Group of Components Detection in Engineering Drawings based on Graph Matching

Elena Rica*, Susana Álvarez, Francesc Serratosa

Universitat Rovira i Virgili

Departament d'Enginyeria Informàtica i Matemàtiques

Avinguda dels Països Catalans, 26, Tarragona, Spain

Abstract

Computer-aided design (CAD) applications are paramount tools to inspect piping and instrumentation diagrams (P&ID) that represent the structure and functionality of oil and gas facilities. These tools allow engineers to navigate along the P&IDs and facilitate their inspection and maintenance tasks. Usually, these applications allow the user to search components for their identity or label. For instance, searching component $ID - 2354$ or the components with $Valve$ label. Nevertheless, sometimes, engineers wish to search for a group of components that has a specific structure or similar to it. For example, they want to detect the appearances of structures that include a *valve check* connected to two *general valves* and a *butterfly valve*. This paper proposes a method, based on the graph matching theory, that solves this problem. The computational complexity to detect the appearance of a component in a P&ID is linear with respect to the number of components whereas the computational complexity to detect similar structures is exponential. For this reason, heuristic algorithms have to be used. The aim of this work is to add this functionality to a CAD application.

Keywords: Engineering Drawing inspection, Graph Matching, Piping and instrumental Diagram (P&ID)

*Corresponding author

Email address: mariaelena.rica@urv.cat (Elena Rica)

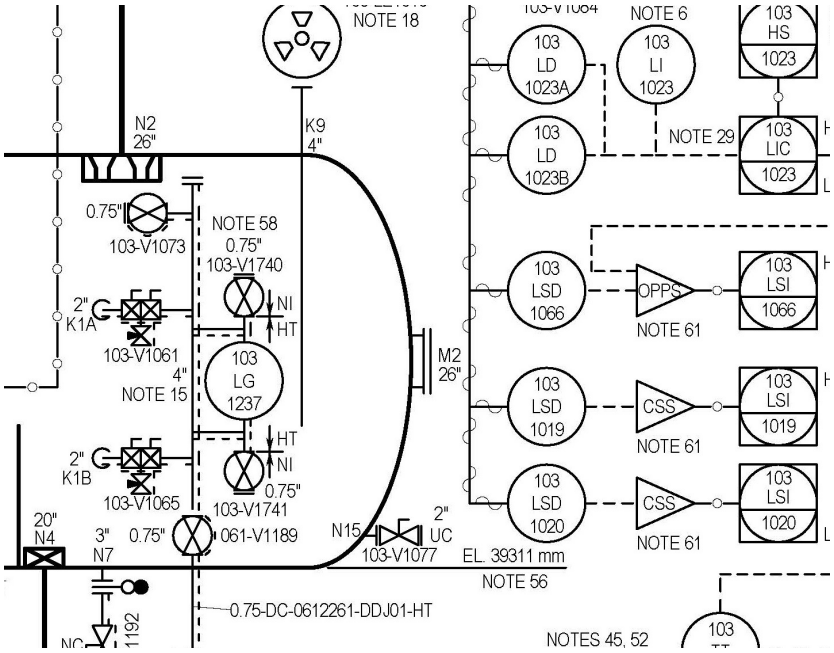


Figure 1: An example of a portion of a P&ID.

1. Introduction

Piping and Instrumentation Diagrams (P&IDs) are commonly used for representing the structure and functionality of oil and gas facilities such as oil rigs and plants. These complex engineering drawings are generated and inspected by means of computer-aided design (CAD) tools.

In the past, they were drawn in books composed of thousands of sheets, which during the last years have been digitised using several tools [1, 2]. Since these facilities are huge and composed of thousands of electric, electronic or mechanical components connected by a vast network of pipelines, working on the CAD tools has been an important advance, compared to using those huge books composed of a large number of sheets. Figure 1 shows a tiny portion of a P&IDs to illustrate the complexity of the structure of these oil and gas facilities.

CAD applications usually have some tools to search for specific components

by their identity number and visualise them in their locations at the P&ID [3, 4, 5, 6, 7, 8]. Nevertheless, the gas facilities have thousands of some specific components making the search task overwhelming.

While inspecting or analysing the gas facilities, engineers are usually interested in some structures, composed of a small set of connected components, instead of a specific type of component. For instance, they want to detect the appearances of structures that include a *valve check* connected to two *general valves* and a *butterfly valve*. Thus, engineers want to query a structure instead of a component and visualise it in the P&ID. Note that the aim of this last query is not to return the locations of the exact appearance of the specific structure but the locations in which appear structures that could be similar to it. Thus, considering the example above, it could be interesting to return the locations of structures composed of a *valve check* connected to one or three *general valves* and one or two *butterfly valves*.

This problem fits with the topology challenge of P&IDs contextualisation [9], [10], whose objective is to understand the topology and connections of P&IDs. As commented before, there are some CAD applications[1] to work with P&IDs, which have online tools to quickly visualise the network of components, such as NetVis or Netlist2CAD[2]. Some of these tools offer the option to search a component and its adjacent components, but none of them gives the possibility to look for sub-structure and return the similar ones to it. Therefore, the proposed method could be included into a CAD application.

If we want to search for structures in the P&ID similar to a query structure, we need to define a distance between them. The more similar the structures are, the shorter the distance is. An interesting option to define this distance is to represent the P&ID as a graph and the structure we are looking for as a smaller graph, since graphs are mathematical representations of elements composed of local parts. Thus, the similarity between structures is defined as the inverse of

[1] `https://www.geminivalve.com/best-piping-design-software/`

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

a distance between graphs.

Note, moving from detecting the appearance of a component in a P&ID to detecting the appearance of similar structures in a P&ID makes the problem to be much more complex. In the first case, it can be solved in a linear time with respect to the number of components in the P&ID. In the second case, we are facing an NP-problem (it is not solvable in polynomial computational cost with respect to the number of components), worsened by the fact that P&ID have thousands of components.

We present a method that given a P&ID and a small set of connected components, returns K locations in the P&ID where similar sets of components appear. Moreover, the method returns them ordered by the distance between the query structure and the detected one in ascending order. Figure 2 shows a screen shot of the application. The window application is composed of three parts. In the upper left corner, the structure query is edited. In the lower left corner, the returned structures ordered by increasing distance are shown. In the right corner, the P&ID is visualised and the detected structures are highlighted.

This research is part of a larger project in which we deepen on semi-supervised techniques applied to the gas facilities. These techniques are based on solving problems by introducing the human knowledge as part of the system. For instance, in [11], we presented a method that aims to make easier human inspection and validation of automatically digitised P&ID.

The paper is structured as follows. Section 2 presents how graphs can be used to represent and analyse structured objects. Section 3 presents our method to detect local structures in P&IDs. Section 4 presents the experiments carried out to validate the proposed model. Finally, Section 5 is reserved for conclusions and future work.

2. Graphs for representing and analysing structured objects

In this section, we introduce and summarise how graphs can be used to represent structured objects, as for instance P&IDs, and to analyse their content,

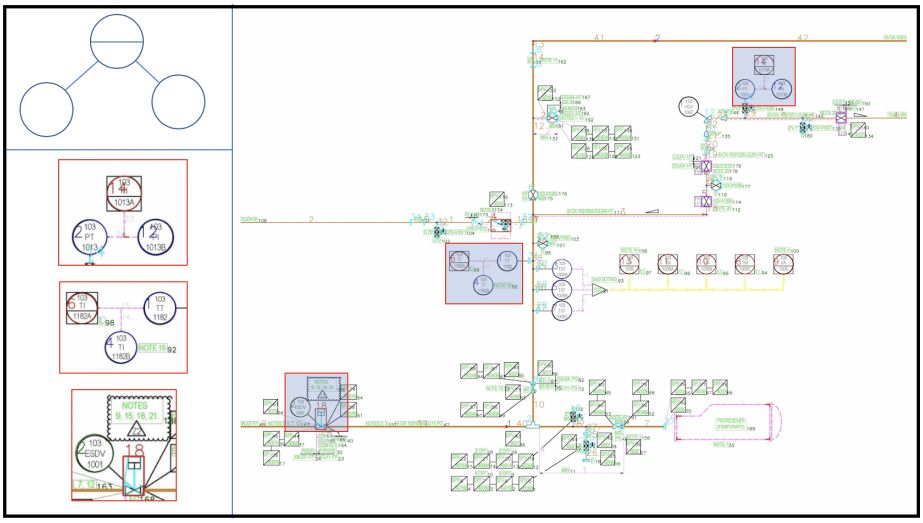


Figure 2: Screen shot of our application. Upper left corner: query structure. Lower left corner: returned structures ordered by increasing distance. Right: a portion of the P&ID in which three locations of the query have been detected.

as for example, locating some specific combinations of components connected by pipelines. The section is composed of two sub-sections. In the first one, we summarise the graph matching techniques and the computation of these techniques. Then, in the second sub-section, we summarise the techniques that query sub-graphs in a graph.

2.1. Graph Matching

Attributed graphs are mathematical representations of objects that have the ability to keep the information of the local parts of the objects. Their applicability ranges from automatic character recognition of handwritten characters [12] to toxicity analysis of chemical compounds [13], between others. These representations have been of crucial importance in pattern recognition throughout more than four decades, [14, 15, 16], since they have been used to model several kinds of problems. If objects in pattern recognition are modelled through attributed graphs, error-tolerant graph-matching algorithms to compute the distance between graphs and a mapping between nodes of both graphs that minimises some

objective function are needed. [17, 18].

Given an attributed graph G, we define v_a as the a^{th} node in G and we define $e_{a,b}$ as the edge between node v_a and node v_b in G. Similarly, given an 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.

One of the most used methods to define a distance between attributed graphs is the graph edit distance [16]. The graph edit distance between a pair of attributed graphs G and G' consists in finding the best sequence of edit operations that converts one graph into another [19]. There are three operations on the nodes and also on the edges: substitution, deletion and insertion. To quantify how much important these operations are while transforming the graphs, a cost is assigned to them depending on the attributes on the involved nodes or edges. The cost of substituting a node v_a by a node v'_i is $C_v^S(a,i)$. The cost of deleting a node v_a is $C_v^D(a)$. Finally, a 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 $C_e^I(i,j)$. Thus, the GED is the minimum transformation cost, given all transformations from one graph into another.

A node-to-node mapping f between nodes of both graphs, G and G', can be used to define the mentioned graph transformation. Thus, $f(a) = i$ represents the mapping from node v_a to node v'_i. We suppose both graphs have the same number of nodes since they have been expanded with new nodes, called $Null$. $f(a) = i$ 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.

Computing the GED has an exponential computational cost in the number of nodes of the involved graphs [20]. For this reason, some heuristic algorithms have been presented that deduce a sub-optimal distance in polynomial time [21, 22, 23, 24, 25]. These algorithms define the edit cost of two graphs given a specific node-to-node mapping f between nodes of both graphs as the addition

of the substitution, deletion and insertion costs of the local structure of nodes.

$$Cost(G, G', f) = \sum_{\forall Subs_v} C^S_{Star}(a, i) + \sum_{\forall Del_v} C^D_{Star}(a) + \sum_{\forall Ins_v} C^I_{Star}(i) \quad (1)$$

Usually, the local structure of nodes are composed of the node itself, its connected edges and the nodes these edges connect. These structures are called *Stars* [26]. $Subs_v$, Del_v and Ins_v represent the substitutions, deletions and insertions of nodes respectively. Moreover, $C^S_{Star}(a, i)$ denotes the cost of substituting the *Star* centred at node v_a by the *Star* centred at node v'_i. $C^D_{Star}(a)$ 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 $C^I_e(i, j)$ (See [26]).

Finally, the graph edit distance is defined as:

$$GED(G, G') = \min_{\forall f}(Cost(G, G', f)) \quad (2)$$

One of the newest graph matching algorithms that have been published is *Belief Propagation* algorithm [27]. Its computational cost is only linear with respect to the number of nodes, although in some applications its accuracy could be too low. Moreover, it needs some initial node-to-node mappings, which are called *Seeds*.

Our method is based on minimising the cost in Equation 1 through *Belief* graph matching algorithm [27]. We use this algorithm because it has two main properties not found in the other algorithms:

- It is the algorithm that has the lowest computational cost. This property is important since P&ID could have thousands of components.
- The substituted nodes in G', which are the query result, form a compact graph. This property is important since the engineer desires to visualise compact sub-structures in the P&ID.

The Belief graph matching algorithm is composed of three main parts:

- An initial mapping between a pair of nodes is imposed and introduced into the *pending cue*. Moreover, the costs between their stars, C^S_{Star}, is computed and also the node-to-node mapping between the external nodes of both stars.

- Iterate this step until the *pending cue* is empty. The node-to-node mapping that has the minimum C^S_{Star} in the *pending cue* is extracted from it and marked as part of the final correspondence. The mapped external nodes of the stars of these nodes are introduced in the *pending cue* and their C^S_{Star} are computed.

- The non-considered nodes of both graphs are added into the final correspondence as deleted or inserted nodes.

2.2. Top-K Sub-graph Search

Finding the appearances of a graph into a larger graph has been widely used in some applications and currently, social data analysis may be the most common application. Usually, this problem is called *Top-k sub-graph search* and consists in finding K appearances of a query graph in a commonly larger graph. This issue has been addressed in two different directions. The first one is finding the exact appearances of the query graph. Then, the algorithm seeks K exact sub-graph isomorphisms [28, 29]. The second one is finding sub-graphs in the larger graph that are similar to the query graph. Then the algorithm seeks the K sub-graphs in the larger graph that have the shortest distance between them and the query graph. In this case, the algorithm returns what is usually called non-exact sub-graph isomorphism [30, 31, 32, 33].

We are interested in the second type of methods because engineers search for similar appearances of the query graph, but not only exact appearances. Nevertheless, methods presented in [30, 31, 32, 33] and similar ones cannot be applied to solve our problem because they assume that connected nodes tend to be similar but, clearly, components connected by pipelines in a P&ID can have

completely different properties. For this reason, we present our new method which has been called *Top-K-GED*.

3. Top-K-GED for local Structure Detection

Our method 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 component identity (valve, compressor,...). Usually, in the graph matching field, this type of attribute is called *Label*. Edges are unattributed and undirected since we do not have information about the type of pipelines or their features. Moreover, our method defines the local structure to be queried as another graph that has the same features as the P&ID but it is much smaller.

Before explaining the algorithm, we set the following three premises that will condition the values of its input parameters:

- Only engineers know how similar are two compounds in the P&ID. This knowledge is introduced into the system through the cost of substituting nodes imposed by the engineers before doing the query. For instance, if the engineer queries a graph that has a *valve* and wants to visualise all the sub-graphs similar to this query that have this *valve*, then one has to impose the substitution cost between a *valve* and the rest of compounds to be infinite. Contrarily, if one knows that two compounds are similar, then one can consider the substitution cost between them to be zero.

- In a similar way than the previous item, only engineers know how important is a compound or a pipeline in the P&ID. This knowledge is considered in the node and edge deletion and insertion costs.

- Engineers want to visualise the locations where the query or similar queries appear in the P&ID. For this reason, it is desired that the method returns several connected compounds in the P&ID. These restrictions need to be handled by the search algorithm.

This section has been divided into two sub-sections, In Section 3.1, the input and output parameters of our algorithm are defined and the general edit costs defined in Section 2.1 are adapted to our problem. Finally, in Section 3.2 our algorithm is detailed.

3.1. Input and output parameters of our method

Considering the above premises, and assuming v_a and v_b are two nodes of Q and, v'_i and v'_j are two nodes of G, the **input** of our method is composed of:

- Q: A small graph that represents the query.
- G: A large graph that represents the P&ID.
- S_v: A square matrix of node substitution costs previously set by the engineer. Each cell is a Real non-negative number that represents the cost of substituting two types of components and depends on the attribute of the nodes. $S_v(a,i) = C_v^S(a,i)$. All the elements in the diagonal are zeros.
- D_v: A vector of node deletion costs previously set by the engineer. Each cell is a Real non-negative number that depends on each specific component. $D_v(a) = C_v^D(a)$.
- D_e: 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, D_e, previously set by the engineers, $D_e = C_e^D(a,b)$.
- K: The number of compact sub-graphs the method has to return.

Note the edge substitution cost is not an input parameter since edges do not have attributes. Moreover, the edge and node insertion costs are not input parameters either since they equal zero, to assure the number of components in the P&ID does not influence on the final distance.

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

- $\{f_1 : Q \rightarrow G$,..., $f_K : Q \rightarrow G\}$. A list of K node-to-node mappings between the query graph Q and the P&ID graph G.

- $\{Cost(Q, G, f_1)$,..., $Cost(Q, G, f_K)\}$. A list of K edit costs (Equation 1), given the query graph, the P&ID 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, $GED(Q, G_p) = Cost(Q, G, f_p)$ where GED is defined in Equation 2.

The returned list of mappings f_1,...,f_K hold the following four conditions:

- For each $1 \leq p \leq K$, the set of nodes $G(f_p(v))$, $\forall v \in Q$, defines a connected sub-graph.

- Components in P&ID can be reached by several mappings. Nevertheless, two mappings cannot be identical. Formally: $If\ f_p(v) = f_q(v), \forall v \in Q \Rightarrow p = q$.

- The mappings f_1,...,f_K are listed in ascending order on their edit costs. Formally: $Cost(Q, G, f_1) \leq Cost(Q, G, f_2)$,..., $\leq Cost(Q, G, f_K)$.

- The mappings f_1,...,f_K might have to be the ones that return the minimum cost. Formally: $If\ f \notin \{f_1, ..., f_K\} \Rightarrow Cost(Q, G, f) \geqslant Cost(Q, G, f_p)$, $\forall p = 1, ..., K$. Note this last imposition cannot always be guaranteed due to our algorithm does not always return the optimal solution.

3.2. Algorithm

In this section, we explain *Top-K-GED* algorithm, that has three main steps.

In the first one, a cost matrix C is computed with dimensions mXn, where m and n are the number of nodes of the query graph Q and the P&ID graph G, respectively. We assume, $m \leq n$. 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:

$$C(a, i) = C^S_{Star}(a, i) \quad 1 \leq a \leq m \quad and \quad 1 \leq i \leq n \tag{3}$$

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 algorithm *Belief* 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,

$$Seed_p = (a, i), \quad being \quad C_p = C(a, i) \tag{4}$$

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 slightly 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 P&ID G. This property could be considered a drawback in other methods but it becomes a must in our case.

In the original *Belief* algorithm, as it is explained in [27], it computes only some cells of the cost matrix C. This is the reason why it has a linear cost, instead of a quadratic cost. Since, in our case, we need the cost matrix to be computed to deduce the *Seeds*, in our adaptation, an input parameter of *Belief* algorithm is C instead of the edit costs. Thus, avoiding computing some of the cells in C in the third step that had been computed in the first step. If we do not want to consider this optimisation, the call of *Belief* algorithm would have to be $Belief(Q, G, S_v, D_v, D_e, Seed_p)$ instead of the call $Belief(C, Seed_p)$ that we propose in *Top-K-GED* algorithm. Matlab implementation in [3].

Algorithm *Top-K-GED*

Input: Q, G, S_v, D_v, D_e, K

Output: f_1, ... , f_K, $Cost_1$, ... , $Cost_K$, being $Cost_p = Cost(Q, G, f_p)$

Begin Algorithm

$C = ComputeCostMatrix(Q, G, S_v, D_v, D_e)$

$(Seed_1, ..., Seed_K) = SelectLowerCostCells(C, K)$

$For\ p = 1..K$

$(f_p, Cost_p) = Belief(C, Seed_p)$

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

End For

End Algorithm

4. Experimental Validation

The experimental validation is detailed in five sub-sections. In the first two ones, we analyse the influence of the graph matching algorithm and the edit costs on the query results. In the other three sections, we detail the P&IDs database used in the experiments, we show two examples of query results and finally we evaluate the performance of our proposal.

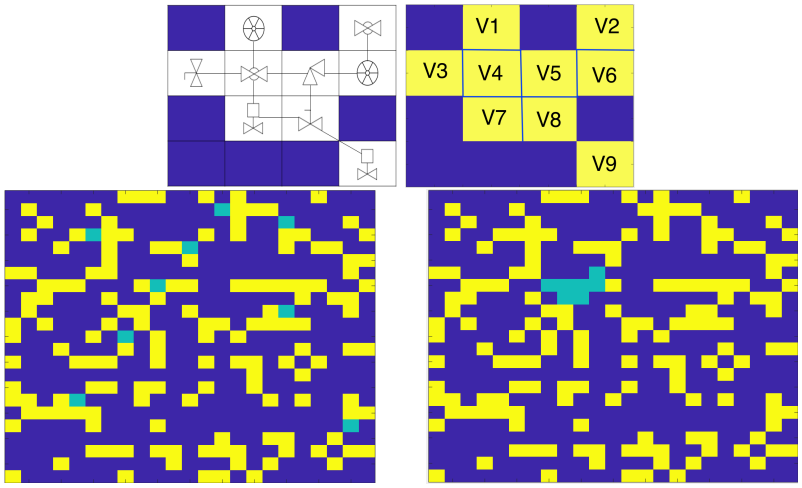
4.1. Analysis of the graph matching algorithm

The aim of this section is to heuristically compare two graph matching algorithms and to show that *Belief* graph matching [27] is the best option between them since it returns a compact sub-graph.

Figure 3 shows a query graph with nine nodes (components) and eight edges (pipelines) 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 two different matching algorithms. In the left, cyan cells are the components that have been mapped by *Fast Bipartite* algorithm [22] and in the right, cyan cells are the components that have been mapped by *Belief* algorithm [27].

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 our method, while it is not the case of the mapped sub-graph returned by the Fast Bipartite graph matching. Nevertheless, the GED (Eq. 2) might be lower in the Bipartite case than the *Belief* case since the *Bipartite* tends to deduce a less sub-optimal solution (the computational cost is cubic) than *Belief* (the computational cost is linear). In our application at hand, it is worth to deduce a more sub-optimal edit distance but a compact sub-graph than a less sub-optimal one. There are nine cyan components found by *Bipartite* graph matching, this means that it

has deduced the minimum cost by substituting all the nodes in the query. Contrarily, there are only seven cyan components found by *Belief* algorithm. This means that it has deduced the best node-to-node mapping which is composed of seven node substitutions and two node deletions. Finally, the algorithms that return an optimal distance [34] cannot be executed due to time restrictions. Moreover, returning an optimal edit distance does not mean returning a compact sub-graph.



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

Figure 3: 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, Bipartite algorithm [22] has been used and in the right, *Belief* algorithm [27]. Pipelines are not shown.

4.2. Analysis of the edit costs

In this section, we study how the different combinations of the edit costs affect the returned sub-graphs. To do so, we have to run *Belief* algorithm three times with $K = 3$ given the same query and 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 labels

equally distributed among the nodes. Thus, the node substitution cost S_v is a matrix of $20X20$ cells and the node deletion cost D_v is a vector of 20 cells.

In the three runs of the algorithm, S_v has been defined as a matrix of all ones except for the diagonal cells that equal zero. This means that we consider all the types of components to be equally different. Moreover, the 20 cells of D_v and D_e have been set as follows:

- In the first run: $D_e = D_v(a) = 0.1$, $1 \leq a \leq 20$.
- In the second run: $D_e = D_v(a) = 1$, $1 \leq a \leq 20$.
- In the third run: $D_e = D_v(a) = 10$, $1 \leq a \leq 20$.

Figure 4 shows the query composed of seven nodes in the upper left image. The three following other images in the figure show the P&ID and the returned three sub-graphs, f_1, f_2 and f_3, since $K = 3$, in each of the three runs. The last row in Figure 4 shows the nodes of the query graph that have been substituted by nodes in the P&ID. In the first case, an exact match has been found. Note the GED is independent of the rotation and this is the reason why the orientation of the query is different from the orientation of the first returned sub-graph.

In the first case, deleting a node and an edge is very cheap, 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 number of substituted nodes increases as it does the cost of deleting nodes and edges. Only in the last run, the detected sub-graph has the same structure than the query, which means that all the nodes have been substituted and any node is deleted. This sub-graph does not emerge in the other previous runs because the cost of substituting the nodes was larger than deleting some of them and only substituting the ones that had the same label.

It is unlikely the user desires the results obtained in the first two runs because the number of deleted nodes is too high and the structures seem to be completely different. This is the reason why it is usual to define the deletion costs higher than the substitution costs, as it is done in the third run.

In general, D_e, $D_v(a)$ and $S_v(a)$ are not important per se, but their relation. Note that if all of these values increase or are multiplied by a Real number, the

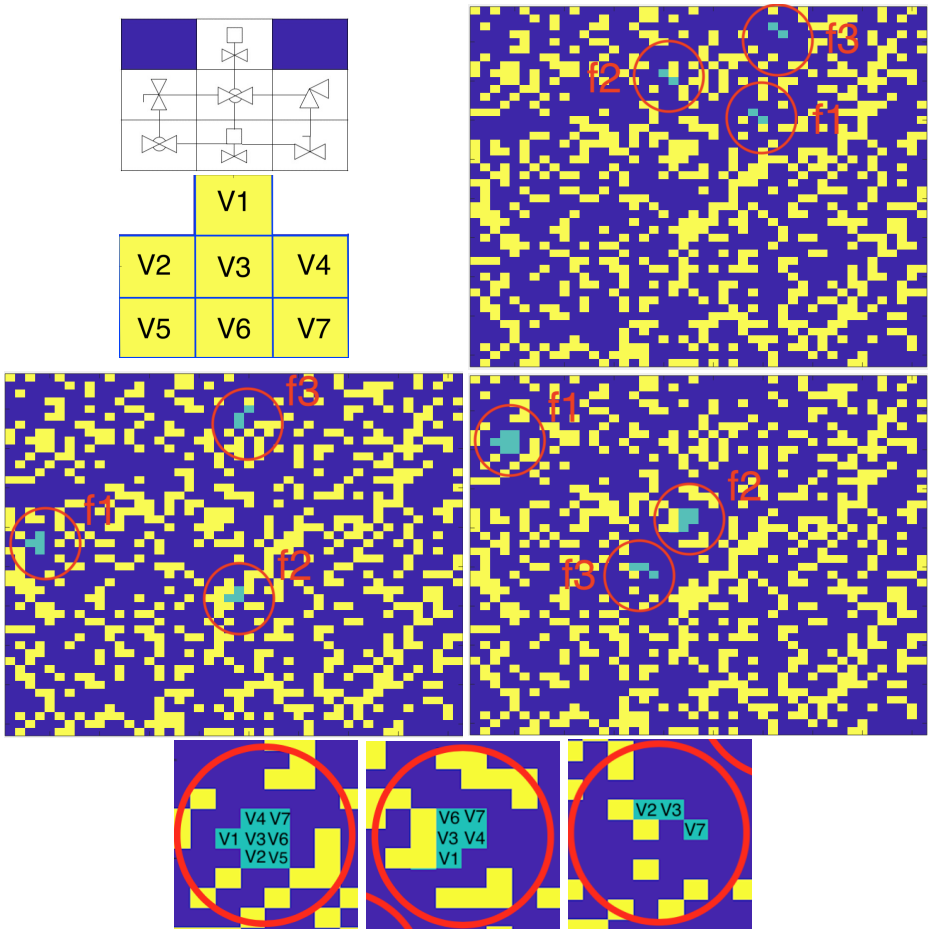


Figure 4: Upper left: The query graph and its representation based on its spacial positions. Upper right and second row: The P&ID components represented by yellow cells and the three returned mappings in each of the three runs of our algorithm are highlighted in cyan. Upper right image: First run. Middle left image: Second run. Middle right image: Third run. Lower row: A detail of the three mappings deduced in the last run. From left to right: f_1, f_2 and f_3.

final optimal node-to-node correspondence will be the same but the GED (2) will increase or be multiplied by this number. This is because (2) is a linear equation. Nevertheless, the increase or decrease of one of these values will make the optimal correspondence to be different and therefore, the resulting GED. In [19], it is empirically demonstrated that the best results are achieved when $D_e = D_v(a) = 0.5S_v(a)$, given some databases on image identification or character recognition. In those applications, breaking the graph by deleting the edges is a frequent operation. Contrarily, in our application, users aim to visualise graphs almost structurally equal to the ones proposed by them. For this reason, in the previous experiment, the best results appeared when $D_e = D_v(a) = 10S_v(a)$. This way, we prioritise substitutions instead of deletion of connections and components. We have realised that numbers larger than 10 make the method to only return graphs that are structurally isomorphic. Usually, the same type of components is connected to the other components in the same way. For this reason, we set the cost of deleting a component to be the same than the cost of deleting a connection, $D_e = D_v(a)$. As a final conclusion, we propose the default combination of values: $D_e = 10$, $D_v(a) = 10$ and $S_v(a) = 1$.

4.3. Database of Engineering Drawings

We have used four sheets that compose a P&ID presented in the experiments of [11]. As explained in that paper, this P&ID is anonymised real data from an industrial partner. They presented a method to validate the P&IDs in a semi-automatic way. The data we present is the ground-truth data they used to validate their method[4]. Figure 2 shows one of these sheets in our application.

Table 1 shows the number of components and pipelines in each P&ID sheet in the first two rows. The total number of components and pipelines appear in the last column. Moreover, the number of different types of components (which is the same as the number of different Labels in the nodes) appear in the third

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

row. Note that many types of components appear in several sheets, for this reason the sum of the sheets' labels is not 39. Moreover, Table 2 shows the six most common types of components with their number of repetitions in each sheet.

	Sheet1	**Sheet2**	**Sheet3**	**Sheet4**	**Total**
Components	203	165	261	157	786
Pipelines	360	228	369	164	1121
Labels	23	11	25	19	39

Table 1: Number of components, pipelines and types of components in the four sheets.

4.4. Two examples of sub-graph detection

Before the analysis of the quality of our method in Section 4.5, we show two examples of sub-graph detection to clarify the method. In the first example, the left graph in Figure 5 has been queried in Sheet 2 and shown in Figure 6. In the second example, the right graph in Figure 5 has been queried in Sheet 3 and shown in Figure 7. In both examples, K has been set to 10.

All edit costs in this section and in Section 4.5 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 1). And, for the substitution costs, $S_v(a, b) = 1$ if $a \neq b$ and $S_v(a, a) = 0$, $1 \leq a, b \leq 39$.

Table 3 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 mappings, respectively. For the rest of the table, if the cell in i^{th} row and

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

Table 2: Most common types of components in the four P&IDs (or different Labels in the graphs).

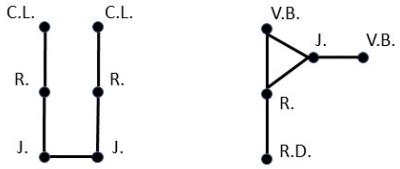


Figure 5: Queries searched in Sheets 2 and Sheet 3 of our database, respectively. Notation: C.L.: Continuity label, R.: Reducer, R.D.: Rupture Disc, V.B.: Valve ball, J.: Junction.

	$\boldsymbol{Cost_i}$ (**Eq. 1**)	$\boldsymbol{v_1 = C.L.}$	$\boldsymbol{_2 = R.}$	$\boldsymbol{v_3 = J.}$	$\boldsymbol{v_4 = J.}$	$\boldsymbol{v_5 = R.}$	$\boldsymbol{v_6 = C.L.}$
$\boldsymbol{f_1}$	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 3: 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.

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 label 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.

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 labels are different. This configuration forces the label of each query node in Q to be the same as the label of its image node in G and also the pipelines to be located in the same positions.

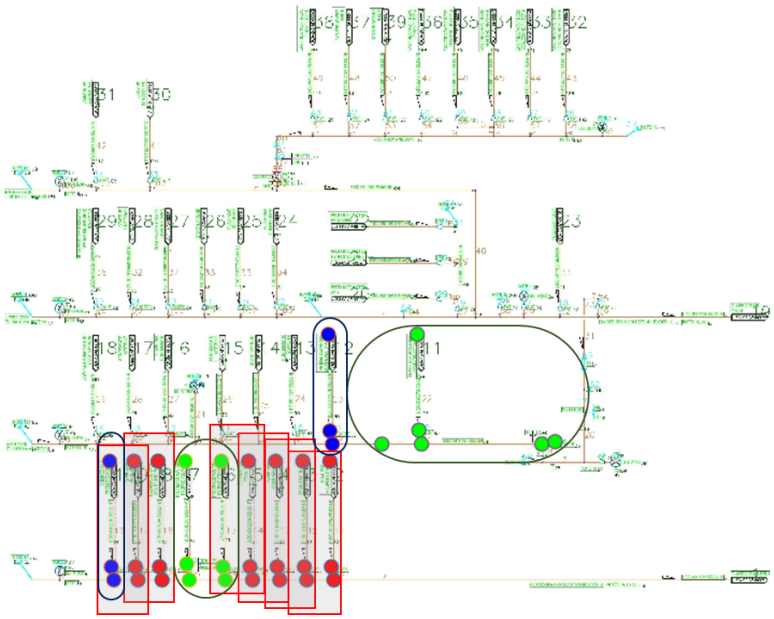


Figure 6: A screen shot of the first example.

This is seen in the table since the labels 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 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 7 shows a screen shot of the second example. In this case, we do not show the specific mappings, as it is done in Table 3 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 there are nodes that are involved in several mappings.

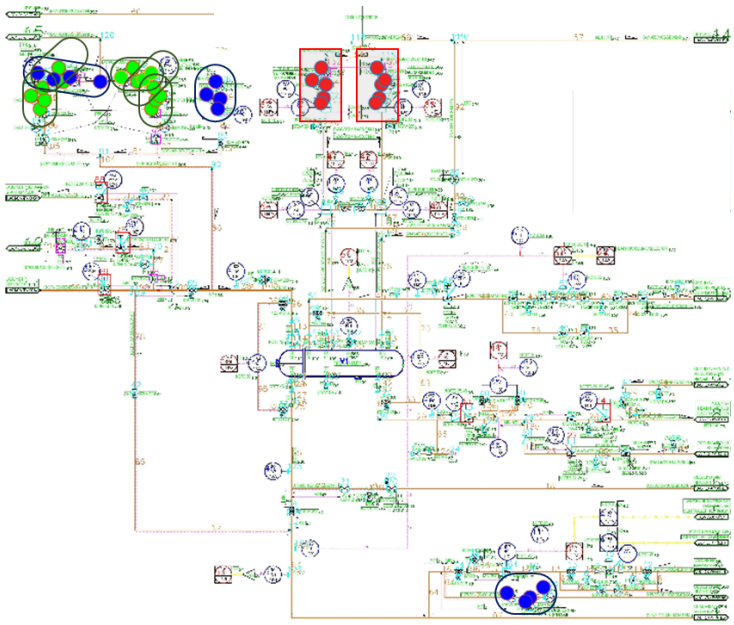


Figure 7: A screen shot of the second example.

4.5. Recall: The metric of the quality

We have used the *Recall* to analyse our method, as it is usual in the retrieval applications.

$$Recall = \frac{Number\ of\ relevant\ retrieved\ sub\text{-}graphs}{Number\ of\ relevant\ sub\text{-}graphs\ in\ the\ P\&ID} \tag{5}$$

The relevant retrieved sub-graphs are the sub-graphs returned by the system that have a cost lower than a threshold when compared to the query. The

number of relevant sub-graphs in the P&ID are the ones that have a cost lower than the same threshold when compared to the query. This threshold is a parameter independent of our method and it is defined to decide which returned sub-graphs are considered correct and which are not. Note, being under or lower this threshold is not bad or good, by itself. The threshold is only needed to perform a fair normalisation. More specifically,

$$Recall(Q, G, K, Cost_{max}) = \frac{|\{f_p : Q \to G, p = 1...K \; s.t. \; Cost(Q, G, f_p) \leqslant Cost_{max}\}|}{|\{f : Q \to G \; s.t. \; Cost(Q, G, f) \leqslant Cost_{max}\}|} \tag{6}$$

Where $Cost_{max}$ is the previously commented threshold. Note the numerator depends on K and $Cost_{max}$ but the denominator only depends on $Cost_{max}$. Thus, *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.

Figure 8 shows the average *Recall* obtained in the four sheets of the database given 10 different query sub-graphs and three different thresholds, $Cost_{max}$. 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 labels is allowed. In the cases that $Cost_{max} = 2$ or $Cost_{max} = 3$ further combinations of edit costs are allowed that have a cost lower or equal to $Cost_{max}$.

If we fix parameter K, the value of the numerator and the denominator only depends on parameter $Cost_{max}$. Moreover, both, the numerator and the denominator, are non-decreasing functions with respect $Cost_{max}$. From the experiment shown in Figure 8 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 8 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,

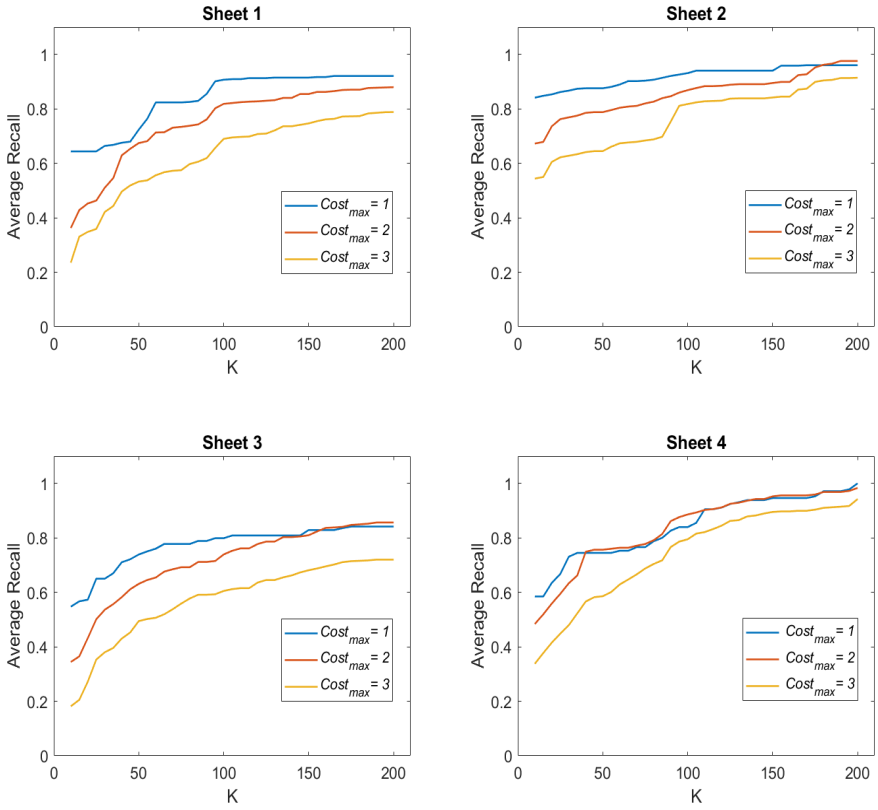


Figure 8: Average Recall returned by our method given 10 queries in the four sheets.

where the engineer has to look up a whole P&ID.

The conclusion of the experimental section is that our method is useful to detect sub-graphs in P&IDs. We first have seen that given different configurations of edit costs, the returned sub-graphs have different properties and for this reason, these edit costs are parameters that the user has to deal with. Then, we have shown two examples of queries in two different sheets. We have seen that we have a friendly application in which it is easy to visualise the queried sub-graphs. Finally, we have visualised the *Recall*, which is a common metric in database retrieval. We have seen its dependencies on parameter K, and therefore, a parameter that the user also has to deal with.

In Figure 9, we show average runtimes in seconds of our algorithm (Matlab 2020a and Intel Core i7). We realise that the runtime is linear with respect to K although the increment is not very important.

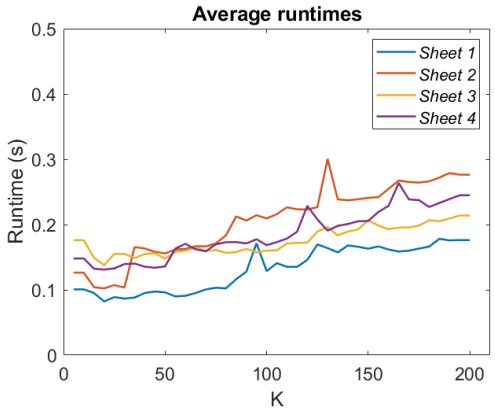


Figure 9: Average runtimes of our method.

5. Conclusions and Future Work

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. This paper proposes a method that solves this problem with the aim of adding this functionality to a CAD application. The method has been framed in the graph theory. Since graph matching is an NP problem, heuristic algorithms have been used to achieve an acceptable runtime at the cost of not obtaining the optimal results.

Our method has been applied to four piping and instrumentation diagrams, which had been previously published, and we show some examples of sub-graph detection. We do not compare our method to other ones since we have seen that none of the published ones can be adapted to our problem. On the one hand, the other graph matching algorithms cannot be applied since they return non-connected sub-graphs. On the other hand, the other Top-K searching sub-

graph algorithms cannot be used since they return only an exact match or they are based on the feature that close nodes tend to have similar properties.

The next step in our work will be the implantation of the beta version of our code in a CAD application. Moreover, we also want to add other functionalities, which are related to searching for a set of components. For instance, the automatic replacement in the CAD sheets of some components that are part of the returned sub-graphs.

Acknowledgements

This research is supported by Martí i Franquès PhD grants program from Universitat Rovira i Virgili.

References

[1] C. Howie, J. Kunz, T. Binford, T. Chen, K. H. Law, Computer Interpretation of Process and Instrumentation Drawings, Advances in Engineering Software 29 (7-9) (1998) 563–570. doi:10.1016/S0965-9978(98)00022-2.

[2] W. C. Tan, I. M. Chen, H. K. Tan, Automated identification of components in raster piping and instrumentation diagram with minimal pre-processing, IEEE International Conference on Automation Science and Engineering November (2016) 1301–1306. doi:10.1109/COASE.2016.7743558.

[3] C. F. Moreno-García, E. Elyan, C. Jayne, Heuristics-Based Detection to Improve Text / Graphics Segmentation in Complex Engineering Drawings, in: Engineering Applications of Neural Networks, Vol. CCIS 744, 2017, pp. 87–98.

[4] R. Rahul, S. Paliwal, M. Sharma, L. Vig, Automatic Information Extraction from Piping and Instrumentation Diagrams, in: International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2019, pp. 163–172. arXiv:1901.11383, doi:10.5220/0007376401630172.
URL http://arxiv.org/abs/1901.11383

[5] M. Rantala, H. Niemistö, T. Karhela, S. Sierla, V. Vyatkin, Applying graph matching techniques to enhance reuse of plant design information, Computers in Industry 107 (2019) 81–98. doi:10.1016/j.compind.2019.01.005.
URL https://doi.org/10.1016/j.compind.2019.01.005

[6] S.-O. Kang, E.-B. Lee, H.-K. Baek, A digitization and conversion tool for imaged drawings to intelligent piping and instrumentation diagrams (P&ID), Energies 12 (2593) (2019) 1–26. doi:10.3390/en12132593.

[7] L. P. Cordella, M. Vento, Symbol recognition in documents: a collection of techniques?, International Journal on Document Analysis and Recognition 3 (2) (2000) 73–88.

[8] K. Tombre, et al., Graphics Recognition: Algorithms and Systems: Second International Workshop, GREC'97, Nancy, France, August 22-23, 1997, Selected Papers, Vol. 2, Springer Science & Business Media, 1998.

[9] C. F. Moreno-García, E. Elyan, Digitisation of Assets from the Oil & Gas Industry: Challenges and Opportunities, in: International Conference on Document Analysis and Recognition (ICDAR), no. Workshop on Industrial Applications of Document Analysis and Recognition (WIADAR), 2019, pp. 16–19. doi:10.1109/ICDARW.2019.60122.

[10] C. F. Moreno-García, E. Elyan, C. Jayne, New trends on digitisation of complex engineering drawings, Neural Computing and Applications 31 (6) (2019) 1695–1712. doi:10.1007/s00521-018-3583-1.

[11] E. Rica, C. F. Moreno-García, S. Álvarez, F. Serratosa, Reducing human effort in engineering drawing validation, Comput. Ind. 117 (2020) 103198. doi:10.1016/j.compind.2020.103198.
URL https://doi.org/10.1016/j.compind.2020.103198

[12] A. Chaudhuri, K. Mandaviya, P. Badelia, S. K. Ghosh, Optical character recognition systems, in: Optical Character Recognition Systems for Different Languages with Soft Computing, Springer, 2017, pp. 9–41.

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

[14] H. Bunke, G. Allermann, Inexact graph matching for structural pattern recognition, Pattern Recogn. Lett. 1 (4) (1983) 245–253. doi:10.1016/0167-8655(83)90033-8.
URL http://dx.doi.org/10.1016/0167-8655(83)90033-8

[15] A. Sanfeliu, K.-S. Fu, A distance measure between attributed relational graphs for pattern recognition, Systems, Man and Cybernetics, IEEE Transactions on SMC-13. doi:10.1109/TSMC.1983.6313167.

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

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

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

[19] F. Serratosa, Graph edit distance: Restrictions to be a metric, Pattern Recognition 90 (2019) 250–256. doi:10.1016/j.patcog.2019.01.043.
URL https://doi.org/10.1016/j.patcog.2019.01.043

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

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

[22] F. Serratosa, Fast computation of bipartite graph matching, Pattern Recognition Letters 45 (2014) 244–250.

[23] F. Serratosa, Speeding up fast bipartite graph matching through a new cost matrix, International Journal of Pattern Recognition and Artificial Intelligence 29 (2014) 1550010. doi:10.1142/S021800141550010X.

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

[25] K. Riesen, A. Fischer, H. Bunke, On the impact of using utilities rather than costs for graph matching, Neural Processing Letters 48 (2) (2018) 691–707. doi:10.1007/s11063-017-9739-7.
URL https://doi.org/10.1007/s11063-017-9739-7

[26] F. Serratosa, X. Cortés, Graph edit distance, Pattern Recogn. Lett. 65 (C) (2015) 204–210. doi:10.1016/j.patrec.2015.08.003.
URL https://doi.org/10.1016/j.patrec.2015.08.003

[27] P. Santacruz, F. Serratosa, Error-tolerant graph matching in linear computational cost using an initial small partial matching, Pattern Recognition Letters.

[28] L. Zou, L. Chen, Y. Lu, Top-k subgraph matching query in a large graph, in: A. S. Varde, J. Pei (Eds.), 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, ACM, 2007, pp. 139–146. doi:10.1145/1316874.1316897.
URL https://doi.org/10.1145/1316874.1316897

[29] Z. Yang, A. W.-C. Fu, R. Liu, Diversified top-k subgraph querying in a large graph, in: Proceedings of the 2016 International Conference on Management of Data, SIGMOD '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 1167–1182. doi:10.1145/2882903.2915216.
URL https://doi.org/10.1145/2882903.2915216

[30] G. Gou, R. Chirkova, 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, Association for Computing Machinery, New York, NY, USA, 2008, p. 581–594. doi:10.1145/1376616.1376676.
URL https://doi.org/10.1145/1376616.1376676

[31] W. Fan, X. Wang, Y. Wu, Diversified top-k graph pattern matching, Proc. VLDB Endow. 6 (13) (2013) 1510–1521. doi:10.14778/2536258.2536263.
URL https://doi.org/10.14778/2536258.2536263

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

[33] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, S. Tao, Neighborhood based fast graph search in large networks, in: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, SIGMOD '11, Association for Computing Machinery, New York, NY, USA, 2011, p. 901–912. doi:10.1145/1989323.1989418.
URL https://doi.org/10.1145/1989323.1989418

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