 16(4):365–368.

Arroyo, E., Hoernicke, M., Rodríguez, P., and Fay, A. (2016). Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams. *Computers and Chemical Engineering*, 92:112–132.

Bajorath, J. (2001). Selected concepts and investigations in compound classification, molecular descriptor analysis, and virtual screening. *J. Chem. Inf. Comput. Sci.*, 41(2):233–245.

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

Barker, E. J., Buttar, D., Cosgrove, D. A., Gardiner, E. J., Kitts, P., Willett, P., and Gillet, V. J. (2006). Scaffold hopping using clique detection applied to reduced graphs. *J. Chem. Inf. Model.*, 46(2):503–511.

Barker, E. J., Gardiner, E. J., Gillet, V. J., Kitts, P., and Morris, J. (2003). Further development of reduced graphs for identifying bioactive compounds. *J. Chem. Inf. Comput. Sci.*, 43(2):346–356.

Barnard, J. M. (1993). Substructure searching methods: Old and new. *J. Chem. Inf. Comput. Sci.*, 33(4):532–538.

Bender, A. and Glen, R. C. (2004). Molecular similarity: a key technique in mol. inf. *Org. Biomol. Chem.*, 2(22):3204–3218.

Beno, B. R. and Mason, J. S. (2001). The design of combinatorial libraries using properties and 3d pharmacophore fingerprints. *Drug Discovery Today*, 6(5):251–258.

Birchall, K., Gillet, V. J., Harper, G., and Pickett, S. D. (2006). Training similarity measures for specific activities: application to reduced graphs. *J. Chem. Inf. Model.*, 46(2):577–586.

Blumenthal, D. B. and Gamper, J. (2018). On the exact computation of the graph edit distance. *Pattern Recognit. Lett.*, 0(0):1–12.

Boselli, R., Cesarini, M., Mercorio, F., and Mezzanzanica, M. (2018). Classifying online job advertisements through machine learning. *Future Generation Computer Systems*.

Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recogn. Lett.*, 1(4):245–253.

Caelli, T. and Kosinov, S. (2004). An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):515–519.

Caetano, T. S., McAuley, J. J., Cheng, L., Le, Q. V., and Smola, A. J. (2009). Learning graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 31(6):1048–1058.

Calvo-Zaragoza, J., Castellanos, F., Vigliensoni, G., and Fujinaga, I. (2018). Deep Neural Networks for Document Processing of Music Score Images. *Applied Sciences*, 8(5):654.

Cao, R. and Tan, C. L. (2002). Text/Graphics Separation in Maps. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 167–177.

Cereto-Massagué, A., Ojeda, M. J., Valls, C., Mulero, M., Garcia-Vallvé, S., and Pujadas, G. (2015). Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63.

Chaudhuri, A., Mandaviya, K., Badelia, P., and Ghosh, S. K. (2017). Optical character recognition systems. In *Optical Character Recognition Systems for Different Languages with Soft Computing*, pages 9–41. Springer.

Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298.

Conte, D. and Serratosa, F. (2020a). Interactive online learning for graph matching using active strategies. *Knowledge-Based Systems*, 205:106275.

Conte, D. and Serratosa, F. (2020b). Interactive online learning for graph matching using active strategies. *Knowl. Based Syst.*, 205:106275.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cortés, X., Conte, D., and Cardot, H. (2019). Learning edit cost estimation models for graph edit distance. *Pattern Recognition Letters*, 125:256–263.

Cortés, X., Conte, D., Cardot, H., and Serratosa, F. (2018). A deep neural network architecture to estimate node assignment costs for the graph edit distance. In Cortés et al. (2018), pages 326–336.

Cortés, X. and Serratosa, F. (2015). An interactive method for the image alignment problem based on partially supervised correspondence. *Expert Systems With Applications*, 42(1):179–192.

Cortés, X. and Serratosa, F. (2015). Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recognition Letters*, 56:22–29.

Cortés, X. and Serratosa, F. (2016a). Cooperative pose estimation of a fleet of robots based on interactive points alignment. *Expert Systems With Applications*, 45:150–160.

Cortés, X. and Serratosa, F. (2016b). Learning graph matching substitution weights based on the ground truth node correspondence. *International Journal of Pattern Recognition and Artificial Intelligence*, 30(02):1650005.

Cortés, X. and Serratosa, F. (2016). Learning graph matching substitution weights based on the ground truth node correspondence. *Int. J. Pattern Recognit. Artif. Intell.*, 30(2):1650005:1–1650005:22.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3837–3845.

Duin, R. P. W. and Pekalska, E. (2011). The dissimilarity representation for structural pattern recognition. In *CIARP*, volume 7042 of *Lecture Notes in Computer Science*, pages 1–24. Springer.

Elyan, E., Moreno-García, C. F., and Jayne, C. (2018). Symbols classification in engineering drawings. In *International Joint Conference on Neural Networks (IJCNN)*.

Fan, W., Wang, X., and Wu, Y. (2013). Diversified top-k graph pattern matching. *Proc. VLDB Endow.*, 6(13):1510–1521.

Ferrer, M., Serratosa, F., and Riesen, K. (2015). Improving bipartite graph matching by assessing the assignment confidence. *Pattern Recognition Letters*, 65:29–36.

Fisanick, W., Lipkus, A. H., and Rusinko, A. (1994). Similarity Searching on CAS Registry Substances. 2. 2D Structural Similarity. *J. Chem. Inf. Comput. Sci.*, 34(1):130–140.

Foggia, P., Percannella, G., and Vento, M. (2014). Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(01):1450001.

Gao, X., Xiao, B., Tao, D., and Li, X. (2010a). A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129.

Gao, X., Xiao, B., Tao, D., and Li, X. (2010b). A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129.

Garcia-Hernandez, C., Fernández, A., and Serratosa, F. (2019). Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of chemical information and modeling*, 59(4):1410–1421.

Garcia-Hernandez, C., Fernández, A., and Serratosa, F. (2020). Learning the edit costs of graph edit distance applied to ligand-based virtual screening. *Current topics in medicinal chemistry*, 20(18):1582–1592.

Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.

Gatica, E. A. and Cavasotto, C. N. (2011). Ligand and decoy sets for docking to g protein-coupled receptors. *J. Chem. Inf. Model.*, 52(1):1–6.

Gellaboina, M. K. and Venkoparao, V. G. (2009). Graphic symbol recognition using auto associative neural network model. In *Proceedings of the 7th International Conference on Advances in Pattern Recognition, ICAPR 2009*, pages 297–301.

Gibert, J., Valveny, E., and Bunke, H. (2012). Graph embedding in vector spaces by node attribute statistics. *Pattern Recognition*, 45(9):3072–3083.

Gillet, V. J., Downs, G. M., Holliday, J. D., Lynch, M. F., and Dethlefsen, W. (1991). Computer storage and retrieval of generic chemical structures in patents. 13. reduced graph generation. *J. Chem. Inf. Comput. Sci.*, 31(2):260–270.

Gillet, V. J., Willett, P., and Bradshaw, J. (2003). Similarity searching using reduced graphs. *J. Chem. Inf. Comput. Sci.*, 43(2):338–345.

Gou, G. and Chirkova, R. (2008). Efficient algorithms for exact ranked twig-pattern matching over graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, page 581–594, New York, NY, USA. Association for Computing Machinery.

Güner, O. F. (2000). *Pharmacophore perception, development, and use in drug design*, volume 2. Internat'l University Line.

Habi, A., Effantin, B., and Kheddouci, H. (2019). Diversified top-k search with relaxed graph simulation. *Social Network Analysis and Mining*, 9:1–15.

Harper, G., Bravi, G. S., Pickett, S. D., Hussain, J., and Green, D. V. S. (2004). The reduced graph descriptor in virtual screening and data-driven clustering of high-throughput screening data. *J. Chem. Inf. Comput. Sci.*, 44(6):2145–2156.

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

Heikamp, K. and Bajorath, J. (2013). The future of virtual compound screening. *Chem. Biol. Drug Des.*, 81(1):33–40.

Howie, C., Binford, T., Chen, J., Kunz, J., and Law, K. H. (1995). Computer Interpretation of Process and Instrumentation Drawings (Progress Report). Technical report.

Howie, C., Kunz, J., Binford, T., Chen, T., and Law, K. H. (1998). Computer Interpretation of Process and Instrumentation Drawings. *Advances in Engineering Software*, 29(7-9):563–570.

James, C. and Weininger, D. (1995). Daylight, 4.41 theory manual. *Daylight Chemical Information Systems Inc., Irvine, CA, USA.*

Johnson, M. A. and Maggiora, G. M. (1990). *Concepts and applications of molecular similarity.* Wiley.

Justice, D. and III, A. O. H. (2006). A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214.

Kang, S.-O., Lee, E.-B., and Baek, H.-K. (2019). A digitization and conversion tool for imaged drawings to intelligent piping and instrumentation diagrams (P&ID). *Energies*, 12(2593):1–26.

Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., and Tao, S. (2011). Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 901–912, New York, NY, USA. Association for Computing Machinery.

Kirchmair, J., Distinto, S., Markt, P., Schuster, D., Spitzer, G. M., Liedl, K. R., and Wolber, G. (2009). How to optimize shape-based virtual screening: choosing the right query and including chemical information. *J. Chem. Inf. Model.*, 49(3):678–692.

Kubinyi, H., Mannhold, R., and Timmerman, H. (2008). *Virtual screening for bioactive molecules*, volume 10. John Wiley & Sons.

Lagarde, N., Ben Nasr, N., Jeremie, A., Guillemain, H., Laville, V., Labib, T., Zagury, J.-F., and Montes, M. (2014). Nrlist bdb, the manually curated nuclear receptors ligands and structures benchmarking database. *J. Med. Chem.*, 57(7):3117–3125.

Lajiness, M. (1990). Molecular similarity-based methods for selecting compounds for screening. In *Computational chemical graph theory*, pages 299–316. Nova Science Publishers, Inc.

Leordeanu, M., Sukthankar, R., and Hebert, M. (2012). Unsupervised learning for graph matching. *International journal of computer vision*, 96(1):28–45.

Livi, L. and Rizzi, A. (2013). The graph matching problem. *Pattern Analysis and Applications*, 16(3):253–283.

Livingstone, D. J. (2000). The characterization of chemical structures using molecular properties. a survey. *J. Chem. Inf. Comput. Sci.*, 40(2):195–209.

Luqman, M. M., Ramel, J.-Y., Lladós, J., and Brouard, T. (2013). Fuzzy multilevel graph embedding. *Pattern Recogn.*, 46(2):551–565.

Martineau, M., Raveaux, R., Conte, D., and Venturini, G. (2018). Learning error-correcting graph matching with a multiclass neural network. *Pattern Recognition Letters*.

McGregor, M. J. and Pallai, P. V. (1997). Clustering of large databases of compounds: using the mdl "keys" as structural descriptors. *J. Chem. Inf. Comput. Sci.*, 37(3):443–448.

Melville, J. L., Burke, E. K., and Hirst, J. D. (2009). Machine learning in virtual screening. *Comb. Chem. High Throughput Screening*, 12(4):332–343.

Menard, P. R., Mason, J. S., Morize, I., and Bauerschmidt, S. (1998). Chemistry space metrics in diversity analysis, library design, and compound selection. *J. Chem. Inf. Comput. Sci.*, 38(6):1204–1213.

Moreno, C. and Serratosa, F. (2015). Consensus of multiple correspondences to increase the accuracy in image registration. *Comput. Vis. Image Underst.*

Moreno-García, C. F. (2018). Digital interpretation of sensor-equipment diagrams. In *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability (ReaLX 2018).*

Moreno-García, C. F., Cortés, X., and Serratosa, F. (2016). A graph repository for learning error-tolerant graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 519–529. Springer.

Moreno-García, C. F. and Elyan, E. (2019). Digitisation of Assets from the Oil & Gas Industry: Challenges and Opportunities. In *International Conference on Document Analysis and Recognition (ICDAR)*, number Workshop on Industrial Applications of Document Analysis and Recognition (WIADAR), pages 16–19.

Moreno-García, C. F., Elyan, E., and Jayne, C. (2017). Heuristics-Based Detection to Improve Text / Graphics Segmentation in Complex Engineering Drawings. In *Engineering Applications of Neural Networks*, volume CCIS 744, pages 87–98.

Moreno-García, C. F., Elyan, E., and Jayne, C. (2019). New trends on digitisation of complex engineering drawings. *Neural Computing and Applications*, 31(6):1695–1712.

Munkres, J. (1957). Algorithms for the Assignment and Transportation Problems. *J. Soc. Ind. Appl. Math.*, 5(1):32–38.

Mysinger, M. M., Carchia, M., Irwin, J. J., and Shoichet, B. K. (2012). Directory of useful decoys, enhanced (dud-e): better ligands and decoys for better benchmarking. *J. Med. Chem.*, 55(14):6582–6594.

Neuhaus, M. and Bunke, H. (2005). Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):503–514.

Neuhaus, M. and Bunke, H. (2007). Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247.

Neuhaus, M., Riesen, K., and Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 163–172. Springer.

Nikolova, N. and Jaworska, J. (2003). Approaches to measure chemical similarity–a review. *Mol. Inf.*, 22(9-10):1006–1026.

Pearlman, R. S. and Smith, K. M. (1999). Metric validation and the receptor-relevant subspace concept. *J. Chem. Inf. Comput. Sci.*, 39(1):28–35.

Rahul, R., Paliwal, S., Sharma, M., and Vig, L. (2019). Automatic Information Extraction from Piping and Instrumentation Diagrams. In *International Conference on Pattern Recognition Applications and Methods (ICPRAM)*, pages 163–172.

Rantala, M., Niemistö, H., Karhela, T., Sierla, S., and Vyatkin, V. (2019). Applying graph matching techniques to enhance reuse of plant design information. *Computers in Industry*, 107:81–98.

Rarey, M. and Dixon, J. S. (1998). Feature trees: a new molecular similarity measure based on tree matching. *J. Comput.-Aided Mol. Des.*, 12(5):471–490.

Rica, E., Álvarez, S., and Serratosa, F. (2019). On-line learning the edit costs based on an embedded model. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 121–130. Springer.

Rica, E., Álvarez, S., and Serratosa, F. (2021). On-line learning the graph edit distance costs. *Pattern Recognit. Lett.*, 146:55–62.

Rica, E., Moreno-García, C. F., Álvarez, S., and Serratosa, F. (2020). Reducing human effort in engineering drawing validation. *Comput. Ind.*, 117:103198.

Riesen, K. and Bunke, H. (2009). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959.

Riesen, K., Fischer, A., and Bunke, H. (2018). On the impact of using utilities rather than costs for graph matching. *Neural Processing Letters*, 48(2):691–707.

Rohrer, S. G. and Baumann, K. (2009). Maximum unbiased validation (muv) data sets for virtual screening based on pubchem bioactivity data. *J. Chem. Inf. Model.*, 49(2):169–184.

Sanders, M. P., Barbosa, A. J., Zarzycka, B., Nicolaes, G. A., Klomp, J. P., de Vlieg, J., and Del Rio, A. (2012). Comparative analysis of pharmacophore screening tools. *J. Chem. Inf. Model.*, 52(6):1607–1620.

Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., and Vergés-Llahí, J. (2002). Graph-based representations and techniques for image processing and image analysis. *Pattern Recognition*, 35:639–650.

Sanfeliu, A. and Fu, K.-S. (1983a). A distance measure between attributed relational graphs for pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-13.

Sanfeliu, A. and Fu, K.-S. (1983b). A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362.

Santacruz, P. and Serratosa, F. (2018a). Error-tolerant graph matching in linear computational cost using an initial small partial matching. *Pattern Recognit. Lett.*, 0(0):1–10.

Santacruz, P. and Serratosa, F. (2018b). Learning the sub-optimal graph edit distance edit costs based on an embedded model. In *S+SSPR*, volume 11004 of *Lecture Notes in Computer Science*, pages 282–292. Springer.

Santacruz, P. and Serratosa, F. (2019). Learning the graph edit costs based on a learning model applied to sub-optimal graph matching. *Neural Processing Letters*, pages 1–24.

Santacruz, P. and Serratosa, F. (2020). Learning the graph edit costs based on a learning model applied to sub-optimal graph matching. *Neural Process. Lett.*, 51(1):881–904.

Schneider, G., Clément-Chomienne, O., Hilfiger, L., Schneider, P., Kirsch, S., Böhm, H.-J., and Neidhart, W. (2000). Virtual screening for bioactive molecules by evolutionary de novo design. *Angew. Chem. Int. Ed.*, 39(22):4130–4133.

Schnur, D. (1999). Design and diversity analysis of large combinatorial libraries using cell-based methods. *J. Chem. Inf. Comput. Sci.*, 39(1):36–45.

Serratosa, F. (2014a). Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250.

Serratosa, F. (2014b). Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence*, 29:1550010.

Serratosa, F. (2015). Computation of graph edit distance: Reasoning about optimality and speed-up. *Image Vision Comput.*, 40:38–48.

Serratosa, F. (2021). Redefining the graph edit distance. *SN Computer Science*, 2(6):1–7.

Serratosa, F. and Cortés, X. (2015a). Graph edit distance. *Pattern Recogn. Lett.*, 65(C):204–210.

Serratosa, F. and Cortés, X. (2015b). Graph edit distance: Moving from global to local structure to solve the graph-matching problem. *Pattern Recognition Letters*, 65:204–210.

Skoda, P. and Hoksza, D. (2017). Benchmarking platform for ligand-based virtual screening. In *Proceedings - 2016 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2016*, pages 1220–1227.

Solé, A., Serratosa, F., and Sanfeliu, A. (2012). On the graph edit distance cost: Properties and applications. intern. *Intern. Journal of Pattern Recognit. and Artificial Intell.*, 26(5).

Solé-Ribalta, A., Serratosa, F., and Sanfeliu, A. (2012). On the graph edit distance cost: Properties and applications. *IJPRAI*, 26(5).

Stauffer, M., Tschachtli, T., Fischer, A., and Riesen, K. (2017). A survey on applications of bipartite graph edit distance. In *GbRPR*, volume 10310 of *Lecture Notes in Computer Science*, pages 242–252.

Stephen, I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 50(2):179.

Stiefl, N., Watson, I. A., Baumann, K., and Zaliani, A. (2006). Erg: 2d pharmacophore descriptions for scaffold hopping. *J. Chem. Inf. Model.*, 46(1):208–220.

Sun, H. (2008). Pharmacophore-based virtual screening. *Curr. Med. Chem.*, 15(10):1018–1024.

Takahashi, Y., Sukekawa, M., and Sasaki, S. (1992). Automatic identification of molecular similarity using reduced-graph representation of chemical structure. *J. Chem. Inf. Model.*, 32(6):639–643.

Tan, W. C., Chen, I. M., and Tan, H. K. (2016). Automated identification of components in raster piping and instrumentation diagram with minimal preprocessing. *IEEE International Conference on Automation Science and Engineering*, November:1301–1306.

Tombre, K., Tabbone, S., Lamiroy, B., and Dosch, P. (2002). Text/Graphics Separation Revisited. In *Document Analysis Systems*, volume 2423, pages 200–211.

Vaxiviere, P. and Tombre, K. (1992). Celesstin: CAD Conversion of Mechanical Drawings. *IEEE Computer Magazine*, 25(7):46–54.

Vento, M. (2013). A one hour trip in the world of graphs, looking at the papers of the last ten years. In Kropatsch, W. G., Artner, N. M., Haxhimusa, Y., and

Jiang, X., editors, *Graph-Based Representations in Pattern Recognition*, pages 1–10, Berlin, Heidelberg. Springer Berlin Heidelberg.

Vento, M. (2015). A long trip in the charming world of graphs for pattern recognition. *Pattern Recognition*, 48(2):291–301.

Willett, J. (1987). *Similarity and clustering in chemical information systems.* John Wiley & Sons, Inc.

Willett, P. (2004). Evaluation of molecular similarity and mol. diversity methods using biological activity data. In *Chemoinformatics*, pages 51–63. Springer.

Xanthopoulos, P., Pardalos, P. M., and Trafalis, T. B. (2013). Linear discriminant analysis. In *Robust data mining*, pages 27–33. Springer.

Xia, J., Tilahun, E. L., Reid, T.-E., Zhang, L., and Wang, X. S. (2015). Benchmarking methods and data sets for ligand enrichment assessment in virtual screening. *Methods*, 71:146–157.

Xue, L. and Bajorath, J. (2000). Molecular descriptors in chemoinformatics, computational combinatorial chemistry, and virtual screening. *Comb. Chem. High Throughput Screening*, 3(5):363–372.

Yang, Z., Fu, A. W.-C., and Liu, R. (2016). Diversified top-k subgraph querying in a large graph. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1167–1182, New York, NY, USA. Association for Computing Machinery.

Yu, Y., Samal, A., and Seth, S. C. (1997). A system for recognizing a large class of engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):868–890.

Zhang, X., Yang, T., and Srinivasan, P. (2016). Online asymmetric active learning with imbalanced data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2055–2064.

Zhao, P. and Hoi, S. C. (2013). Cost-sensitive online active learning with application to malicious url detection. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 919–927.

Zou, L., Chen, L., and Lu, Y. (2007). Top-k subgraph matching query in a large graph. In Varde, A. S. and Pei, J., editors, *Proceedings of the First Ph.D. Workshop in CIKM, PIKM 2007, Sixteenth ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 9, 2007*, pages 139–146. ACM.

UNIVERSITAT
ROVIRA i VIRGILI